YU ISSN 0350-5596

gorenje informatika '83

Industrija
prosto programirani sistemi
procesni mikroračunalniški sistemi

Komunikacije s prenosom podatkov
teletekst
videotekst

Informacijski sistemi
Obrambni in varnostni sistemi
Medicinska elektronika in informatika

Mali poslovni sistemi
sodelovanje z Birostrojem

Potrošna elektronika
Terminali
Paka 2000
Mikroračunalniške enote in moduli

informatica

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 7, 1983 – Št. 4

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

UREDNIŠKI ODBOR:

T. Aleksić, Beograd; D. Bitrakov, Skopje; P. Dragojlović, Rijeka; S. Hodžar, Ljubljana; B. Horvat, Maribor; A. Mandžić, Sarajevo; S. Mihalić, Varaždin; S. Turk, Zagreb

GLAVNI IN ODGOVORNI UREDNIK: Anton P. Železnikar

TEHNIČNI ODBOR:

V. Batagelj, D.Vitas -- programiranje
I. Bratko -- umetna inteligenca
D. Ćećez-Kecmanović -- informacijski sistemi
M. Exel -- operacijski sistemi
B. Džonova-Jerman-Blažič -- srečanja
L. Lenart -- procesna informatika
D. Novak -- mikroračunalniki
Neda Papić -- pomočnik glavnega urednika
L. Pipan -- terminologija
V. Rajkovič -- vzgoja in izobraževanje
M. Špegel, M. Vukobratović -- robotika
P. Tancig -- računalništvo v humanističnih in družbenih vedah
S. Turk -- materialna oprema
A. Gorup -- urednik v SOZD Gorenje

TEHNIČNI UREDNIK: Rudolf Murn

ZALOŽNIŠKI SVET:

T. Banovec, Zavod SR Slovenije za statistiko, Vožarski pot 12, Ljubljana
A. Jerman-Blažič, DO Iskra Delta, Parmova 41, Ljubljana
B. Klemenčič, Iskra Telematika, Kranj
S. Saksida, Institut za sociologijo Univerze Edvarda Kardelja, Ljubljana
J. Virant, Fakulteta za elektrotehniko, Tržaška 25, Ljubljana

UREDNIŠTVO IN UPRAVA: Informatica, Parmova 41, 61000 Ljubljana; telefon (061) 312-988; teleks 31366 YU Delta

LETNA NAROČNINA za delovne organizacije znaša 1900 din, za redne člane 490 din, za študente 190 din; posamezna številka 590 din.
ŽIRO RAČUN: 50101-678-51841

Pri financiranju časopisa sodelůje 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

VSEBINA

informatica

JOURNAL OF COMPUTING AND INFORMATICS

YU ISSN 0350–5596

VOLUME 7, 1983 – No 4

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

EDITORIAL BOARD:

T. Aleksić, Beograd; D. Bitrakov, Skopje; P. Dragojlović, Rijeka; S. Hodžar, Ljubljana; B. Horvat, Maribor; A. Mandžić, Sarajevo; S. Mihalić, Varaždin; S. Turk, Zagreb

EDITOR-IN-CHIEF: Anton P. Železnikar

TECHNICAL DEPARTMENTS EDITORS:

V. Batagelj, D. Vitas -- Programming
I. Bratko -- Artificial Intelligence
D. Ćećez-Kecmanović -- Information Systems
M. Exel -- Operating Systems
B. Džonova-Jerman-Blažič -- Meetings
L. Lenart -- Process Informatics
D. Novak -- Microcomputers
Neda Papić -- Editor's Assistant
L. Pipan -- Terminology
V. Rajkovič -- Education
M. Špegel, M. Vukobratović -- Robotics
P. Tancig -- Computing in Humanities and Social Sciences
S. Turk -- Computer Hardware
A. Gorup -- Editor in SOZD Gorenje

EXECUTIVE EDITOR: Rudolf Murn

PUBLISHING COUNCIL:

T. Banovec, Zavod SR Slovenije za statistiko, Vožarski pot 12, Ljubljana
A. Jerman-Blažič, DO Iskra Delta, Parmova 41, Ljubljana
B. Klemenčič, Iskra Telematika, Kranj
S. Saksida, Institut za sociologijo Univerze Edvarda Kardelja, Ljubljana
J. Virant, Fakulteta za elektrotehniko, Tržaška 25, Ljubljana

HEADQUARTERS: Informatica, Parmova 41, 61000 Ljubljana, Yugoslavia
Phone: 61-312-988; Telex: 31366 YU DELTA

ANNUAL SUBSCRIPTION RATE: US$ 22 for companies, and US$ 10 for individuals

Opinions expressed in the contributions are not necessarily shared by the Editorial Board

PRINTED BY: Tiskarna Kresija, Ljubljana

DESIGN: Rasto Kirn

CONTENTS

AN ANALYSIS OF ARITHMETIC EXPRESSIONS BASED ON VECTOR-GENERATRICE CONCEPT

DUŠAN VELAŠEVIĆ

ELEKTROTEHNIČKI FAKULTET, BEOGRAD

UDK: 514.742

Various aspects of analysis of arithmetic expressions in multiregister environment are considered: a)determination of the number of general-purpose registers necessary for the code generation without store commands,b)generation of allocation and deallocation sequences for registers and memory locations before the code generation is started,c)target register control which directs the result of an evaluation to a predefined register,d)cost function analysis intended to control the load distribution among the processors in the case of parallel processing,e)parallel evaluation in multiprocessor machines without control of load distribution,and with control of load distribution exercised through the value of the cost function.To make the analysis easier and more efficient, the vector-generatrice is transformed and reduced to a more concise form.The implementation of some aspects of analysis is also discussed.

JEDNA ANALIZA ARITMETIČKIH IZRAZA ZASNOVANA NA KONCEPTU VEKTOR-GENERATRISE.U radu se razmatraju razni aspekti analize aritmetičkih izraza u višeregistarskim sistemima:a)odredjivanje broja registara opšte namene za generisanje koda bez naredbi upisivanja,b)generisanje sekvenci dodeljivanja i oslobadjanja registara i memorijskih lokacija,c)kontrola ciljnog registra kojom se rezultat izračunavanja upućuje u unapred zadati registar,d)izračunavanje funkcije koštanja generisanog koda, e)paralelno izračunavanje izraza u multiprocesorskim mašinama.Celokupna analiza je sprovedena na osnovu koncepta vektor-generatrise.Realizacija nekih aspekata analize je takodje diskutovana.

1.INTRODUCTION

Arithmetic expressions were the subject of various analysis:1)algorithms for code generation,2)code optimization at the single expression level,3)code optimization at the level of a group of expressions,4)determination of the number of registers required to compute an expression,5)register and temporaries allocation problem,6)evaluation of the cost function,7)translating arithmetic expressions into the code for parallel computers,and 8)assigning tasks to parallel processors.We shall concentrate in this paper on problems 4),5),6) and 8).

The determination of the number of registers required to compute an expression has been investigated by Nakata(1967),Redziejowski(1969) and Sethi and Ullman(1970).Sethi and Ullman (1970) use the binary tree representation of an expression to determine the number of registers necessary to generate the code without STORE instructions.This approach requires the labeling of syntax tree.Then,no STORE instructions will be generated if the label of the root is less than the number of available registers. Sethi and Ullman(1970) also give the cost function analysis of expressions by introducing the concept of major,minor and interior nodes whose sum represents the cost of a syntax tree.This cost is a reasonable measure of the complexity of a syntax tree,in that the number of instructions needed to compute a syntax tree is equal to the cost of the tree.

The register allocation problem was discussed by many authors.Belady(1966) has formulated one heuristic technique for making register assignments in a sequence of expression evaluations.He has shown that this technique is optimal in some situations.However,the model assumed by this algorithm(which was designed for paging) does not exactly model the straight-line code. In particular,it assumes the order of computation to be fixed while an advantage may be achieved by reordering computations.A similar register allocation problem is discussed by Horwitz at al.(1966).They assume that we are given a sequence of operations which reference and change values.The problem is to assign these values to registers so that the number of loads and stores from the register to main memory is minimized.Their solution is to select a least-cost path in a dag of possible solutions.Techniques for reducing the size of the dag are given. Further investigation of register allocation where order of computation is not fixed has been done by Kennedy(1972) and Sethi(1972).

The general problem of assigning tasks optimally to parallel processors is very difficult. The problem is to generate code which takes advantage of this parallelism.Allard et al.(1964) describe their attempts at this type of optimization of arithmetic statements.The major methods used are reordering the operations,eliminations of redundant loads and stores,and assignment of registers.Hellerman(1966) and Squire (1963) also report on multipass algorithm for compiling expressions for parallel processors. Stone(1967) gives a one-pass algorithm which transforms an arithmetic expression into parenthesized internal form suitable for parallel computation.Further investigation of parallel computation has been done also by Baer and Bovet(1968) and Graham(1972).

In this paper,we investigate the application of the vector-generatrice concept(Velašević, 1982) to solving discussed problems.As the first step towards this goal,the vector-generatrice(VG) is mapped to a reduced and transformed form(RTVG),more suitable for further considerations.RTVG can be obtained as a side effect of syntax analysis,or separately,by mapping VG upon its generation.Three essential definitions related to handling of RTVG are made:1)observation point,2)current state of an observation point,and 3)terminating state of an observation point.

2.VECTOR-GENERATRICE

2.1 Transformation to vector-generatrice

Let E_i be an arithmetic expression in the infix notation transformed to postfix form E_p:

$$E_p=a_1a_2\cdots a_mb_1b_2\cdots b_ja_{m+1}a_{m+2}\cdots a_{m+i}b_{j+1}b_{j+2}\cdots b_{j+k}\cdot a_{m+i+n}a_{m+i+n+1}\cdot a_{m+i+M}b_{j+k+q}b_{j+k+q+1}\cdots b_{j+k+N}a_{m+i+M+1} \quad (1)$$

where $a_1,a_2,a_3...a_m,..a_{m+i+M+1}$ are the operands (or their addresses) and $b_1,b_2,b_3,...b_j,...b_{j+k+N}$ the operators(or their indices in delimiter table).The dollar sign $ denotes the end of E_p,

$$a_{m+i+M+1}=\$ \quad (2)$$

Using Eqn(1),VG can be generated in the following way.Firstly,we define the substrings of E_p. A substring is a sequence of all successive operands or operators.If the number of operand substrings is N_1,the number of operator substrings N_2 must be

$$N_2=N_1 \quad (3)$$

The substrings of E_p in Eqn(1) are:

$a_1a_2..a_m$ (operands); $b_1b_2..b_j$ (operators),
$a_{m+1}a_{m+2}..a_{m+i}$(operands)$b_{j+1}b_{j+2}..b_{j+k}$(operators)
.....
$a_{m+i+M+1}$ (operand)

The total number of substrings N_3(including the endmarker as an operand substring) is

$$N_3=2N_1+1 \quad (4)$$

By assigning to each substring of operands(OPD) a successor substring of operators(OPR),one obtains a pair of connected substrings CS_ℓ,where ℓ is its ordinal number.For example,$a_1a_2..a_mb_1b_2..b_j$ forms a pair of connected substrings CS_ℓ, $a_1a_2..a_m$ represents a substring of operands(OPD_1) followed by a successor substring of operators(OPR_1).The number of pairs is

$$N_{CS}=\lfloor N_3/2\rfloor=N_1 \quad (5)$$

For each pair of connected substrings we create a 3-tuple $VG_\ell=(N^O_\ell,N^B_\ell,P_\ell)$,where N^O_ℓ number of operands in CS_ℓ,P_ℓ pointer to beginning of CS_ℓ in E_p,N^B_ℓ number of operators in CS_ℓ.For CS_1,3-tuple is(m,j,1) because $N^O_1=m$,$N^B_1=j$ and P=1.VG is formed now by concatenating all 3-tuples of connected substrings by their ordinal numbers into a vector of $(N_{CS}+1)x3$ dimension:

$$VG_1catVG_2cat...catVG_{N_{CS}+1} \quad (6)$$

The endmarker is considered as an operand substring connected to a dummy operator substring. Its corresponding 3-tuple is $(1,0,P_{N_{CS}+1})$.

2.2 Covers in connected substrings

Consider a pair of connected substrings CS_ℓ. The value for N^O_ℓ and N^B_ℓ may satisfy one of the following conditions:

$$a)\ N^O_\ell\le N^B_\ell-N^B_\ell(unary)\ ;\ b)\ N^O_\ell>N^B_\ell-N^B_\ell(unary) \quad (7)$$

where $N^B_\ell(unary)$ denotes the number of unary operators in OPR_ℓ.

If the condition a) is satisfied,then N^O_ℓ operands are fully covered by N^B_ℓ in its own CS_ℓ, i.e.,the evaluation of CS_ℓ gives one resulting operand and a certain surplus of operators.For CS_ℓ with such a property we say that it has the full own cover.Two distinctions should be made in the case a).Firstly,for $N^O_\ell=1\wedge N^B_\ell-N^B_\ell(unary)\neq 0\wedge OPR_\ell(1)\neq$ - (- unary minus),the resulting operand will reside in a memory location.Secondly, in all other situations the resulting operand will be in a register.The surplus of operators forms the remote cover,and is used for covering another CS_i,$(i<\ell)$ not satisfying the condition a).

If the condition b) is satisfied,then only $N^B_\ell-N^B_\ell(unary)+1$ operands are covered in its own CS_ℓ.For CS_ℓ with such a property we say that it has the partial own cover.The surplus of operands will be covered by one or more remote covers of CS_is$(i<\ell)$ to obtain the full cover of CS_ℓ.

3.MAPPING OF VECTOR-GENERATRICE

In performing the mapping operation,each tuple of VG and each pair of connected substrings,CS_ℓ,has to be analyzed starting from the right end of VG.Generally,a triple of VG,VG_ℓ,is mapped to another triple,$RTVG_1$,composed of the following components:

$$D_1=f_1(N^O_\ell,N^B_\ell) \quad (8)$$

representing mostly the difference between the number of operands and binary operators in a CS_ℓ

$$N^{in}_1=f_2(N^B_\ell) \quad (9)$$

giving the number of noncommutative operators in the full or partial own cover of a CS_ℓ,and

$$N^{out}_1=f_3(N^B_\ell,N^B_{\ell+1},N^B_{\ell+2},...) \quad (10)$$

counting the number of noncommutative operators in the remote cover of other connected substrings which make the full cover of a CS_ℓ.

Consider now the mapping operation by investigating the properties of triples in VG.

Property No.1

If the following holds:

$N^O_\ell=1$ (11);$N^B_\ell-N^B_\ell(unary)=1$ (12);$OPR_\ell(1)\neq$ - (13)

then

$f_1=\xi$(14);$f_2=\xi$(15);$f_3=\xi$(16)

where ξ is an empty symbol.Eqns (14),(15) and (16) imply that the triple under consideration has to be excluded from RTVG because it does contribute neither to allocation nor deallocation of registers.

Property No.2

If Eqn (11) holds,and if

$$N^B_\ell-N^B_\ell(unary)=0 \quad (17)$$

then

$f_1=1$ (18);$f_2=0$ (19);$f_3=0$ (20)

It is evident from Property No.2 that such a triple will require the allocation of a register.Because all unary operators are consumed by a single operand,f_1 is set to unity indicating in this way that a register is allocated without the residual of operators.

Property No.3

If Eqns(11) and (13) hold,and if

$$N^B_\ell-N^B_\ell(unary)>1 \quad (21)$$

then Eqns (14),(15) and (16) are valid because the triple does not generate a request for the allocation of a register.However,the triple contributes to the allocation and deallocation process in connectionwith lower-numbered triples, due to possible noncommutative operators in OPR_ℓ. This fact is taken into account by concatenating $OPR_\ell(2)OPR_\ell(3)..OPR(N^B_\ell)$ to $OPR_{\ell-1}$.The concatenation is started from $OPR_\ell(2)$because the operator $OPR_\ell(1)$ does contribute neither to allocation nor deallocation of registers.If more successive triples with Property No.3 occur,for example,$VG_\ell,VG_{\ell+1}...VG_{\ell+k}$,then it holds

$$OPR_{\ell-1}=OPR_{\ell-1}catOPR_\ell(2)..catOPR_\ell(N^B_\ell)catOPR_{\ell+1}(2)..catOPR_{\ell+1}(N^B_{\ell+1})..catOPR_{\ell+k}(2)..OPR_{\ell+k}(N^B_{\ell+k}) \quad (22)$$

The realization of the concatenation will be performed only logically,not physically,thus eliminating the overhead.The concatenation of two operator substrings from triples $VG_{\ell-1}$ and VG_ℓ may cause the change of the property of $VG_{\ell-1}$. Thus,when a triple is checked for a property it must be examined whether the next higher-numbered triples have Property No.3 in order to determine the resulting property of the triple under consideration.

Property No.4

If Eqn (11) is valid,and if

$N^B_\ell-N^B_\ell(unary)\geq 1$ (23);$OPR_\ell(1)=$ - (24)

then Eqn (19) holds,and

$f_1=-(N^B_\ell-N^B_\ell(unary))$ (25);$f_3=N^B_\ell(noncommutative)$ (26)

where $N^B_\ell(noncommutative)$ designates the number of noncommutative operators in the operator substring of a CS_ℓ.The function f_1 is set to this value because VG_ℓ requires the allocation of a register and $N^B_\ell-N^B_\ell(unary)$ operators are left unconsumed.The function f_3 is set to the number of noncommutative operators in OPR_ℓ to

help the determination of the allocation and deallocation pattern.

Property No.5

If the following is true:

$N_\ell^O>1$ (27); $N_\ell^O>N_\ell^B-N_\ell^B$(unary) (28), then

$f_1=N_\ell^O-(N_\ell^B-N_\ell^B(\text{unary}))$ (29)

$f_2=N_\ell^B(\text{noncommutative})$ (30)

$f_3=\sum_{\ell'<\ell} N_{\ell'}^B(\text{noncommutative in remote cover})$ (31)

The function f_1 is set to this value because $N_\ell^B-N_\ell^B(\text{unary})+1$ operands are consumed leaving an operand as a result in a register. This surplus of operands will be consumed by higher-numbered triples having $f_1<0$ because the triple with Property No.5 changes the moving in VG space from left to right. In this way, the covering of the triple with Property No.5 is provided. If $f_2>0$, then this triple will require an additional register thus increasing the maximum number of allocated registers to two. Upon the exit from the triple, the total number of allocated registers will amount one for the allocation of an additional register is only temporary. Because the operands are partially covered in their own triple, the function f_3 must give the number of noncommutative operators in all remote covers of CS_ℓ ($\ell'<\ell$) participating in full covering of CS_ℓ.

Property No.6

If Eqn (27) holds, and if

$N_\ell^O\leq N_\ell^B-N_\ell^B(\text{unary})$ (32), then

$f_1=N_\ell^O-(N_\ell^B-N_\ell^B(\text{unary}))-1$ (33)

$f_2=N_\ell^B(\text{noncommutative in full own cover})$ (34)

$f_3=N_\ell^B(\text{noncommutative out of full own cover})$ (35)

The value of the function f_1 is the number of unconsumed operators. These operators will be consumed by operands in triples with Property No.5. Similarly to Property No.5, if $f_2>0$, then an additional register must be temporary allocated.

Property No.7

If Eqn (11) holds, and if

$N_\ell^B=0$ (36), then

$f_1=0$ (37); $f_2=-1$ (38); $f_3=0$ (39)

In fact, this is the mapping of the endmarker of VG, which becomes now the startmarker of RTVG.

4. MEMORYLESS CODE GENERATION

The functions f_1 and f_2 possess enough information to make possible the determination of the number of registers for the memoryless code generation. If $f_2=0$ in all triples of RTVG, then that number can be found on the basis of f_1. As the first step toward this goal, we formulate the following theorem:

THEOREM I. For an arbitrary E_R with VG of $N_{CS}+1$ triples, the upper limit of the number of registers required for the memoryless code generation is equal to $N_{CS}+1$.

Proof: The worst case with respect to register allocation is that one in which E_R and VG produce RTVG satisfying the conditions:

$RTVG[N_{CS}*3+1]>0$ (40); $RTVG[l*3+1]<0, l=1,2..N_{CS}-1$ (41)

$RTVG[N_{CS}*3+2]\neq 0$ (42)

For each function f_1 satisfying Eqn (41) a register must be allocated. This amounts $N_{CS}-1$ registers. A register must be allocated also for the last function f_1, giving the total of N_{CS} registers. If Eqn (42) is satisfied, CS_1 contains a noncommutative operator in its own partial cover and an additional register must be allocated. This gives the total of $N_{CS}+1$ registers.

COROLLARY. If THEOREM I is valid and if exists a triple satisfying the conditions:

$RTVG[(l-1)*3+1]>0$, $3\leq l\leq N_{CS}$ (43)

$RTVG[(l-1)*3+2]\neq 0$, $3\leq l\leq N_{CS}$ (44)

the upper limit of the number of registers required for the memoryless code generation up to this triple included is l, and upon its handling at most $l-2$.

Proof: Because RTVG triple with ordinal number l satisfies the same conditions as expressed by Eqns (40) and (42) with the exception that N_{CS} should be replaced by $l-1$, the upper limit must be l. During the handling of the corresponding triple, at least two deallocations occur: 1) deallocation due to noncommutative operator in the own partial cover of the triple, and 2) one or more deallocations due to moving right in VG space. It comes out that after the handling of the triple at the point l the total of allocated registers amounts at most $l-2$.

In order to find out the actual number of registers, we shall consider RTVG at the observation points defined by Eqn(43) for $\forall l\in 2,3..L$, where L denotes the number of triples in RTVG. The distance of two successive observation points is given by $d=l_2-l_1$, where l_2 and l_1 are their ordinal numbers. We define two states in these points: 1) current, and 2) terminating. The observation point is in the current state when its corresponding CS_ℓ is reached and the partial own cover handled. The observation point is in the terminating state during the moving right in RTVG space. We can state now the following:

THEOREM II. The total number of registers needed for the memoryless code generation up to an observation point is given by

$R_{l_{obs}}^{A,C}=R_{l_{pobs}}^{A,T}+d+sign(RTVG[(l_{obs}-1)*3+2])$

$R_{l_{obs}}^{A,C}$ - total number of registers allocated up to the current state of an observation point(l_{obs})

$R_{l_{pobs}}^{A,T}$ - total number of registers allocated upon the termination of the terminating state of the previous observation point(l_{pobs})

$$sign(X)=\begin{cases}1, X>0\\ 0, X\leq 0\end{cases}$$

Proof: It is easily seen that $d+sign(RTVG[(l_{obs}-1)*3+2])$ represents the number of registers which should be allocated between two successive points in order to achieve the memoryless code generation. If $RTVG[(l_{obs}-1)*3+2])>0$, then an additional register is needed due to noncommutative operators. Otherwise, d registers should be allocated. We shall designate this number by $R_d^{A,C}$. Upon the handling in the current state, the terminating state begins thus causing the deallocation process. Upon the termination of the deallocation process, the number of allocated registers at the previous observation point is

$$R_{l_{pobs}}^{A,T}=R_{l_{pobs}}^{A,C}-R_{l_{pobs}}^{D,T} \quad (45)$$

where D designates the deallocation. Thus, the total number of registers allocated up to the current state of the observation point l_{obs} is

$$R_{l_{obs}}^{A,C}=R_{l_{pobs}}^{A,T}+R_d^{A,C} \quad (46)$$

as stated by THEOREM II. For convenience, we have assumed that $R_{l_{pobs}}^{D,T}$ comprises the register allocated and deallocated in the current state due to noncommutative operators.

THEOREM III. The number of registers deallocated in the terminating state of an observation point is equal to the number of RTVG triples left to the observation point having $D_l\leq -1$ and satisfying the condition $|\sum_{i=1}^{n} D_l - 1| - n \geq D_{l_n}$, where n is the number of RTVG triples with $D_l\leq -1$ left to the observation point.

Proof: The moving right in VG space will cause the deallocation of registers. If $D_l=-1$, then we have one register-to-register operation which deallocates one register allocated in D_l. As a consequence, D_l is set to zero indicating that

the binary operator is consumed. We continue in this way until a RTVG triple is encountered having $D_1<-1$. This triple gives the next register deallocation with setting $D_1=D_1+1$. If $D_{1_{obs}}>1$, the consumption of operators in D_1 can be continued until D_1 becomes zero by setting $D_1=D_1+1$ and $D_{1_{obs}}=D_{1_{obs}}-1$ (the deallocation does not occur because the operation is register-to-memory) before $D_{1_{obs}}$ reaches unity (thus enabling further moving right in VG space), or until $D_{1_{obs}}$ reaches unity before D_1 becomes zero (thus inhibiting further moving right in VG space) when we set $D_{1_{obs}}=0$. Evidently, the inhibition occurs at the point satisfying

$$|\Sigma D_1<-1|-n \geq D_{1_{obs}} \qquad (47)$$

Because only one deallocation is possible for each RTVG triple, the total number of deallocated registers is equal to the number of nonzero D_1 left to the observation point and up to the point satisfying Eqn (47).

Starting from Eqn (46), we have

$$R_1^{A,C}=R_0^{A,T}+R_{d_1}^{A,C} \qquad R_2^{A,C}=R_1^{A,T}+R_{d_2}^{A,C}$$
$$R_3^{A,C}=R_2^{A,T}+R_{d_3}^{A,C} \qquad \ldots\ldots\ldots\ldots$$
$$\ldots\ldots\ldots\ldots \qquad R_k^{A,C}=R_{k-1}^{A,T}+R_{d_k}^{A,C} \qquad (48)$$

where k is the enummeration of the observation points and d_k the enummeration of the corresponding distances. Bearing in mind that $R_0^{A,T}=0$ and substituting Eqn (45) in Eqn (48), one obtains

$$R_k^{A,C}=\sum_{i=1}^{k}R_{d_i}^{A,C}-\sum_{i=1}^{k-1}R_i^{D,T} \qquad (49)$$

By THEOREMS I and II we have

$$\sum_{i=1}^{m}R_{d_i}^{A,C}=l_m-1+\sum_{i=1}^{m}\mathrm{sign}(RTVG[(l_i-1)*3+2]) \qquad m=1,2,3,..k \qquad (50)$$

where l_i is the ordinal number of the i-th observation point. Eqn (49) becomes now

$$R_m^{A,C}=l_m-1+\sum_{i=1}^{m}\mathrm{sign}(RTVG[(l_i-1)*3+2])-\sum_{i=1}^{m-1}R_i^{D,T} \qquad m=1,2,3,..k \qquad (51)$$

The number of registers needed for the memoryless code generation can be found from Eqn (52):

$$N_R=\max_{m=1,..k}\{R_m^{A,C}\} \qquad (52)$$

To find N_R it is not necessary to perform calculations at each observation point. This process can be stopped at the observation point satisfying the condition

$$R_m^{D,T} \geq L-l_m+\mathrm{sign}(RTVG[(L-1)*3+2]) \qquad (53)$$

i.e., the number of registers deallocated at the observation point l_m should be greater than or equal to the maximal number of registers eventually requested by the rest of expression.

5. TRACKING OF ALLOCATION AND DEALLOCATION PROCESS

In order to make feasible the tracking of the allocation and deallocation process, we define the allocation and deallocation zones. The deallocation zone is a set of registers successively deallocated without any intervening allocation. The allocation zone is a set of registers successively allocated without any intervening deallocation. The tracking of the allocation and deallocation process can be formulated as the determination of allocation and deallocation zones. From THEOREMS I, II and III one can conclude the following:

a) the allocation zones occur during the transition from one to another observation point, during the moving right in VG space due to eventual noncommutative operators, and in the current state of an observation point also due to eventual noncommutative operators in the partial own cover

b) the deallocation zones occur during the moving right in VG space, due to eventual noncommutative operators in the remote covers and during the transition from one to another observation point due to noncommutative operators in the full own covers.

Consider a pool of stacked available registers, FR, after the handling of an observation point:

$$FR=\{RM,RN,..,RI,..,RJ,RK,RL\} \qquad (54)$$

and the set of allocated registers, AR,

$$AR=\{\ldots RI_1,RJ_1,RL_1,RK_1\} \qquad (55)$$

In order to present the algorithm for tracking of allocation and deallocation process in a concise form, we define the following operations:

STACK(X)→put an item on the stack X
UNSTACK(X)→remove the top entry from the stack X
TOP(X)-take the top entry from X without removing it
OPEN(Z)-open the new zone for allocation or deallocation
CLOSE(Z)-close the current zone Z
DECREMENT(N)-decrement N by one
INCREMENT(N)-increment N by one

Now, we can describe the procedure for tracking of allocation and deallocation process during the transition from one to another observation point:

a) for each RTVG triple satisfying Eqn (41), the operations

UNSTACK(FR)→STACK(AR) TOP(AR)→AZ

are performed; AZ designates the allocation zone

b) N_1^{in} is examined to find noncommutative operators in the full own cover of the triple; for each noncommutative operator the following operations have to be performed:

REPEAT	CLOSE(AZ)
OPEN(DZ)	CLOSE(DZ)
UNSTACK(AR)→DZ	OPEN(AZ)
UNSTACK(FR)→STACK(AR)	DECREMENT(N_1^{in})
DZ→STACK(FR)	EXIT ON(N_1^{in}=0)
TOP(AR)→AZ	ENDREPEAT

where DZ designates the deallocation zone. This sequence of operations is repeated until N_1^{in} becomes zero. Then, new RTVG triple is accessed.

The transition phase terminates at the observation point which is handled in the current state similarly to any triple during the transition. If $N_1^{in}\neq 0$, the following sequence of operations has to be applied:

UNSTACK(FR)→STACK(AR)	TOP(AR)→AZ
TOP(AR)→AZ	CLOSE(AZ)
REPEAT	DECREMENT(N_1^{in})
OPEN(DZ)	EXIT ON(N_1^{in}=0)
UNSTACK(AR)→DZ	CLOSE(DZ)
UNSTACK(FR)→STACK(AR)	OPEN(AZ)
DZ→STACK(FR)	ENDREPEAT

and if $N_1^{in}=0$,

UNSTACK(FR)→STACK(AR)	CLOSE(AZ)
TOP(AR)→AZ	OPEN(DZ)

The essential difference in handling of an observation point is in the action taken over upon the termination of the current state, i.e., new RTVG triple is not accessed but the previous one. The procedure applied in the terminating state is as follows:

a) for each triple belonging to the terminating state perform the sequence of operations

INCREMENT(D_1)	TEMP→STACK(AR)
UNSTACK(AR)→TEMP	TOP(FR)→DZ
UNSTACK(AR)→TOP(FR)	

b) if a triple belonging to the terminating state has $N_1^{out}\neq 0$ and $N_{1_{obs}}^{out}\neq 0$, then perform the following sequence of operations after a):

REPEAT	OPEN(DZ)	DZ→STACK(FR)
CLOSE(DZ)	UNSTACK(AR)→DZ	TOP(AR)→AZ
OPEN(AZ)	UNSTACK(FR)→STACK(AR)	DECREMENT($D_{1_{obs}}$)

INCREMENT(D_1) CLOSE(AZ)
DECREMENT(N_1^{out}) EXIT ON(N_1^{out}=0vN$_{1\,obs}^{out}$=0)
DECREMENT($N_{1\,obs}^{out}$) ENDREPEAT

c)if a triple belonging to the terminating state has N_1^{out}=0 or $N_{1\,obs}^{out}$=0,then perform the following operations after a):

REPEAT DECREMENT($D_{1\,obs}$)
EXIT ON($D_{1\,obs}$=1^D_1≠0vD$_1$=0) ENDREPEAT
INCREMENT(D_1)

d)if a triple belonging to the terminating state has N_1^{out}=0 or $N_{1\,obs}^{out}$=0 as the result of the step b),then apply the step c) after the step b).

If $D_{1\,obs}$=1^D_1≠0 occurs,then the transition to another observation point is performed with $D_{1\,obs}$=0,OPEN(AZ) and CLOSE(DZ) operations previously executed.If D_1=0,then we proceed backwards in RTVG space.

The tracking of the allocation and deallocation process is given only for the memoryless code generation.In the general case,the algorithm must include:

-detection of the exhausted source of available registers
-request for a memory pool intended for temporary storage.

Alternatively,a memory pool can be initially supplied:

$$XP=\{X+m,X+m-1,\ldots,X+2,X+1,X\} \qquad (56)$$

where X is the address of the first location in the memory pool.The definition of the allocation and deallocation zones is now modified to include the memory locations.The algorithm remains basically the same with few necessary changes:

-whenever FR is exhausted and there is a request for an additional register,then the following sequence of operations has to be executed:

UNSTACK(XP)→AZ DZ→STACK(FR)
OPEN(DZ) CLOSE(AZ)
UNSTACK(AR)→DZ CLOSE(DZ)
AZ→STACK(AR) OPEN(AZ)

thereafter,the procedure is continued in the usual way

-whenever a deallocation occurs due to binary operations,it must be checked whether the deallocated resource belongs to FR or XP.

6.TARGET REGISTER CONTROL

It is of interest sometimes to have the result of an evaluation in a predefined(target) register.This goal can be achieved very easily by using the allocation and deallocation zones.The target register is uniquely defined by the initial arrangement of FR stack.However, it is very difficult to find this initial arrangment in order to establish the direct target register control.An indirect control is more appropriate.Namely,we start with an arbitrary initial arrangment of FR stack to obtain the allocation and deallocation zones.Thereafter,we perform

$$AZ\_DZ=AZ[LAST]\cap DZ[LAST] \qquad (57)$$

to find the registers in the last allocation zone,deallocated in the last deallocation operation ,and

$$RESULT\_IS\_IN\_REGISTER=AZ[LAST]-AZ\_DZ \qquad (58)$$

to obtain the register which will contain the result of an evaluation.If our register coincides with RESULT_IS_IN_REGISTER,the allocation and deallocation zones remain unchanged. Otherwise,each appearance of RESULT_IS_IN_REGISTER in each allocation or deallocation zone has to be replaced by a target register and vice versa,to establish the target register control.

The memoryless code generation,tracking of allocation and deallocation process and target register control can be applied to the code generation for arithmetic expressions in compilers.We shall demonstrate it in the following example.

EXAMPLE.Let is given an arithmetic expression in postfix notation

E_p=A/*MN*-B/BCDEF/*/+-MN-+-JK*D+-/*E-F/*/*$

with its associated vector-generatrice

VG=110 222 116 538 2316 2121 1424 1129 1231 1034

Generate the code if the number of available registers is six.The target register is RO.

Firstly,we must construct RTVG according to properties of VG triples.We obtain

RTVG=0-10 -100 -301 -201 312 -100 100

By aplying THEOREMS I,II and III,we found that five registers were sufficient for the memoryless code generation.According to this conclusion,the allocation and deallocation zones are:

allocation zones: {RO,R1,R2,R3,R4},{R2},{R1},{R2,R4}
deallocation zones: {R3,R2},{R4,R1},{R2},{R2,R1,RO}

By applying Eqns (57) and (58),one obtains the target register R4.To change the target register to RO,RO and R4 must be changed to R4 and RO,respectively,in each relevant allocation and deallocation zone:

allocation zones: {R4,R1,R2,R3,RO},{R2},{R1},{R2,RO}
deallocation zones: {R3,R2},{RO,R1},{R2},{R2,R1,R4}

Bearing im mind the algorithm for right-to-left code generation(Velašević,1982) and the tracking zones,the following code is readily obtained:

```
LDA  R4,F     ADD  R3,E     LDA  R2,M
NEG  R4       LDA  RO,D     MUL  R2,N
LDA  R1,J     SUB  RO,R3    LDA  RO,A
MUL  R1,K     ADD  RO,R2    NEG  RO
ADD  R1,D     LDA  R2,C     SUB  RO,R2
LDA  R2,M     SUB  R2,RO    DIV  RO,B
SUB  R2,N     SUB  R2,R1    MUL  RO,R1
LDA  R3,F     LDA  R1,B     SUB  RO,E
NEG  R3       DIV  R1,R2    MUL  RO,R4
```

7.COST FUNCTION ANALYSIS

We would like the resulting assembly language program to be good under some cost function such as the number of assembly language instructions or number of memory fetches.For further considerations we introduce the cost function

CF=number_of_generated_instruction_lines (59)

We shall define now CF in terms of three functions:1)L-function(LF),2)S-function(SF),and 3) O-function(OF),i.e.,

CF=LF+SF+OF (60)

L-,S-,and O-functions give the number of generated LDA,STA and operator instructions,respectively.All these functions can be easily determined from VG and RTVG-based data structures.

The O-function is evaluated quickly from VG because the full information on the number of operators resides there.Thus,

$$OF=\sum_{i=0}^{N_{OS}-1} VG[i*3+2] \qquad (61)$$

The evaluation of L-function is based on the following theorem:

THEOREM IV.The number of generated LDA instructions does not depend on the number of available registers but it depends only on the structure of arithmetic expression.

Proof:Consider RTVG data structure.Whenever we make a transition from one RTVG triple to another,LDA instruction must be generated without regard to empty or full FR pool.Whenever N_1^n

and/or $N_1^{out} \neq 0$, LDA instruction must be also generated independently on the state of FR pool. This leads to the conclusion expressed by THEOREM IV.

Thus, the L-function is given by

$$LF = L - 1 + \sum_{i=1}^{l_{obs}-1} RTVG[i*3+2] + \sum_{\substack{i=1 \\ i \neq l_{obs}}}^{l_{obs}} RTVG[i*3+3] \qquad (62)$$

The value of the L-function can be decreased by generating for all subtract operators in non-commutative positions the instruction sequence SUB RI, memory and NEG RI. The cost function OF remains unchanged because the decrease of LF-value is compensated by the increase of OF value.

The evaluation of the S-function is performed as follows. Let the number of available registers is N_{FR}. Now, we start moving right in RTVG space. At every transition from one to another triple N_{FR} is decremented by one because every transition requires the allocation of a register. If N_{FR} becomes less than zero, a location named NUMSTA is incremented by one. Then, N_1^{in} is examined. For $N_1^{in} \neq 0$, N_{FR} is decremented again by one. IF $N_{FR} < 0$, NUMSTA is incremented by N_1^{in} and N_{FR} by one. The right moving discontinues when an observation point is encountered and handled. Then, N_{FR} is examined. If $N_{FR} \geq 0$, the left moving in RTVG space is started to obtain only the number of deallocated registers. This number is added to the current value of N_{FR}. The ordinal number of the triple where the left moving terminates, I^{RTVG}_{lm}, is determined by Eqn (63):

$$RTVG[(l_{obs}-1)*3+1] + \sum_{l=I^{RTVG}_{lm}}^{l_{obs}-1} sign(|RTVG[(l-1)*3+1]|) \leq \sum_{l=I^{RTVG}_{lm}}^{l_{obs}-1} |RTVG[(l-1)*3+1]| \qquad (63)$$

Eqn(63) is obtained through the following reasoning. In the left moving we must consume all operands at the observation point and singuler operands in RTVG triples which are accessed during this moving (some of these triples can have zero values for $RTVG[(l-1)*3+1]$ as a consequence of previous observation points). These operands will be consumed by the operators in RTVG triples accessed during the left moving with $D_l = RTVG[(l-1)*3+1]$. Thus, the stop point I^{RTVG}_{lm} corresponds to that triple whose contribution to the operand consumption satisfies Eqn (63). D_l's in all triples with $I^{RTVG}_{lm} < l \leq l_{obs}$ will be set to zero. $D_{I^{RTVG}_{lm}}$ is given by

$$RTVG[(l_{obs}-1)*3+1] + \sum_{l=I^{RTVG}_{lm}}^{l_{obs}-1} sign(RTVG[(l-1)*3+1]) - \sum_{l=I^{RTVG}_{lm}}^{l_{obs}-1} |RTVG[(l-1)*3+1]| - 1 \qquad (64)$$

N_{FR} is increased by the value of the second term in Eqn (64) because it represents the number of deallocated registers up to the point I^{RTVG}_{lm}.

If $N_{FR} < 0$, the following procedure applies for each triple:

a) $N_{FR} \leftarrow N_{FR}+1$, if $N_{FR} > 0$ then goto c)

b) for $N_1^{out} \neq 0$ and $N_{l_{obs}}^{out} \neq 0$, NUMSTA $\leftarrow$ NUMSTA $+ N_1^{out}$, $N_{l_{obs}}^{out} \leftarrow N_{l_{obs}}^{out} - N_1^{out}$

c) if stop point then set $D_{I^{RTVG}_{lm}}$; set $N^{out}_{I^{RTVG}_{lm}}$; restart moving right else $D_l \leftarrow 0$; goto^{lm}c).

If N_{FR} becomes zero during the left moving, only the steps a) and c further apply. The stop point is determined also by Eqns(63) and (64). Thereafter, the right moving is resumed by the same procedure but with a new value for N_{FR}. At the stop point a correction must be done because we did not control the step b) on the zero value for $N^{out}_{l_{obs}}$. The correction is made simply by setting $N^{out}_{I^{RTVG}_{lm}} = -N^{out}_{l_{obs}}$. Evidently, the location NUMSTA I^{RTVG}_{lm} will contain the value of the S-function. This value can be very easily found from the allocation zones:

$$SF = \sum_{i=1}^{LAST} XPMAZ[i] \qquad (65)$$

This is quite comprehensive because every allocation from the set XP means the generation of the STA instruction. The cost function can be applied in the optimization of arithmetic expressions.

8. EVALUATION OF EXPRESSIONS IN MULTIPROCESSOR MACHINES

VG and RTVG-based data structures can be used for the determination of parallelism in evaluation of expressions in multiprocessor machines.

The most simple way to distribute the load among the processors is to assign to each processor the evaluation from one to next observation point. Let the number of processors is NP and the number of observation points NOBS. If the condition NP>NOBS is valid, then each processor in the set NOBS is loaded with one observation point. Otherwise, NP-(NOBS)modNP processors will be loaded with $\lfloor NOBS/NP \rfloor$ observation points, and NOBSmodNP processors with $\lceil NOBS/NP \rceil$ observation points. The loading with an observation point comprises all instructions generated between two successive observation points as well as the instructions generated in the current and terminating state of the observation point under consideration. Because the registers of separate processors are mutually inaccessible, the contents of all registers belonging to the set AR must be stored in predefined locations upon the handling of an observation point. This increases the cost function of an expression.

The load distribution calculated according to the cost function defined by Eqn (60) does not offer the right insight into the load of each processor because of different durations of particular operations. To take into account this fact, it is necessary to establish relative weighting measures of different operations. The exact measures are not easily obtainable because the same operation in a given expression can be of register-to-register and /or register-to-memory type.

It is readily verified that the number of register-to-register operations, NO_{RR} is given by

$$NO_{RR} = \sum_{i=1}^{LAST} \not\perp DZ[i] + \sum_{i=1}^{NOB} N_i^B(unary) \qquad (66)$$

where $\not\perp DZ[i]$ means the number of elements in the zone i. The number of register-to-memory operations, NO_{RM}, is easily found from

$$NO_{RR} = OF - NO_{RR} \qquad (67)$$

Furthermore, the duration of + and - operations differs substantially from the duration of * and / operations. Bearing this in mind, we can establish the following relative measures:

-,+(RR)	-1	LDA(RM)	-2	*(RM)	-m+2
-,+(RM)	-2	STA(RM)	-2	/(RR)	-d+1
//(R)	-1	*(RR)	-m+1	/(RM)	-d+2

m and d are relative durations of multiply and divide operations, respectively.

The total cost function expressed through these relative weighting measures is

$$CF_W = 2LF + 2BF + NO_{RR} + 2NO_{RM} + m \sum_{i=1}^{NOB} N_i^B(\text{multiply}) + d \sum_{i=1}^{NOB} N_i^B(\text{divide}) \quad (68)$$

where N_i^B(multiply) and N_i^B(divide) represent the number of multiply and divide operators in a triple VG, respectively.

The gain achieved in execution speed by such a procedure of load distribution is approximately

$$G = CF_W / \max_{l_{obs} \in 1,L} \{CF_W^{l_{obs}}\} \quad , NP \geq NOBS \quad (69)$$

The load distribution through the observation points is uncontrolled in the sense that the control of load is not introduced. The control can be established by the following procedure:

a) calculate the total weighting cost function by applying Eqn (68)
b) find the load of one processor: LOAD = CF_W/NP
c) set i=1
d) start the code generation from the current point in VG
e) evaluate the momentary weighting cost function ($CF_{W,i}$) during the code generation
f) when $CF_{W,i}$ becomes the closest to LOAD, suspend temporary the code generation and generate the STA instructions whose number should be equal to the population of the set {AR}; if i=NP then omit this operation and terminate the code generation
g) set $CF_{W,i} = CF_{W,i} + 2\mathcal{L}\{AR\}$
h) i = i+1
i) goto d)

The gain function for controlled load distribution, G_c, is given by

$$G_c = CF_W / \max_{i=1,NP} \{CF_{W,i}\} \quad (70)$$

Although the controlled load distribution is more fair, high correlation between different segments of expressions may significantly decrease its efficiency due to delays caused by necessary synchronization of different segments of expression. This correlation is much lower in the uncontrolled load distribution thus providing greater gain in efficiency.

9. CONCLUSION

It is shown that the concept of vector-generatrice is very suitable for various aspects of analysis of arithmetic expressions. The analysis comprised: a) determination of the number of registers required to compute an arithmetic expression without STORE commands, b) tracking of allocation and deallocation process (a new approach to analysis of arithmetic expressions), which provides the complete insight into the sequence of allocation and deallocation of registers by introducing the concept of allocation and deallocation zones, c) target register control which ensures that the result of an evaluation will be found in a given register, d) cost function analysis which gives the number of generated instruction lines, and e) expression evaluation in multiprocessor machines.

The algorithm for a) can be used for quick determination of the condition for the memoryless code generation.

The allocation and deallocation zones obtained through the tracking of allocation and deallocation process are very suitable for the code generation; however, this has not been here particularly explored.

The expression evaluation in multiprocessor machines deals with the problem of load distribution in the evaluation of an expression in a controlled or incontrolled manner. Although the controlled load distribution is more fair, high correlation between different segments of expression may significantly decrease its efficiency due to delays caused by synchronization of evaluation of different segments of the expression. This correlation is much lower in the uncontrolled load distribution thus providing greater gain in efficiency. This efficiency may be further improved by assigning two or more observation points to one processor to achieve a better load balance among the processors.

10. REFERENCES

1. I. Nakata, On compiling algorithms for arithmetic expressions, CACM 12, February 1967.
2. R. Rudziejowski, On arithmetic expressions and trees, CACM 12, February 1969.
3. R. Sethi and J. Ullman, the generation of optimal code for arithmetic expressions, JACM 17, April 1970.
4. L. Belady, A study of replacement algorithms for a virtual storage computer, IBM Systems Journal 5, May 1966.
5. L. Horwitz et al., Index register allocation JACM 13, January 1966.
6. K. Kennedy, Index register allocations in straight-line code and simple loops, in: Design and Optimization of Compilers (Ed. by R. Rustin), Prentice-Hall, Englewood Cliffs, New Jersey, 1972.
7. R. Sethi, Validating register allocations for straight-line programs, Phd Thesis, Department of Electrical Engineering, Princeton University, 1973.
8. R. Allard et al., Some effects of the 6600 computer on languages structures, CACM 7, February 1964.
9. H. Hollerman, Parallel processing of algebraic expressions, IEEE Transactions on Electronic Computers EC-15, January 1966.
10. J. Squire, A translation algorithm for a multiple processor computer, NACM, 1963.
11. H. Stone, One-pass compilation of arithmetic expressions for a parallel processor, CACM 10, April, 1967.
12. J. Baer and D. Bovet, Compilation of arithmetic expressions for parallel computation, Proceedings of the IFIP Congress 84-B10, 1968.
13. R. Graham, Bounds on multiprocessing anomalies and related packing algorithm, Proceedings of the AFIPS Spring Joint Computer Conference, Vol. 40, AFIPS Press, Montvale, New Jersey, 1972.
14. D. Veljašević, Right-to-left code generation for arithmetic expressions, Computer Journal, Vol. 25, No. 3, 1982.

INFORMATICA 4/1983

JEZGRA VIŠEPROCESORSKOG OPERACIONOG SISTEMA REALIZIRANA NA SISTEMU S MIKROPROCESORIMA IM6100

M. JELAVIĆ

UDK: 681.3.06.IM6100

INSTITUT „RUĐER BOŠKOVIĆ", ZAGREB

U ovom radu razmatra se programska podrška za višeprocesorski sistem sa višesabirničkim posrednim pristupom realiziran s mikroprocesorima IM6100. Programska podrška je realizirana u obliku jezgre operacionog sistema, koja omogućava efikasnu implementaciju paralelizma na nivou zadatka nekog korisničkog programa. Dani su algoritmi za implementaciju najosnovnijih funkcija jezgre koji podržavaju upravljanje, kontrolu i komunikaciju sa ulazno-izlaznim jedinicama i u kojima su implementirani osnovni mehanizmi za komunikaciju i koordinaciju zadataka u toku izvođenja.

In this paper a software support for multiprocessor multibus system with indirect access, realized with IM6100 microprocessors, is given. Software support is realized with a kernel of the operating system which enable efficient implementation of parallelism at task level. Algorithms for primitives of the kernel are given, through which governing, control and communication of peripheral devices is established and in which essential mechanism for communication and task coordination is implemented.

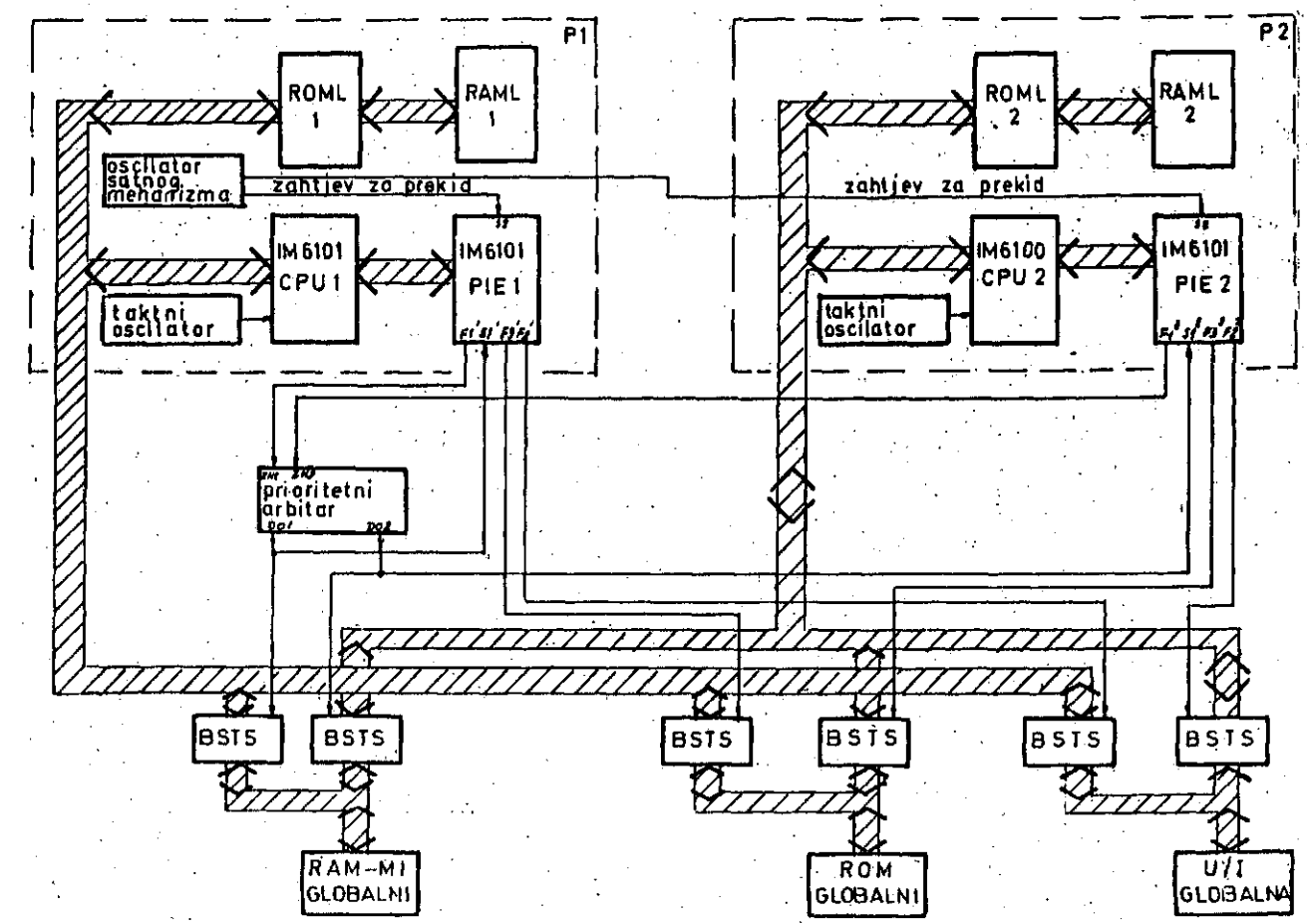
1. UVOD

Pod jezgrom operacionog sistema podrazumijevamo skup programskih odsječaka čija je funkcija upravljanje, kontrola, sinhronizacija i međusobna komunikacija između logičkih cijelina nekog korisničkog programa koje nazivamo zadacima /1/. Jezgra može biti realizirana kao jednostavni monolitni monitor koji ima mogućnosti predodjelivanja procesne jedinice zadacima, primanja i posluživanja prekida koji dolaze od ulazno-izlaznih jedinica, zatim startanje raznih jedinica (ADC ili DAC npr.) i mogućnost manipuliranja sa semaforima. Izvan jezgre operacionog sistema svi prekidi su "nevidljivi" tj. ne mogu utjecati na logički nastavak zadatka koji se trenutno izvršava na procesnoj jedinici. U osnovi jezgra je sastavni dio operacionog sistema u kojoj su implementirane najprimitivnije funkcije i njena struktura u biti ovisi o samoj konfiguraciji sistema. U odnosu na ostale dijelove operacionog sistema ona je dio kojije najprisnije vezan sa "hardware-om".

Operacioni sistem pa prema tome i jezgra operacionog sistema koja podržava sistemske organizacije sa više procesnih jedinica složenija je i zahtijeva implementaciju programskih struktura specifičnih za takve organizacije. Osim osnovnih funkcija mora zadovoljavati i dodatne zahtijeve koji proizlaze iz mogućnosti da se zadaci mogu vršiti istovremeno na više procesnih jedinica tj. mora podržavati međuprocesorsku komunikaciju i koordinaciju zadataka u svrhu maksimalnog korištenja postojećih sredstava u sistemu.

Sposobnost podržavanja dva ili više zadataka u obradi ("multitasking") je jedno od osnovnih svojstava jezgre operacionog sistema. U sistemu sa jednom procesnom jedinicom to omogućava kvaziparalelan rad više zadataka dok u višeprocesorskom sistemu omogućava raspodjelu zadataka na više procesnih jedinica i kao posljedicu toga istovremeno izvršavanje zadataka.

U ovom radu prikazan je primjer implementacije jezgre operacionog sistema prvenstveno prilagođen konfiguraciji sa posrednim pristupom realiziranoj sa mikroprocesorima IM6100 (Slika 1.).



Sl. 1. Organizacija sa posrednim pristupom

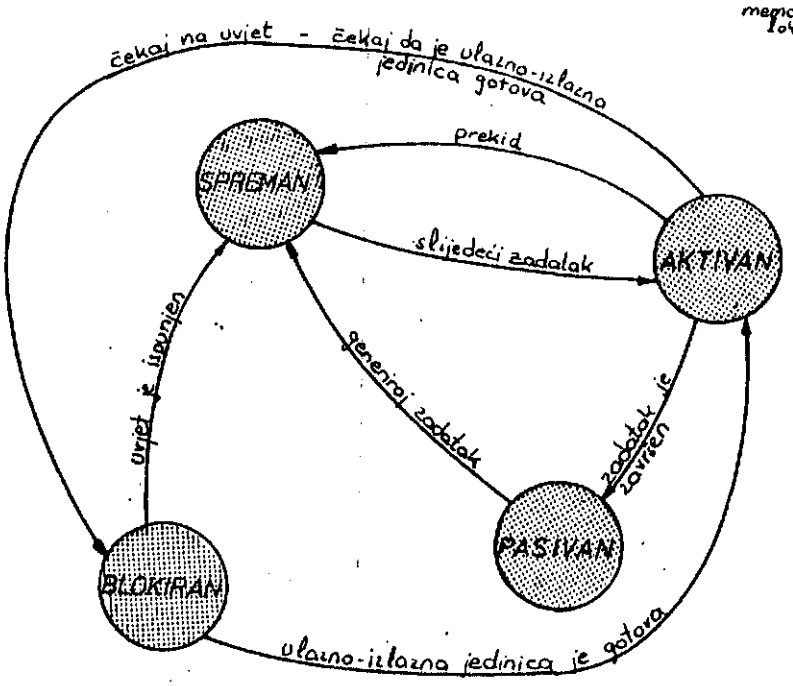
2. KARAKTERISTIKE JEZGRE

Jezgra operacionog sistema implementirana na višeprocesorskom sistemu sa posrednim pristupom sa mikroprocesorima IM6100 realizirana je u obliku *monolitnog monitora*. Sva komunikacija izmedu korisničkih zadataka i ulazno-izlaznih jedinica mora se odvijati preko jezgre. Ulaskom u jezgru zabranjuju se svi prekidi do izlaska iz nje. Svaka procesna jedinica ima po kopiju jezgre u svojoj lokalnoj memoriji. Jezgra operacionog sistema može podržavati lokalne zadatke koji se nalaze u lokalnoj memoriji procesne jedinice i globalne zadatke koji se nalaze u globalnoj memoriji i koji mogu biti izvršavani na bilo kojoj procesnoj jedinici. Lokalni zadaci imaju prioritet izvršavanja nad globalnim zadacima. Zajedničke liste i polja (kao što su globalne liste čekanja i deskriptori globalnih zadataka) nalaze se u globalnoj memoriji RAM-a. Primitivne funkcije implementirane u jezgri operacionog sistema jesu:

- međusobno isključivanje (sklopovski dio)
- spremanje zadatka u listu čekanja
- vađenje zadataka iz liste čekanja
- pamćenje statusa zadatka
- restauriranje statusa zadatka
- prekid satnim mehanizmom
- čekanje uvjeta
- signaliziranje uvjeta
- startanje ulazno-izlazne jedinice
- signaliziranje do je ulazno-izlazna jedinica gotova
- P i V mehanizmi

Zadatak u sistemu može biti u jednom od 4 stanja prikazanih na slici 2.

Zadatak je u stanju PASIVAN kada nema potrebe za njegovim izvršenjem ili kada još nije iniciran. Inicirati zadatak može satni mehanizam ili korisnik koji generira zadatak. Zadatak se stavlja u listu čekanja za procesnu jedinicu i na taj način prelazi u stanje SPREMAN. Kada je procesna jedinica slobodna ona se dodjeljuje prvom od zadataka u listi čekanja za procesnu jedinicu, koji nakon dodjele prelazi u stanje AKTIVAN. U stanju SPREMAN je onoliko zadataka koliko ih je u listi čekanja za procesnu jedinicu dok u stanju AKTIVAN može biti onoliko zadataka koliko ima procesnih jedinica u sistemu.

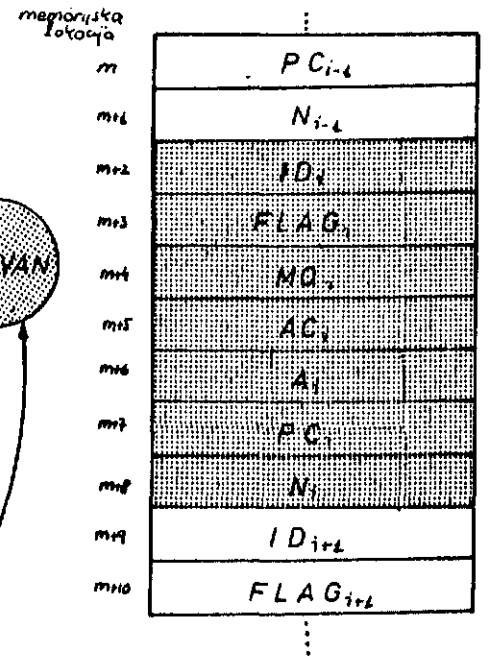


Sl. 2. Osnovna stanja zadatka

Svaki put kada nastupi prekid trenutno aktivan zadatak postavlja se u stanje SPREMAN (tj. sprema se u listu čekanja za procesnu jedinicu) i poduzimaju se predviđene radnje u vezi prekida. Ako je prekid došao od satnog mehanizma slijedeći zadatak iz liste čekanja iz stanja SPREMAN prelazi u stanje AKTIVAN. U slučaju da je prekid došao od ulazno-izlazne jedinice zadatak koji čeka završetak ulazno-izlazne jedinice i u stanju je BLOKIRAN prelazi u stanje AKTIVAN. Ako zadatak koji je AKTIVAN ima za daljnje napredovanje postavljen nekakav uvijet on prelazi u stanje BLOKIRAN a napredovanje se nastavlja sa prvim zadatkom u listi čekanja za procesnu jedinicu. Ispunjenjem uvijeta svi zadaci koji su čekali u stanju BLOKIRAN za taj uvijet stavljaju se u stanje SPREMAN. Po završetku zadatak prelazi u stanje PASIVAN.
Za ovakvu koncepciju jezgre i implementaciju primitivnih funkcija moraju se definirati dve programske strukture; deskriptor zadataka i lista čekanja.

3. DESKRIPTOR ZADATKA

Svakom korisničkom zadatku ZD u sistemu pridružen je jedinstveni indeks "i" iz skupa pozitivnih cijelih brojeva i jedinstveno polje koje nazivamo deskriptorom zadatka. Deskriptor zadatka sadrži sve informacije potrebne da stanje zadatka bude definirano u bilo kojem trenutku. Njegova dužina ovisi o mikroprocesoru sa kojim je sistem realiziran i kompleksnosti jezgre operacionog sistema.



Sl. 3. Deskriptor zadatka

Za sistem sa mikroprocesorima IM6100 deskriptor zadataka ZD_i je uređena sedmorka (Sl.3)

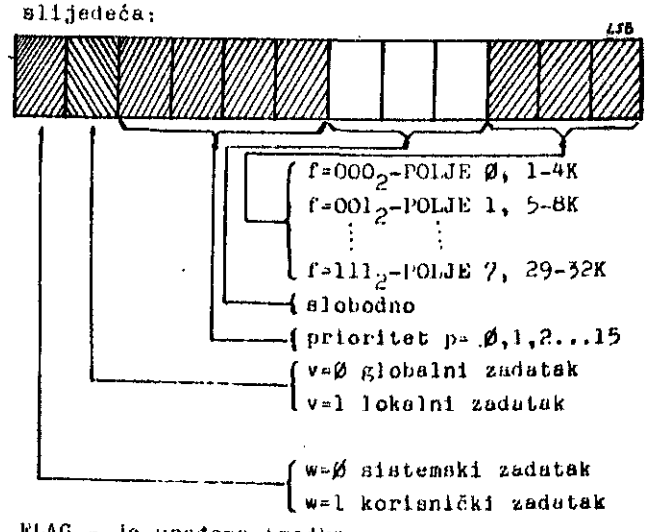
$$D_i = (ID_i, FLAG_i, MQ_i, AC_i, A_i, PC_i, N_i)$$

gdje je:

ID_i - identifikator zadatka "i". Identifikacija se sastoji od informacije da li je zadatak sistemski ili korisnički (w_i) da li je globalni ili lokalni (v_i), od prioriteta zadatka (p_i) i od polja memorije (f_i) u kome se nalazi zadatak. Prema tome ID_i je uređena četvorka:

$$ID_i = (w_i, v_i, p_i, f_i)$$

organizacija riječi identifikatora je slijedeća:

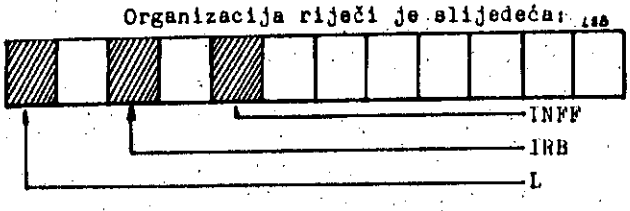


$FLAG_i$ - je uređena trojka:

$$FLAG_i = (L_i, IRB_i, INPF_i)$$

koja daje informaciju o stanju IM6100. L_i označava "link" od akumulatora, IRB

indicira zahtijev za prekid a INFF indicira stanje prekidnog bistabila. Organizacija riječi je slijedeća:



MQ_i - je memorijska lokacija u koju se zapisuje sadržaj internog MQ registra centralno procesne jedinice IM6100.

AC_i - je memorijska lokacija u koju se zapisuje sadržaj akumulatora od IM6100.

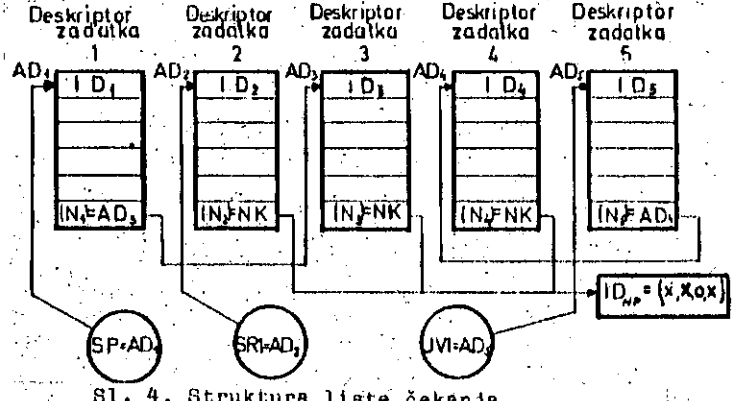
A_i - je memorijska lokacija u koju se zapisuje interni A registar od veznog elementa IM6101.

PC_i - je memorijska lokacija u koju se zapisuje adresa slijedeće instrukcije koja se treba izvršiti.

N_i - je memorijska lokacija u koju se zapisuje adresa deskriptora zadatka koji je slijedeći u listi čekanja.

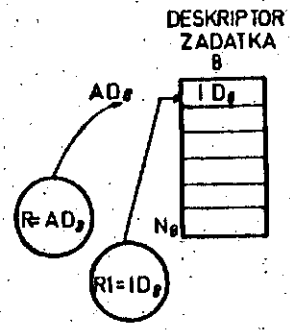
4. LISTA ČEKANJA

U sistemu koji podržava više aktivnih zadataka dešava se da više zadataka zatraži pristup istom sredstvu. Logička posljedica nemogućnosti sredstva da zadovolji sve potrebe odjedanput, je implementacija programske strukture koju nazivamo lista čekanja ili rep. To je linearna lista u kojoj elementi, u ovom slučaju zadaci, čekaju na dodjelu sredstva. Za svako sredstvo bilo sklopovsko ili programsko postoji lista čekanja. Manipuliranje sa listama čekanja svodi se na manipulaciju sa memorijskim lokacijama N_i u deskriptoru zadatka i rezerviranim memorijskim lokacijama u kojima se nalaze prvi elementi u listi čekanja Na slici 4 prikazan je slučaj organizacije liste čekanja kada dva zadatka čekaju na dodjelu procesne jedinice (ZD_1,ZD_3), dva zadatka (ZD_5,ZD_4) čekaju ispunjenje uvjeta l i jedan zadatak (ZD_2) čeka dodjelu sredstva l.



Sl. 4. Struktura liste čekanja

Rezervirana memorijska lokacija SP označava adresu deskriptora prvog zadatka u listi čekanja za procesnu jedinicu. Lokacija N_i deskriptora zadatka koji je zadnji u listi čekanja sadrži nul-kazalo. Nul-kazalo označava adresu memorijske lokacije u kojoj se nalazi "lažni" prioritet zadatka. Vrijednost tog prioriteta je najniža moguća vrijednost koju zadatak može poprimiti iz skupa brojeva Ø do 15. Vrijednost prioriteta Ø je rezervirana samo za lažni prioritet i niti jedan zadatak u sistemu ne može poprimiti tu vrijednost prioriteta. Lažni prioritet označuje kraj liste čekanja. Npr. na slici 4 deskriptori zadatka ZD_2,ZD_3 iZD_4 u lokacijama N_2,N_3 i N_4 sadrže adresu nul-kazala definirajući sebe kao zadatke koji su zadnji u listi čekanja. Svaka procesna jedinica u sistemu ima rezervirane 2 lokalne RAM lokacije sa slijedećim značenjem (Sl. 5.):



Sl. 5.

R - sadrži adresu deskriptora zadatka koji se izvršava na procesnoj jedinici.

R_1 - sadrži identifikator zadatka koji se izvršava na procesnoj jedinici.

Identifikator mora biti prisutan radi informacije da li je zadatak globalni ili lokalni.

5. PRIMITIVNE FUNKCIJE JEZGRE

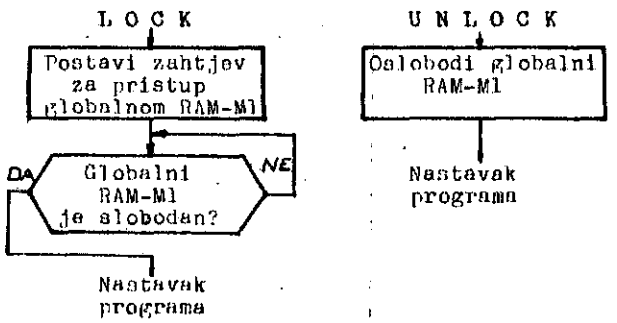
5.1. Medjusobno isključivanje

Kada u sistemu postoji više zadataka koji za svoj daljnji rad trebaju pristup nekom sredstvu moraju se implementirati mehanizmi za jednoznačno i isključivo dodjeljivanje sredstava jednom od zadataka. To je naročito izraženo u sistemu sa više procesnih jedinica kada može biti više od jednog zadatka aktivno i kada mogu istovremeno zatražiti pristup istom globalnom sredstvu. Posebni sklopovi nazvani arbitri rješavaju takve konfliktne situacije /2/. Za sistem za procesnom jedinicom IM6100 mora postojati programska podrška za upotrebu arbitara. Arbitraža preko arbitra vrši se samo za globalni RAM - Ml u kojem se nalaze sve globalne varijable i polja. Za ostala globalna sredstva arbitraža se vrši preko globalnih varijabli nazvanih semaforima i koji su opisani kasnije. Arbitraža za globalni RAM - Ml sastoji se iz

programskih odsječaka LOCK i UNLOCK (Sl.6a i Sl.6b).

Programski odsječak LOCK postavlja zahtijev za dodjelu globalnog RAM - Ml i čeka u petlji dok mu se ne dodijeli. To je jedini slučaj kada zadatak čeka neki uvjet u petlji čekanja. UNLOCK oslobadja globalni RAM - Ml.

Razlog za eksplicitnu arbitražu za globalni RAM leži u samoj gradnji mikroprocesora TM6100 sa kojom je realiziran višeprocesorski sistem /2/. TM6100 u svom repertuaru instrukcija nema niti jednu instrukciju koja bi u jednom ciklusu čitala, pisala i testirala memorijsku lokaciju i zabranila pristup i korištenje memorije između i za vrijeme tog ciklusa. Iz tog razloga sklopovski pristup globalnom RAM-u Ml realiziran je eksplicitnom metodom traženja pristupa tj. programskim odsječkom.



Sl. 6a. LOCK odsječak Sl. 6b. UNLOCK odsječak

```
LOCK,  SPLAG1   /ZII_x=1   UNLOCK,CPLAG1 /ZII_x=0
       SKIP1    /DO_x=1?
       JMP .-1  /ne!
       - -      /da!
       -
       -
```

5.2. Spremanje zadatka u listu čekanja

Spremanje zadatka u listu čekanja vrši se po principu prioriteta. Svakom zadatku pridružen je unaprijed određeni prioritet od 1 do 15 i u ovisnosti o prioritetu sprema se na određeno mjesto u listi. Najveći prioritet imaju zadaci sa p = 15, a najniži sa p = 1. Zadaci sa istim prioritetom spremaju se po principu FIFO tj. koji je prvi stigao. Rezervirana lokacija P sadrži adresu deskriptora prvog zadatka u listi čekanja gdje je:

$$P \ni \{SP, UV1, UV2, \ldots\ldots UV_x\}$$

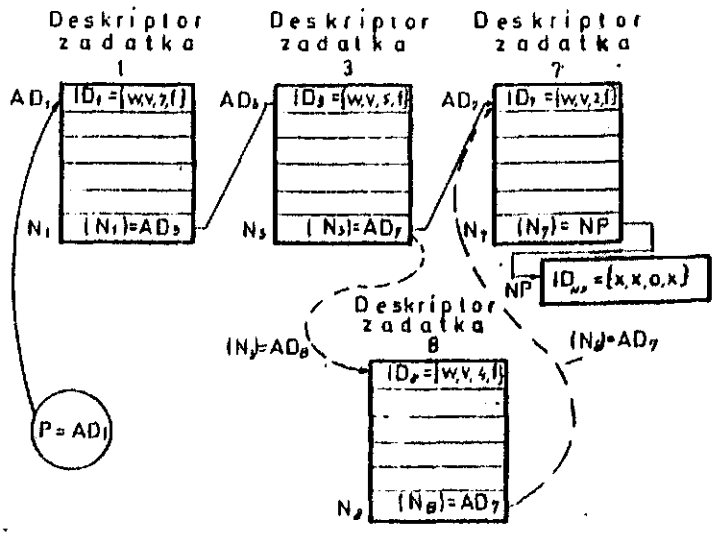
SP je memorijska lokacija u kojoj se nalazi adresa deskriptora zadatka koji je prvi u listi čekanja za procesnu jedinicu.

UV_x je memorijska lokacija u kojoj se nalazi adresa deskriptora zadatka koji je prvi u listi čekanja za uvjet$_x$. Čekanje uvjeta može biti i čekanje da neko sredstvo kao npr. memorija bude slobodna.

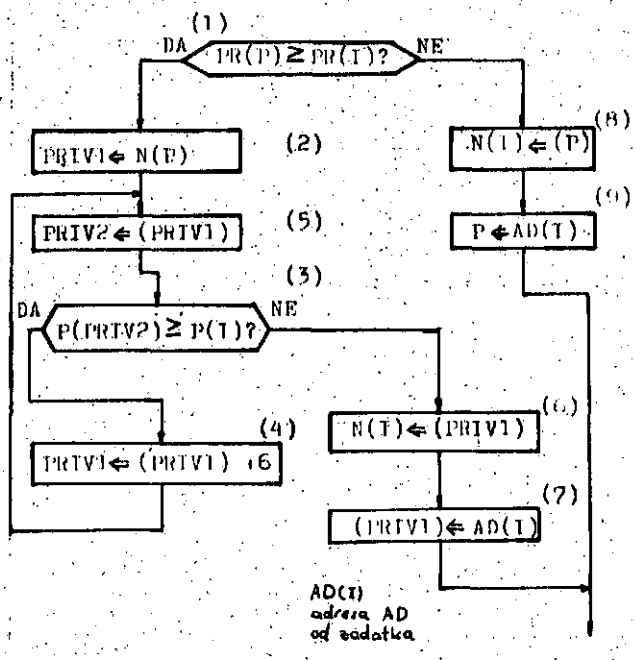
Na slici 7. su prikazane radnje koje treba izvršiti za stavljanje zadatka u listu čekanja, a na slici 8. pripadni algoritam koji to obavlja.

Prvo ispitujemo da li je prioritet zadatka koji je prvi u listi veći od prioriteta zadatka ZD_i kojeg hoćemo staviti u listu (1). Ako nije, to znači da zadatak ZD_i treba doći na prvo mjesto jer je najvećeg prioriteta. U lokaciji N_i zadatka ZD_i stavljamo adresu deskriptora zadatka koji je do sada bio prvi (8), a u lokaciju P adresu deskriptora zadatka ZD_i (9).

Ako prvi zadatak u listi čekanja ima veći prioritet od zadatka ZD_i onda se preko (3), (4) i (5) traži prvi slijedeći zadatak u listi koji ima manji prioritet od zadatka ZD_i. Traženje manjeg prioriteta ima za posljedicu stavljanje zadatka ZD_i u listu čekanja iza svih zadatka koji imaju isti prioritet. Ako u listi čekanja nema niti jednog zadatka, lokacija P sadrži adresu lokacije koja ima "lažni" prioritet. Svaki eventualni dolazak zadatka sa najnižim mogućim prioritetom (p=1) ima za posljedicu stavljanje tog zadatka u listu prije "lažnog" zadatka. Deskriptor "lažnog zadatka sastoji se samo iz identifikatora ID koji sadrži u sebi samo jednu relevantnu informaciju a to je "lažni" prioritet Ø.



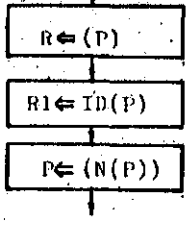
Sl. 7. Spremanje zadatka u listu čekanja



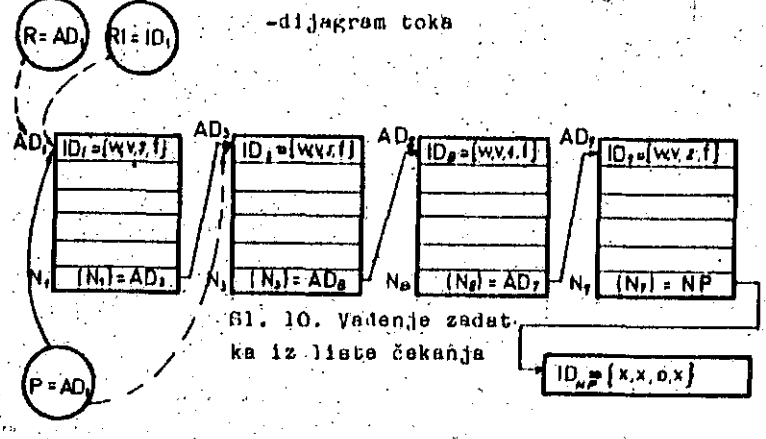
Sl.8. Spremanje zadatka u listu čekanja -dijagram toka

5.3. Vađenje zadatka iz liste čekanja

Vađenje zadatka iz liste čekanja je jednostavna operacija (sl. 9) koja se svodi na promjenu sadržaja lokacije R u kojoj se nalazi adresa deskriptora zadatka koji je ili završen ili prekinut iz nekog razloga. U tu lokaciju stavljamo prvi slijedeći zadaćak u listi, tj. adresu njegovog deskriptora (sl. 10). Prije promjene lokacije P spremamo adresu deskriptora prvog zadatka u listi u lokaciju R označavajući time da će taj zadatak biti dodijeljen procesnoj jedinici, i identifikator tog zadatka ID u lokaciju R1.



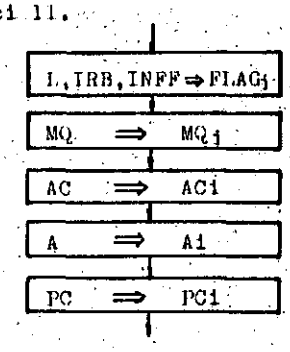
Sl. 9. Vađenje zadatka iz liste čekanja -dijagram toka



Sl. 10. Vađenje zadatka iz liste čekanja

5.4. Pamćenje statusa zadatka

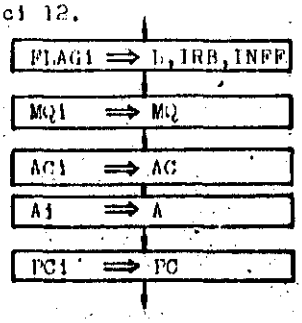
U sistemima u kojima može biti više zadataka u stanju SPREMAN procesne jedinice mogu biti pridodijeljene različitim zadacima u toku izvođenja nekog korisničkog programa. Mehanizam koji omogućava to dodijeljivanje i predodijeljivanje mora osigurati integritet zadatka. To se postiže pamćenjem svih relevantnih informacija u memorijskom polju koje smo nazvali deskriptorom zadatka. Status procesora u nekom trenutku izvođenja nekog zadatka ujedno je i status samog zadatka koji se treba zapamtiti. Status kod sistema sa mikroprocesorom IM6100 treba zapamtiti eksplicitno nakon svakog prekida zadatka. Radnje koje treba obaviti prikazane su dijagramom toka na slici 11.



Sl. 11. Pamćenje statusa zadatka

5.5. Restauriranje statusa zadatka

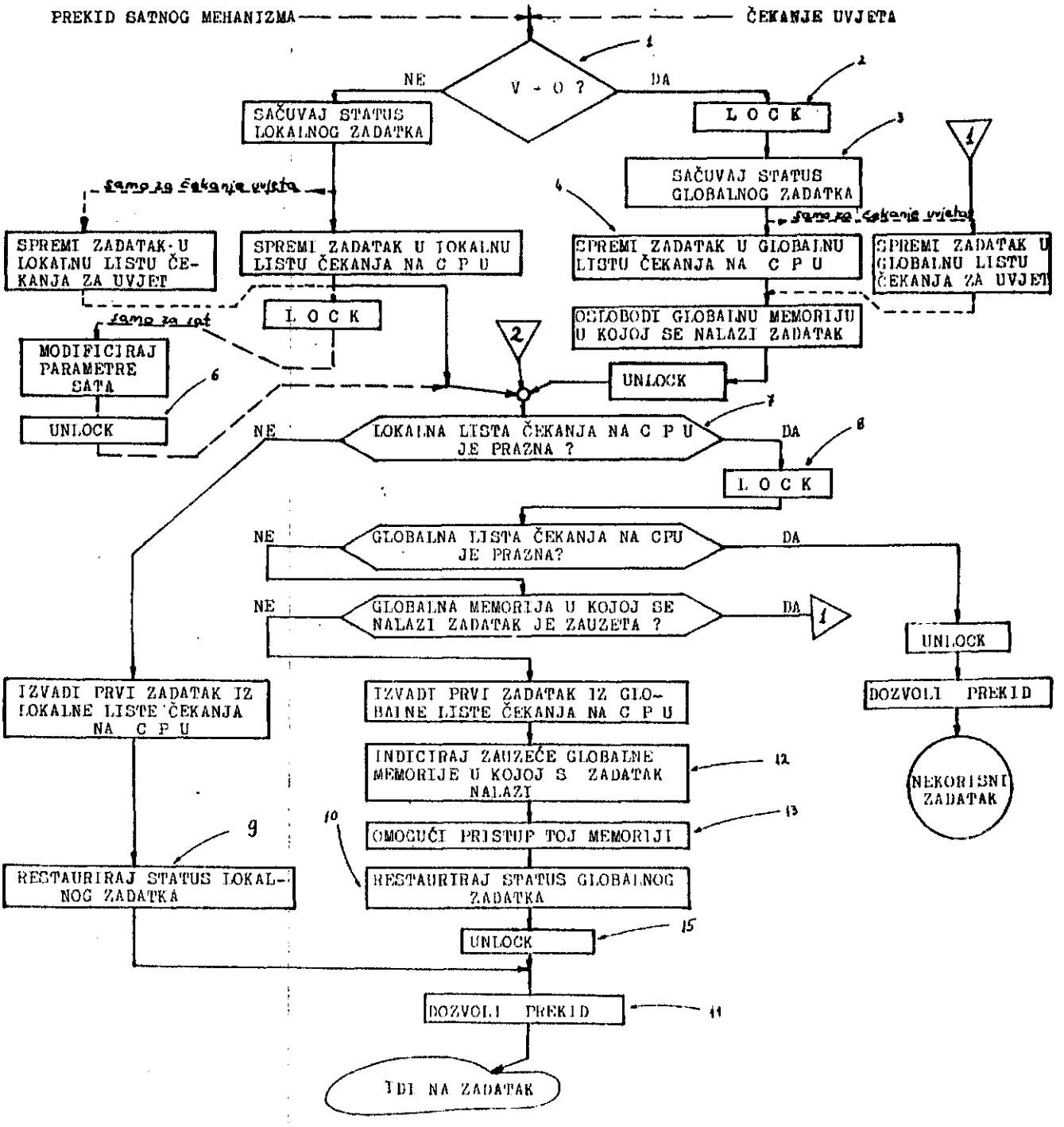
Analogno pamćenju statusa zadatka prilikom njegovog "odstranjivanja" sa procesne jedinice potrebno je kod svakog dodjeljivanja zadatka procesnoj jedinici restaurirati njegov status. Restauriranje statusa zadatka znači praktično dovođenje procesora u onakvo stanje u kojem je bio u trenutku prije nego što je taj zadatak bio "odstranjen". Potrebne radnje koje treba obaviti prikazane su dijagramom toka na slici 12.



Sl. 12. Restauriranje statusa zadatka

5.6. Prekid satnim mehanizmom

Svaki put kada satni mehanizam u točno određenom vremenskom intervalu da



Sl. 14. Prekid satnim mehanizmom i čekanje uvjeta

impuls za prekid, procesna jedinica se nakon potprograma jezgre "prekid satom" dodjeljuje drugom zadatku. U trenutku nakon impulsa sata onemogućuje se eventualni novi prekid (dijagram toka na slici 15) i ispituje se da li je prekinuti zadatak bio lokalni ili globalni(1). Lokalnim zadacima pridruženi su deskriptori koji se nalaze u lokalnim memorijama tako da samo ona procesna jedinica kojoj je pridodijeljen zadatak ima pristup u njegov deskriptor. Isto tako je sa globalnim zadacima čiji se deskriptori nalaze u globalnoj memoriji s tom razlikom da su oni pristupni svim procesnim jedinicama. Lokalni deskriptori zadatka povezani su u lokalne liste čekanja, a globalni u globalnu listu čekanja. Predmet izvršavanja na procesnoj jedinici uvijek imaju lokalni zadaci, tj. zadaci koji se nalaze u lokalnoj listi čekanja za procesnu jedinicu.

Algoritam na slici 14. ispituje parametar "v" identifikatora u deskriptoru zadatka i zaključuje da li je zadatak globalni ili

lokalni (1). Ako je globalni onda se traži isključivo pristup globalnoj memoriji RAM - Ml (2), čuva se status zadatka (3) i sprema ga se u globalnu listu čekanja SPREMNO (4). Nakon toga se oslobađa globalna memorijska jedinica u kojoj se nalazi globalni zadatak. U slučaju lokalnog zadatka nije potrebna arbitraža za globalnu memoriju. Na taj način se jezgru operacionog sistema oslobodilo nepotrebnog ulaska u kritični odsječak i arbitraže za globalnu memoriju.

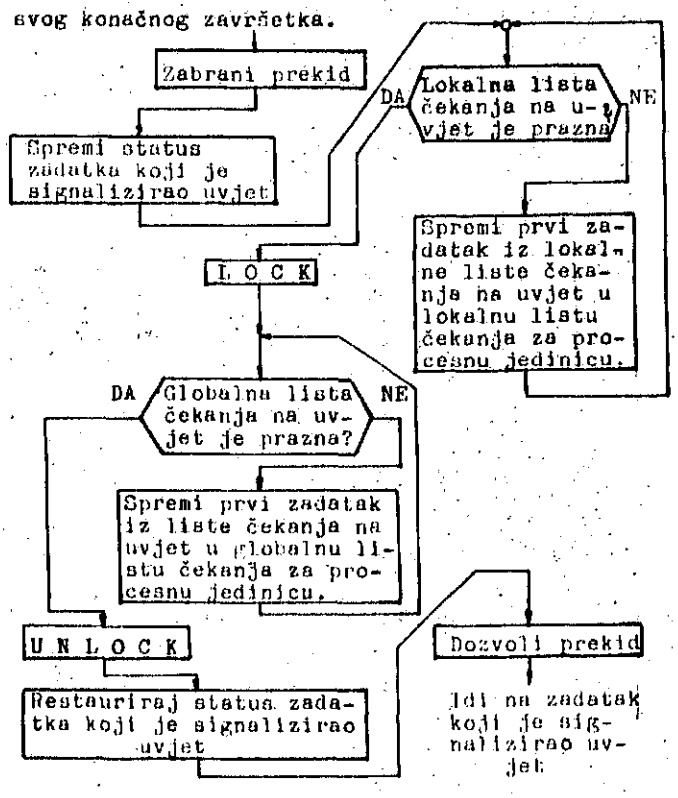
Programski odsječak "modificiraj sat" je sastavni dio jezgre, a specifičan je samo za ovaj potprogram. Prekid satnim mehanizmom je riješen sklopovski dok je modificiranje sekunda, minuta, sata i dana riješeno programski. Memorijske lokacije za parametre satnog mehanizma nalaze se u globalnoj memoriji tako da su pristupne svim procesnim jedinicama. Modifikaciju sata vrši samo jedna procesna jedinica (Pl na slici 1.). Nakon toga oslobađa se pristup globalnoj memoriji (6) i ispituje se da li je lokalna lista čekanja za procesnu jedinicu prazna (7). U slučaju da u listi čekanja više nema zadataka ponovo se traži pristup globalnoj memoriji RAM - Ml (8) u kojoj je lista čekanja za procesnu jedinicu globalnih zadataka. Ako i u toj listi nema zadataka onda se ide na izvršavanje nekorisnog zadatka. Prije početka izvršavanja zadatka mora se restaurirati status zadatka (9),(10) omogućiti novi prekid (11), a u slučaju globalnog zadatka inicirati preko semafora (12) zauzeće memorijskog modula i omogućiti pristup (13) u kome se nalazi taj zadatak te osloboditi globalnu RAM memoriju (15).

5.7. Čekanje uvjeta

Zadatak u svome izvođenju može doći do točke u kojoj je za nastavak rada potrebno ispunjenje nekih uvjeta. To može biti zahtjev za pristup nekom globalnom memorijskom modulu, pristup potprogramu za izračunavanje specijalnih funkcija ili rezultat računanja nekog drugog zadatka. Za svaki od tih uvjeta mora postojati posebna lista čekanja za globalne i lokalne zadatke. Zadatak koji se izvršava na procesnoj jedinici i za koji mora biti ispunjen uvjet za nastavak rada sprema se u pripadnu listu čekanja za taj uvjet, a procesna jedinica se dodjeljuje slijedećem zadatku iz lokalne ili globalne liste čekanja za procesnu jedinicu prema algoritmu koji je isto prikazan na sl. 13.

5.8. Signaliziranje uvjeta

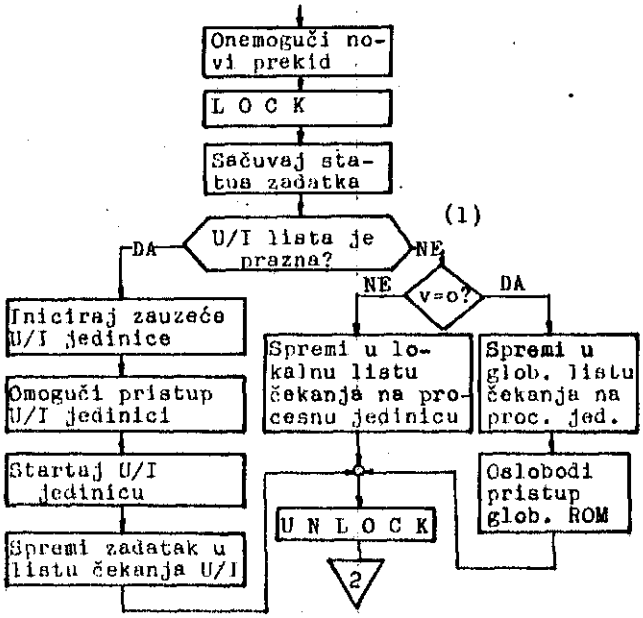
U trenutku kada zadatak oslobodi npr. globalni memorijski modul ili pristup programskom odsječku za izračunavanje specijalne funkcije u nekom trenutku svog izvođenja on signalizira ispunjenje uvjeta. Signaliziranje uvjeta realizirano je preko rezervirane memorijske lokacije - semafora UV. Nakon spremanja statusa zadatka u privremene lokacije ispituju se lokalna i globalna lista čekanja za taj uvjet i svi zadaci u tim listama prebacuju se u liste čekanja za procesnu jedinicu globalnu i lokalnu, inicirajući time da ti zadaci mogu nastaviti sa radom kada na njih dođe red. Posljedica toga je da će lokalni zadatak sa najvećim prioritetom prvi naći uvjet slobodnim i tako nastaviti sa radom. Svaki od uvjeta ima rezerviranu memorijsku lokaciju u lokalnom RAM-u i globalnom RAM-u u kojoj se nalazi adresa deskriptora prvog zadatka u listi čekanja na taj uvjet. Lokalni zadaci koji čekaju na uvjet imaju prednost izvršenja u odnosu na globalne zadatke koji čekaju na taj uvjet. Karakteristika ovog potprograma jezgre prikazanog dijagramom na sl. 14. je da onaj zadatak koji signalizira ispunjenje uvjeta nastavlja kasnije sa radom pa se njegov status ne treba spremiti u deskriptor zadatka nego u privremene memorijske lokacije, što pojednostavljuje i skraćuje trajanje potprograma. Integritet zadatka koji je signalizirao uvjet je osiguran time da potprogram ne može biti prekinut do svog konačnog završetka.



Sl.14. Signaliziranje uvjeta

5.9. Startanje U/I jedinice

Jezgro operativnog sistema ima mogućnost komuniciranja sa ulazno-izlaznim jedinicama preko dvije primitivne funkcije "startaj U/I jedinicu" i "signaliziraj U/I gotovo". Pretpostavka je da se samo jedan od zadataka može naći u listi čekanja za ulazno-izlaznu jedinicu (slika 15). Prije startanja U/I jedinice provjeravamo da li je lista prazna (1). Ako je, iniciramo zauzeće U/I jedinice, omogućavamo pristup, startamo jedinicu i spremamo zadatak u listu čekanja za tu U/I jedinicu. U slučaju da je lista već zauzeta zadatak se smješta u listu čekanja za procesnu jedinicu, globalnu ili lokalnu, ovisi o zadatku koji starta ulazno-izlaznu jedinicu, i tu čeka ponovo svoj red za izvođenje.



Sl.15. Startanje ulazno-izlazne jedinice

Kod U/I jedinica koje su fikšno pridodijeljene procesnim jedinicama programski odsječci LOCK i UNLOCK se ispuštaju. Za svaku U/I jedinicu postoji po jedan ovakav programski odsječak. Kod U/I jedinica koje su pristupne svim procesnim jedinicama U/I lista je globalna pa se prema tome moraju koristiti mehanizmi LOCK i UNLOCK za pristup globalnoj memoriji. Startanje U/I jedinice povlači za sobom omogućavanje i zadržavanje pristupa U/I jedinici sve dok ona ne inicira da je gotova. U slučaju globalnog zadatka to ujedno i znači zadržavanje prava pristupa globalnom ROM-u u kojem se nalazi globalni zadatak. Na taj način globalni zadatak zadržava sva sredstva potrebna za njegov završetak (U/I jedinicu i ROM). Ako globalni zadatak traži pristup U/I jedinici i nađe da je zauzeta to znači da je upotrebljava lokalni zadatak. Globalni zadatak se sprema natrag u listu čekanja globalnih zadataka i oslobađa globalni ROM (preko semafora i zastavice F 3).

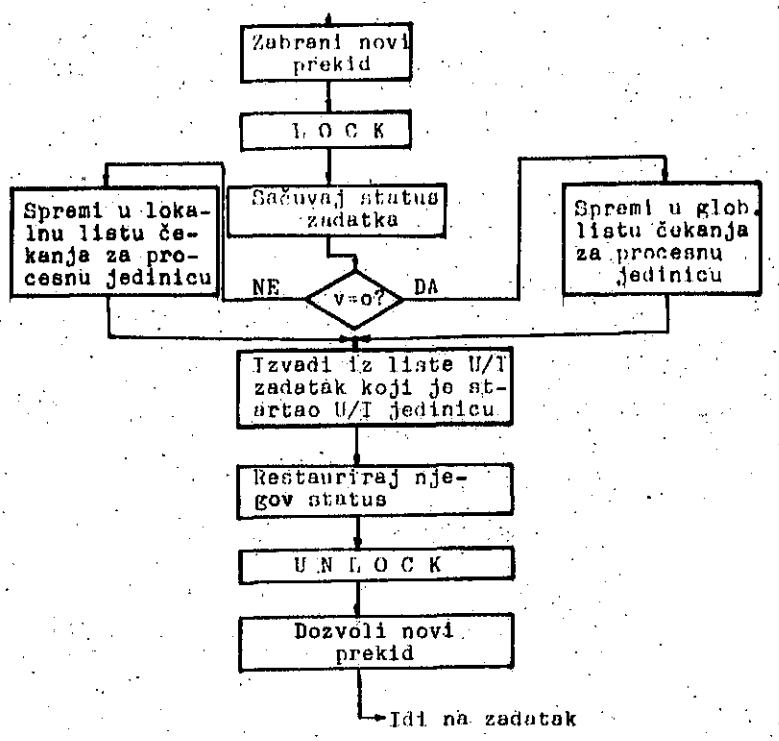
5.10. Signaliziranje da je U/I gotova

Signaliziranje da je ulazno-izlazna jedinica obavila zadatak omogućeno je preko vektorskog prekida. Posljedica toga je ulazak u jezgro operacionog sistema i izvršavanje primitivne funkcije "signaliziraj U/I gotovo". Karakteristika te funkcije (sl.16) je spremanje zadatka koji je do tada bio aktivan u listu čekanja SPREMNO i dodjeljivanje procesne jedinice zadatku koji je startao U/I jedinicu.

Vidljivo je da za svaku ulazno-izlaznu jedinicu mora postojati posebna lista čekanja i posebna prekidna linija. Vektorski prekid omogućava automatsko prepoznavanje ulazno-izlazne jedinice tako da za to nije potreban posebni programski odsječak. Za programske odsječke LOCK i UNLOCK i liste, vrijedi isto što i u potprogramu za startanje ulazno-izlazne jedinice. Nakon signaliziranja da je U/I jedinica gotova nastavlja se zadatak koji je dao start te se u samom zadatku izvršavaju potrebne radnje sa U/I jedinicama (npr. čitanje podataka iz ADC-a). Nakon predviđenih radnji mora se izvršiti V operacija, tj. oslobađanje U/I jedinice. Ukoliko globalna U/I jedinica služi npr. za prijenos informacija onda V operacija može biti sadržana u primitivnoj funkciji "signaliziraj U/I gotovo" jer se u nastavku zadatka U/I jedinica više ne koristi.

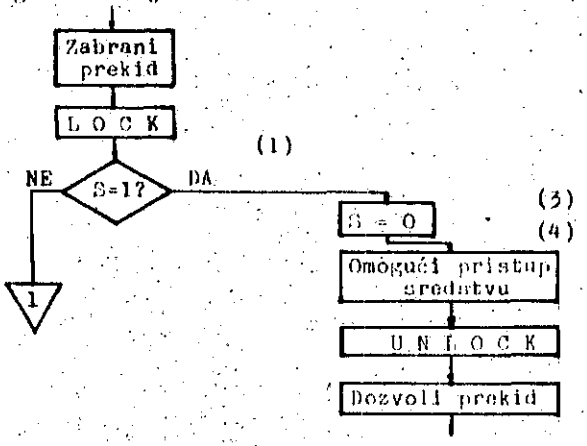
5.11. P i V mehanizmi

Vidjeli smo da je ulazak u kritični odsječak globalnog RAM-a MI osiguran sklopovskom arbitražom pomoću LOCK i UNLOCK programskih odsječaka. Jednoznačan pristup RAM-u osigurava i pristup rezerviranim memorijskim lokacijama-semaforima, kojima kontroliramo ulaske u ostale kritične odsječke. Svako globalno sredstvo; memorijska jedinica, ulazno-izlazna jedinica, pa prema tome i programi u memorijskim jedinicama ponašaju se kao kritični odsječci kontrolirani od semafora. Ako na primjer neki lokalni zadatak u toku svoga izvođenja treba pristup nekom potprogramu u globalnom ROM-u on mora najprije tražiti dozvolu za pristup preko semafora da bi bio siguran da taj ROM ne upotrebljava već neki drugi zadatak.



Sl. 16. Signaliziranje da je U/I gotova

Dozvola za pristup se ne treba tražiti jedino u slučaju da su programski odsječci koje mogu upotrebljavati više zadataka-reentrant programski odsječci. Za pristup semaforu mora "proći" sklopovsku arbitražu za RAM u kojem se nalazi semafor. Ta procedura uvjetovana je specifičnom realizacijom višeprocesorskog sistema sa posrednim pristupom. Koncept semafora prihvaćen je kao efikasno sredstvo zaštite između zadataka koji kooperiraju i koji se natječu za ista globalna sredstva /3/. Ulazak u kritični odsječak može se ostvariti preko binarnog semafora u P - primitivnoj funkciji jezgre na ovaj način:

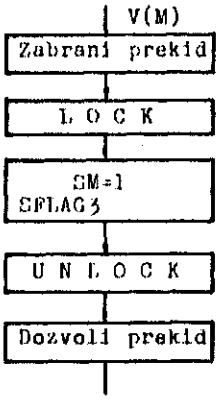


Sl. 17. P mehanizam

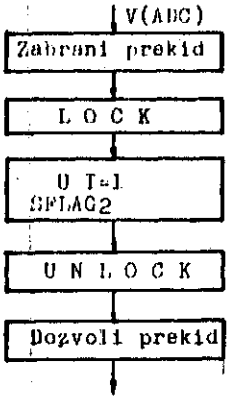
Binarni semafor može poprimiti vrijednost samo O ili 1. Prije ulaska u kritični odsječak zadatak izvršava P primitivnu funkciju. Nakon isključivog dobijanja pristupa semaforu ispituje se njegov sadržaj (1). Ako je O to znači da je to sredstvo zauzeto te se zadatak sprema u listu čekanja za to sredstvo. Ako je sadržaj semafora 1 to znači da je sredstvo slobodno i da zadatak može ući u kritični odsječak. Nakon dobijanja dozvole i iniciranja zauzeća globalnog sredstva preko semafora (3) eksplicitno se omogućuje (4) pristup tom sredstvu pomoću zastavice F (vidi sl. 1). Tek nakon toga može se nastaviti normalno izvršenje zadatka. Velika prednost ovako koncipirane P operacije je da zadatak ne čeka ulazak u kritični odsječak u petlji čekanja već oslobađa procesnu jedinicu za druge zadatke, a on se smješta u listu čekanja za taj odsječak i na taj način prelazi u stanje BLOKIRAN.

Nakon završetka rada u kritičnom odsječku zadatak izvršava V primitivnu funkciju, tj. inicira preko semafora i zastavice da su taj odsječak i globalna memorija u kojoj se nalazi odsječak slobodni. U toku daljeg izvođenja programa ili na završetku, u slučaju da je globalno sredstvo memorija ROM (sl. 18), potrebno je pozvati primitivnu funkciju "Signaliziranje uvjeta" koja zadatke koji su čekali na taj uvjet sprema u listu čekanja SPREMNO

U slučaju globalne U/I jedinice (ADC-a, sl. 19) potrebno je memorijsku lokaciju koja inicira zauzeće U/I jedinice postaviti na 1 i osloboditi sredstvo preko zastavice.



Sl. 18



Sl. 19

6. ZAKLJUČAK

Iz svih primitivnih funkcija koje su implementirane u jezgri operativnog sistema vidljivo je da se u jezgru može ući na jedan od tri načina:

- prekidom uslijed satnog mehanizma
- pozivom iz zadatka ili
- prekidom U/I jedinice

Svaka procesna jedinica ima u svom lokalnom ROM-u svoju privatnu jezgru operativnog sistema. Karakteristika takve organizacije je da zadaci koji su aktivni, tj. koji se izvršavaju na raznim procesnim jedinicama mogu neovisno jedan od drugog ući u jezgru. Na taj način izbjegnuto je rješavanje konflikata oko ulaska u jezgru ako bi postojala samo jedna kopija u globalnom ROM-u. Kako globalni zadaci mogu biti dodijeljeni bilo kojoj procesnoj jedinici to se oni sami, njihovi deskriptori i pripadne liste čekanja moraju nalaziti u globalnoj memoriji. Programske strukture u globalnom RAM - MI su zaštićene na taj način da samo jedna kopija jezgre operativnog sistema u nekom trenutku ima pristup tim strukturama dok ostale moraju čekati na njihovo oslobađanje. Ovako koncipirana jezgra zauzima manje od 1 K memorije bez programskog dijela za inicijalizaciju.

7. LITERATURA

/1/ R.C. HOLT, G.S. GRAHAM, ED LAZOWSKA, M.A. SCOTT, "Structured concurrent Programming with Operating Systems Applications", Addison-Wesley Company, 1978.

/2/ M. JELAVIĆ, "Raspoređivanje procesa u računalima s više procesora" Magistarski rad, Elektrotehnički fakultet, Zagreb, 1982.

/3/ EDWARD G. COFFMAN, JR., PETER J. DENNING, "Operating System Theory" Prentice-Hall, Inc. Englewood Cliffs, New Jersey, 1973, p. 1-82

/4/ M. JELAVIĆ, "Višeprocesorski sistemi s mikroprocesorima TM6100" Informatica (u štampi)

OBJEKTNA ARHITEKTURA NA OSNOVI SEMANTIČNE ZGRADBE INFORMACIJE

V. ŽUMER,
P. KOKOL,
L. PIPAN

UDK: 681.3.07

VISOKA TEHNIŠKA ŠOLA, MARIBOR, JUGOSLAVIJA
FAKULTETA ZA ELEKTROTEHNIKO, LJUBLJANA, JUGOSLAVIJA

Članek podaja semantično zgradbo informacije, kjer imajo objekti omejitev, lastnost, atribut in predstavitev. Prikazana je arhitektura za nekatere značilne objekte. Ta koncept je zelo primeren za zanesljivo neposredno izvajanje.

OBJECT ARCHITECTURE ON THE SEMANTIC STRUCTURE OF INFORMATION. The paper gives the semantic structure of information where objects have scope, property, attribut and representation. Storage architecture for some interesting objects is described. This concept for relisble direct execution of HLL programs is appropriate.

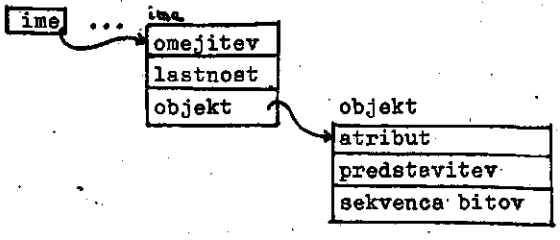
1. UVOD

Kljub izrednemu razvoju elektronskih vezij po eni strani in razvoju programskih jezikov in tehnik programiranj po drugi strani, večina računalnikov deluje na principu klasičnega von Neumannovega koncepta. Posledica tega je, da imamo danes še zmeraj velik semantični razkorak med uporabnikovim okoljem in materialno opremo. Do sedaj znane izboljšave von Neumannovega koncepta v zgradbi informacije so značke (1), ki vnašajo atribut podatka in deskriptorji (2), s pomočjo katerih se dosegajo podatkovne strukture. Prav tako je znan koncept capability adresiranja (3), s katerim se doseže pravilno dodeljevanje informacije. Še višjo abstrakcijo podatkov zasledimo v zadnjem času s tako imenovano objektno orientirano arhitekturo, kjer imamo namesto linearnega pomnilnika množico objektov.

Vse navedene sheme pa uporabljajo še vedno fiksno dolžino informacije kot npr. zloge ali besede. Slabost tega je, da ni mogoče opravljati operacij med različno dolgimi podatki. Prav tako ni mogoče dinamično spreminjati narave objektov. Izboljšanje navedenih slabosti dosežemo s semantično zgradbo informacije, ki se zrcali že v arhitekturi računalnika.

2. SEMANTIČNA ZGRADBA INFORMACIJE

Semantično zgradbo informacije predstavimo kot množico odnosov med dosegom in vrednostjo (4). Odnose razdelimo v štiri hierarhične kategorije: omejitev, lastnost, atribut in predstavitev (sl. 1). S tem je običajno definirani tip razdeljen na lastnost in atribut.



Slika 1

Ime vsebuje omejitev, lastnost ter kazalec na objekt, sam objekt pa vsebuje atribut, predstavitev in sekvenco bitov. Omejitev definira, na kakšen način in komu je dosegljiva informacija preko imena. S tem dosežemo pravilno dodeljevanje informacije in zaščito. Predstavitev daje možnost spremenljive dolžine in strukture informacije. Iz sekvence bitov dobimo glede na atribut in predstavitev vrednost objekta ali kazalec na drug objekt.

Imamo dve osnovni operaciji: asocijacijo in evaluacijo. Asocijacija poveže ime in objekt ter se uporabi tedaj, kadar spremenljivka dobi neko vrednost. Če je bilo ime povezano z objektom, asocijacija sprosti stari objekt in poveže ime z novim objektom. Pri asocijaciji je treba najprej pregledati pravico dosega glede na omejitev v imenu in nato še primernost asocijacije glede na lastnost v imenu. Z evaluacijo se zgradi nov objekt in to pri kakršnih-koli opracijah s starimi objekti. Glede na vrednost omejitve v imenu se ugotavlja primernost evaluacije in možnost operacije. Novi objekt ima samo atribut in predstavitev, ne more pa imeti lastnosti niti omejitve tako dolgo, dokler se ne poveže z imenom.

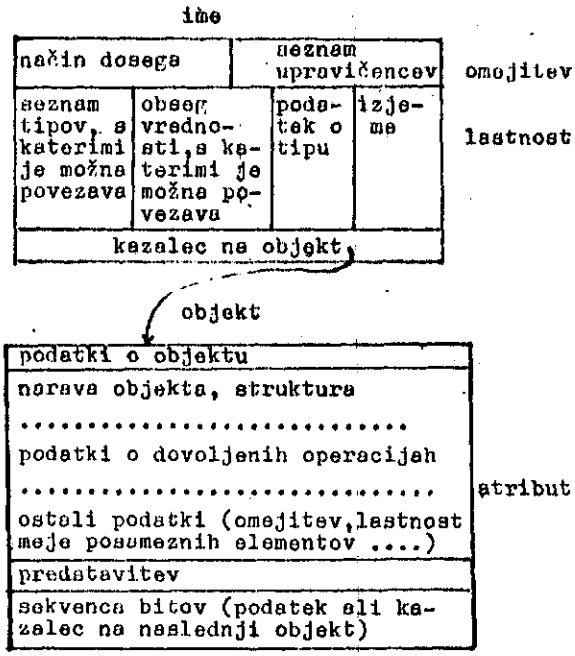
Omejitev vsebuje dva dela: način dosega in seznam upravičencev, ki lahko dosežejo to ime. Zlasti pri dodeljenih in porazdeljenih informacijah omejitev omogoča večjo zanesljivost izvajanja.

Lastnost definira odnos med imenom in množico objektov, ki se lahko povežejo s tem imenom. Lastnost torej določa, kateri atribut v objektu in kakšne vrednosti se lahko povežejo z imenom. V lastnosti se lahko podajo razne izjeme, ki so morda potrebne zaradi implementacije arhitekture.

Atribut definira odnos med objektom in med množico dovoljenih operacij s tem objektom. Atribut določa naravo objekta kot je npr. procedura, celo ali realno število, prav tako pa določa tudi strukturo objekta kot npr. polje, zapis, itd.

Predstavitev določa odnos med objektom in nje-

govo fizično predstavitvijo. Predstavitev določa osnovo številskega sistema, dolžino števila, način zapisa kodiranja, način kako je predstavljeno polje, dolžino dinamičnih struktur itd.



Slika 2

Na sliki 2 prikazujemo podrobnejšo zgradbo imena in objekta.

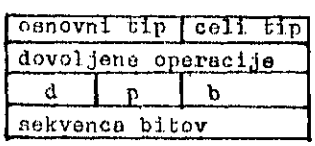
Podatki o objektu vsebujejo dodatno informacijo o objektu, ki je potrebna pri upravljanju s pomnilnikom (dolžina objekta, število segmentov objekta, aktivnost objekta, ...).

3. ARHITEKTURA OBJEKTOV

Predpostavljamo dovolj velik pomnilnik, v katerem je seklјalna tabela in množica naključno razporejenih imen in objektov. Takšen koncept se izkaže kot zelo uporaben pri neposrednem izvajanju visokega programskega jezika, saj moramo v tem primeru preglјedovati primernost evaluacije oz. asocijacije tik pred izvajanjem.

V našem primeru so objekti lahko različne podatkovne strukture, seklјalne tabele, okviri, slike procedur in krmilnih konstruktov kot so npr. if, while, when.

Objekt za celi tip (integer)



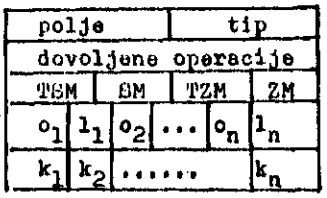
d - dolžina sekvence bitov
b - baza številskemu sistemu
c - predznak števila

Objekt za sestavljeni tip polja (array)

Zaradi enostavnosti implementacije je polje deklarirano enodimenzionalno, več dimenzij dosežemo z rekurzijo.

Splošni stavek za deklaracijo polja je:

ime is array SM:TSM ... ZM:TZM of tip



SM - spodnja meja indeksa polja
TSM - tip SM
ZM - zgornja meja indeksa polja
TZM - tip ZM
$o_1, o_2 \dots o_n$ - omejitve posameznih elementov polja
$l_1, l_2 \dots l_n$ - lastnosti posameznih elementov polja
$k_1, k_2 \dots k_n$ - kazalci na naslednje elemente tega polja, ki so lahko objekti osnovnih tipov ali pa objekti sestavljenih tipov

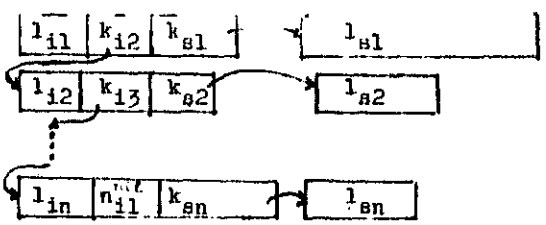
Objekt krmilnih stavkov

Podobno kot za podatkovne strukture, lahko tudi za krmilne stavke upoštevamo semantiko že v arhitekturi. Objekt krmilnega stavka ne potrebuje imena, ima samo vstop v seklјalni tabeli.

Splošna oblika krmilnega stavka je:

struct	logični izraz 1 do lista stavkov 1
else	logični izraz 2 do lista stavkov 2
⋮	
else	logični izraz n do lista stavkov n
end	

Objekt krmilnega stavka vsebuje več objektov:



l_i - logični izraz
l_s - lista stavkov
k_i - kazalec na logični izraz
k_s - kazalec na listo stavkov

4. ZAKLJUČEK

Z navedeno arhitekturo je mogoče zelo zanesljivo izvajanje programov, s čimer se zmanjša cena vzdrževanja programov. Prav tako se poveča učinkovitost izvajanja programov in zaščite informacije. Če zahtevamo izvajanje programov z veliko zanesljivostjo, potem je v navedenem konceptu izvajanje programov mnogo hitrejše kot pa v običajnih računalnikih. To je razumljivo, ker je koncept preverjanje vnesen že v arhitekturo, kjer se dinamično preverjajo omejitev, lastnost, atribut in predstavitev. V primeru, da želimo enako zanesljivost doseči z običajnimi računalniki, opravljajo to delo programi v okviru operacijskega sistema.

Današnja VLSI tehnologija nam ponuja pomnilnike z velikimi kapacitetami na osnovi asociativnega dosega in z dodatno logiko. Ceneni mikroprogramirani procesorji pa omogočajo izdelavo procesorjev za naše posebne namene. Razvoj tehnologije kaže, da bo v nekaj letih implementacija objektne arhitekture kljub kom-

pleksnosti tudi ekonomsko upravičena.

LITERATURA:

1. E.A. Feustel, On the Advantages of Tagged Architecture, IEEE Tr. on Computer, C-22(7), 1973

2. T.A. Welch, An Investigation of Descriptor Oriented Architecture, Proc. of the Third Annual Symposium on Computer Architecture (1976)

3. J.B. Dennis, E.C. Van Horn, Programming Semantisc for Multiprogrammed Computations, CACM 9(3), March 1966

4. M. Tokoro, T. Takinaka, On the semantic structure of information, 9-th Annual Symposium on Computer Architecture, Austin 1982

5. V. Žumer idr., Raziskava mikroprogramirane arhitekture za implementacijo visokega programskega jezika, raziskovalna naloga, VTŠ, Maribor 1982

EXPERIMENTAL LOCAL AREA NETWORK

JERZY BRZEZINSKI,
WOJCIECH CELLARY,
JERZY KREGLEWSKI

UDK: 681.324

INSTITUTE OF CONTROL ENGINEERING
TECHNICAL UNIVERSITY OF POZNAN, POZNAN,
POLAND

Abstract

Modelling and performance evaluation of local area networks' architectures and protocols indispensable for their optimisation, cannot be achieved by classical single-computer programming simulation because the real-time to simulation-time coefficient is greater than 10^4. Thus, the development of distributed simulators improving this coefficient is necessary. In this paper an experimental local area network Exnet is presented for distributed modelling of local area networks with an arbitrary architecture and protocols. The architecture and protocols of Exnet are presented, as well as the hybrid technique of simulation and emulation for distributed modelling of local area networks. This technique allows all time parameters of all protocol functions of a modelled network to be measured. As an example, modelling on Exnet of the well-known local area network Ethernet composed of an arbitrary number of stations is described in detail.

1. Introduction

The growing demand for local area networks necessitates the elaboration of optimal methods for network design. These methods must take into account the influence of network application characteristics. The network design comprises problems like: network architecture, transmission protocols, data rate, packet structure, packet length, etc. The choice of appropriate parameters and the optimal design of a network to be applied require experiments on models. The design and implementation of physical models is obviously very expensive and, therefore, impracticable. Another way to obtain a model is simulation. In practice, for that purpose computer simulation is used. However, if a whole computer network is simulated on one computer site, the real-time to simulation-time coefficient grows to over $1:10^4$. Therefore, this approach is very difficult and also expensive because of time consumption. The above consideration shows the necessity of elaborating a new method of network modelling. In this paper we propose a new method of network modelling on a distributed simulator. For the construction of this distributed simulator the local area network Exnet is used. It has been designed for research purposes. The design of Exnet is intended to enable one to recognize the behaviour and the performance of a distributed system with an arbitrary architecture. This new modelling method is connected with several new problems which must be resolved. The area in which the new problems would appear is the compound technique of emulation and simulation used in this modelling method.

In this paper we present our approach using as an example the modelling of the well-known local area network Ethernet.

In Section 2 we describe the architecture and transmission protocols of the Exnet network on which the experiment is performed.

In Section 3 of the paper we briefly present the architecture and protocols of the Ethernet network.

In Section 4 we present our approach to modelling an arbitrary network on the Exnet network, using Ethernet as an example.

In Section 5 we present conclusions and final remarks.

2. Exnet network - architecture and protocols

In this section we describe the architecture and the transmission protocols of the experimental network Exnet.

The main assumptions underlying the design of Exnet are the flexibility of the network structure and low requirements for the network adapter and host performance. To ensure good connectivity, a global bus structure of Exnet is chosen. In this structure all stations are connected to the shared passive transmission channel and, therefore, the channel access procedure becomes very important for effective sharing of the channel. Since we do not distinguish the supervisor station and the channel

is passive, a fully distributed channel management can lead to collisions. A collision appears when two or more stations simultaneously start to transmit a message through the channel. In this case, the transmission must be annulled and restarted in a manner which avoids subsequent collisions.

In the procedure used in Exnet, when any station wishes to send a packet, a carrier sense mechanism is used first, forcing the station to defer if any transmission is in progress. If no other station is transmitting, the sender can begin immediately, with zero latency. Otherwise, the sender waits until the packet has passed before transmitting.

For detection of a collision a special identity mark is sent at the beginning of each packet. The identity mark of each station is unique in the whole network. The station which starts to transmit sends this mark and checks for its correctness by monitoring the channel. If any other station starts to transmit a packet, it sends its own identity mark, which is different from that of the first station. When they start to transmit simultaneously, a collision occurs. Collision detection in all the colliding stations is ensured by the special structure of the identity mark. It is composed of two bytes of which the first is the unique address of a station and the second is its complement. With this identity mark, for each pair of colliding stations there is at least one bit equal to zero in the identity mark of the first station which matches one bit equal to one in the identity mark of the second station, and vice versa. Therefore, the detection of a collision does not depend on which of the two states is forced in the channel during the collision. Let us notice that if the identity mark comprised only the address, it is possible that the colliding state would match one of the identity marks and the respective station would not detect the collision. It must be noted here that in Exnet the transmission in the channel is asynchronous, its data rate is low (19 200 baud) and, therefore, a packet does not have any preamble which is used for synchronization in synchronous transmission. Low data rate gives another advantage, that is, the propagation delay in the channel is much lower than one bit transmission time period. Therefore, all channel signals are seen by all the stations in the network simultaneously. In such a network, a collision can occur only at the beginning of a packet, and only the transmission of the identity mark must be monitored by the station.

The packet structure is given in Figure 2.1. The identity mark mentioned above begins the packet and gives the source address information to the destination station. The identity mark is followed by the destination address field, then the field of null-bytes separates the destination address and the data field. The null-bytes are used for reducing the time requirements for the host acknowledgement of the interrupt from the address detector. The number of null-bytes can be adjusted to the host performance and can vary for different stations in the network. The data field with the 2-byte CRC forms the rest of the packet.

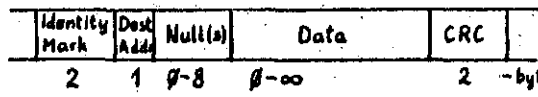


Figure 2.1. Exnet packet structure

In the next section we present the Ethernet architecture and protocols for comparison with those of Exnet described above.

3. Ethernet network - architecture and protocols

In this section we briefly describe the well known local area network Ethernet.

Ethernet is a local area network with a global bus structure and Carrier Sense Multiple Access with Collision Detection (CSMA/CD) access procedure.

In the CSMA/CD access procedure, when any station wishes to send a packet, a carrier sense mechanism is used first, forcing the station to defer if any transmission is in progress. If no other station is transmitting, the sender can begin immediately, with zero latency. Otherwise, the sender waits until the packet has passed before transmitting. Collisions can occur when two or more stations defer the transmission and, after stating that the channel is idle, they start to send their packets simultaneously. Each controller, however, controls its own transmission by monitoring the channel and can provide collision detection when the channel's signal does not match its own output. In this case, each station breaks the transmission, forces the channel into the collision state to ensure that all other colliding stations have seen the collision, and then stops.

In future, the stations will try to retrans-

mit; however, to avoid successive collisions during retransmission, a random period of time is taken as a delay of the retransmission.

The packet structure of Ethernet is shown in Figure 3.1.

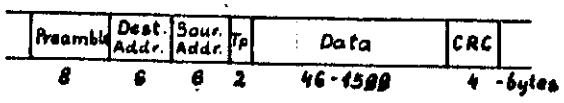


Figure 3.1. Ethernet packet structure

The preamble is a sequence of zero-one bit patterns and is used for synchronization of the receiver clock. This preamble appears only at the Physical Layer. The remaining fields of the packet form a packet at the Data Link Layer. The destination and source address fields perform their usual functions, the use of the type field is reserved for higher levels, and the data and CRC fields perform their usual functions too.

In the next section we show a method of modelling Ethernet on the experimental Exnet network.

4. Modelling arbitrary network on Exnet

In this section we show how to model an arbitrary network on the experimental Exnet network, using Ethernet as an example.

The architecture of the Ethernet network is consistent in its foundations with the ISO 7-layer Open-System Architecture reference model [2]. The Ethernet standard specification [1] provides precise detailed definitions of the two lowest layers, namely, the Physical Layer and the Data Link Layer of the overall network architecture.

At the Physical Layer the following functions are performed:

1. Data encoding
   - preamble generation/removal (for synchronization)
   - bit encoding/decoding (between binary and phase-encoded form)
2. Channel access
   - bit transmission/reception (of encoded data)
   - carrier sense (indicating traffic in the channel)
   - collision detection (indicating contention in the channel);

and at the Data Link Layer:

1. Data encapsulation
   - framing (frame boundary delimitation)
   - addressing (handling of source and destination addresses)
   - error detection (detection of physical channel transmission errors)
2. Link management
   - channel allocation (collision avoidance)
   - contention resolution (collision handling)

We intend to achieve a model of the Ethernet network in which the performance of each function at each layer will be measurable. For this reason, we use a compound technique of emulation and simulation for modelling the network. Emulation is used for the first, Physical Layer, and for higher layers only simulation is used.

We will now present in detail the particular problems arising during Ethernet modelling on the Exnet network.

In general, several Ethernet stations can be modelled on one Exnet station; however, in order to improve the clarity of presentation, we will first consider modelling one Ethernet station on one Exnet station.

Let us note that the implementation of the particular functions in the Ethernet and Exnet networks is different. For the Physical Layer, the data encoding function is implemented in the Ethernet network only. In Exnet, neither preamble generation/removal nor bit encoding/decoding is used; instead, another function appears: the function of byte framing. A byte frame consists of one start bit, one parity bit and one stop bit for each 8-bit byte transmitted in the channel.

The channel access function is implemented in both networks in nearly the same manner. The differences appear in the time characteristics, which depend on data rate and propagation delay in the channel. These time characteristics are also different for the two networks because of the synchronous and asynchronous types of transmission in the channel, and because of the different packet structures.

These are the main differences in the Physical Layer functions. Therefore, an Ethernet station can be modelled on an Exnet station by supplying the functions of the Physical Layer of the Exnet network with three procedures:

- Propagation Time Rectification (PROP)
- Synchronous Transmission Rectification (SYNT)
- Packet Structure Rectification (PACS).

The particular functions of these procedures are as follows:

The PROP procedure

In the Exnet network events occur in the

whole network simultaneously because of the relatively low data rate in comparison with propagation time of signals in the channel. In Ethernet, however, the data rate is high enough for the propagation delay in the channel to be observed. The assumed maximum round trip propagation delay in this network could be of 450 bit times. The task of the PROP procedure is, therefore, a simulation of propagation delays in the modelled Ethernet network. The length of each cable connection is defined in advance for the modelled network and the appropriate delay rectification is computed.

The SYNT procedure

In the Exnet network the transmission is asynchronous and, therefore, 11 bits are used for transmission of 8 bits of data. The three additional bits are the start, parity and stop bits. In the Ethernet network the transmission is synchronous and no additional bits are required for data transmission. Therefore, the SYNT procedure is used for computing the correct transmission time in the modelled network.

The PACS procedure

The packet structures for Ethernet and Exnet were described in Sections 2 and 3. The PACS procedure is thus used for computing a rectification of the transmission time for these two different packet structures.

Now we will explain the method of modelling of the above-mentioned functions of the Ethernet network.

The main problem which appears in this modelling is computing the time of event occurrence related to the Ethernet functions, on the basis of event occurrence in the real Exnet network. In order to obtain time parameters of the bit transmission/reception of the modelled network, we must include all three rectifications in this function.

In the carrier sense function we want to determine the moment of the carrier on and carrier off detection in the modelled network on the basis of the time of this signal occurrence in Exnet. This moment depends on the moment of packet transmission termination and, therefore, the carrier sense function must be supplied with all three procedures listed above.

However, the collision detection function depends solely on the moment of packet transmission initiation and, therefore, it must be supplied with the PROP procedure only.

At the end of our considerations relating to the first layer modelling we refer to the data encoding/decoding function. In Ethernet, this function is used, in the encoder, to translate physically separate signals of the clock synchronization and data into a single, self-synchronizable serial bit stream, suitable for transmission through the coaxial cable by the transceiver, while in the decoder it is used to separate the incoming phase encoded bit stream into a data stream and a clock signal. This function does not influence time characteristics and, therefore, is not included in the simulation.

Passing to the higher layers of the Ethernet network, it must be noticed that on the interface between the Physical Layer and the Data Link Layer information has the format adequate for Ethernet. The higher layers could be modelled by the simulation method and, therefore, they do not need a specific approach.

Let us now briefly discuss the problem of modelling several Ethernet stations on one Exnet station. Then, in the case of transmission from one Ethernet station to another, with both Ethernet stations modelled on the same Exnet station, such transmission is physically performed and addressed by the Exnet station to itself.

The transmission termination or interruption on account of a collision in the modelled network carries information for other stations in the distributed simulator. Such transmission is seen by the other stations of the Exnet network as a transmission between two Exnet stations.

5. Conclusions

The local area network Exnet constitutes a distributed simulator of arbitrary local area networks. It permits modelling and performance evaluation of network architectures and protocols in an efficient way. Thus, it can be used for designing networks best suited for a given application.

References

[1] The Ethernet - A local area network - data link layer and physical layer specification. Version 1.0. DEC, Intel and Xerox (September 1980)

[2] Open System's Architecture Reference Model ISO TC97/SC16/N34

RAČUNALNIK V ŠOLI

ALEKSANDER COKAN
VLADISLAV RAJKOVIČ

UDK: 378.681.3

ZAVOD SR SLOVENIJE ZAŠOLSTVO, LJUBLJANA
VISOKA ŠOLA ZA ORGANIZACIJO DELA KRANJ
IN INSTITUT „J. STEFAN", LJUBLJANA

Članek obravnava mesto in vlogo računalniških znanj in računalniške tehnike v učno-vzgojnem procesu na osnovni in srednji stopnji. Problemi in nekatere možne rešitve glede na naše možnosti in potrebe so podani na osnovi več kot 10 letnih izkušenj na slovenskih srednjih šolah.

COMPUTER AT SCHOOL. In this paper the role of computer subjects and technology at the secondary and primary educational level is discussed. Problems and some possible solutions regarding our needs and possibilities are treated on the base of more than 10 years practical experience in computer teaching at secondary schools in Slovenia.

1. UVOD

Ne da bi ponovno poudarjali pomen sodobne računalniške tehnike, lahko ugotovimo, da je ta tu in da se ji tudi v šoli ne moremo in ne želimo izogniti. Takoj se nam zastavljata vsaj dve vprašanji. Prvo je, kaj lahko od računalniške tehnike pričakujemo v vzgojnoizobraževalnem procesu, in drugo, kakšna znanja naj bi o tej tehniki posredovali mladini.

Odgovora na zastavljeni vprašanji nikakor nista kaka slovenska posebnost, vendar bi ju v prispevku želeli osvetliti tudi z izsledki uvajanja pouka računalništva v slovenske srednje šole.

Računalniško izobraževanje ima v srednjih šolah v SR Sloveniji že več kot desetletno tradicijo [9,10,11,12]. In čeprav v izobraževalnem procesu deset let ni dolga doba, saj do sprememb navadno vodijo majhni koraki, pa lahko za srednješolsko računalniško izobraževanje rečemo, da smo v tem času priča večjemu razmahu, tako zaradi dinamike razvoja celotnega računalniškega področja, kot zaradi uvajanja usmerjenega izobraževanja.

2. POUK S POMOČJO RAČUNALNIKA

Ob prvem vprašanju, kaj lahko od računalniške tehnike pričakujemo v vzgojnoizobraževalnem procesu, običajno pomislimo na programirani pouk s pomočjo računalnika (CAI - Computer Assisted Instruction). Programirani pouk je v osnovi sestavljen iz kratkih poglavij učne snovi, vsakemu poglavju pa sledi test. Glede na rezultate tega testa lahko učenec bodisi nadaljuje s snovjo ali pa se mora še ukvarjati z že predelano snovjo. Če je gradivo za programirani pouk shranjeno v računalniku, lahko uporabimo računalnik za hkratno poučevanje celega razreda učencev, vendar tako, da vsak učenec napreduje glede na dosežene rezultate. Vsak učenec komunicira z računalnikom, npr. preko svojega zaslonskega (video) terminala, na katerem se prikazuje učno gradivo in testna vprašanja, odgovore pa učenec sporoča računalniku preko tastature ali kako drugače. Če so odgovori zadovoljivi, se na zaslonu pokaže naslednja učna snov, sicer pa dodatno gradivo za pojasnilo k neusvojeni snovi. Tudi učenec sam lahko zahteva razne ponovitve ali pojasnila v okviru gradiva, shranjenega v računalniku.

Priprava takega gradiva, tj. programiranih učnih enot, je izredno zahtevno in dolgotrajno delo. Zahtevnost se ne nanaša na računalniški del, ampak na ustrezno metodično in vsebinsko pripravo gradiva. Zato so taki programski sistemi običajno dragi, njihova obsežnost pa zahteva sorazmerno zmogljivo računalniško opremo na šolah.

Zaradi omenjenih problemov programiranega učenja in nekaterih nespornih prednosti računalniške opreme se je v šolah uveljavil širši pojem pouka s pomočjo računalnika (CAL - Computer Assisted Learning), ki obsega pomoč računalnika v vzgojnoizobraževalnem procesu povsod tam, kjer je to mogoče in smiselno. Osnovno izhodišče je izbrana učna situacija. Prednosti, ki jih nudi računalnik pa so: individualizacija in posamezniku prilagojen tempo poučevanja, takojšnji rezultati, zbiranje podatkov, motiviranje učencev, koncentrirano učenje, skrajševanje učnega časa ter simulacije, ki nadomeščajo drago opremo in nevarne praktične vaje.

Eksperimenti so pokazali, da so različni mediji, kot so film, televizija, računalnik, ipd., lahko zelo uspešni v različnih učnih situacijah. Torej je pomembna opredelitev posameznega medija v različnih predmetih, tako rekoč od ene učne ure do druge. Ne gre le za dominacijo ene učne tehnologije, npr. računalniške, nad vsemi ostalimi, ampak za pedagoško učinkovit splet razpoložljivih tehnologij za čimboljše doseganje vzgojnoizobraževalnih smotrov.

Zaradi celovitosti odgovora na prvo vprašanje, kaj v šoli od računalnika lahko pričakujemo, omenimo še računalniško vodeni pouk (CML - Computer Managed Learning), ki obsega pomoč računalnika bolj v smislu šolske administracije. Gre za kompjuterizacijo aktivnosti, kot je npr. priprava in obdelava preizkusov znanja, pa vse do raznih statističnih obdelav in tiskanja spričeval z računalnikom. Tu se računalnik v šoli močno približa poslovni rabi.

Potem, ko smo spregovorili o CAI, CAL in CML, je prav, da spregovorimo še o dilemah in problemih, ki jih prinaša v šolo računalniška tehnika.

3. PROBLEMI UVAJANJA RAČUNALNIKA V ŠOLE

Dejstvo je, da so računalniki že našli pot med mladino in v šole [14,15,16]. Njihovo število nenehno narašča. Otroci se srečajo vsaj s kalkulatorjem že v predšolski dobi. Znana je zgodba, kjer otrok strmi v nekaj kalkulatorjev na mizi pred seboj, mati pa ga vpraša: "Poglej, tu imaš pet kalkulatorjev, če odvzameš dva, koliko ti jih še ostane?". Predno je otrok prepričan, da mu res ostanejo še trije kalkulatorji, to preveri s pomočjo kalkulatorja samega.

Računalnik prodira v šole hitreje, kot se zavedamo. Lahko bi dejali, da nas je računalnik med mladino celo prehitel in našel nepripravljene. Nujno je, da osredotočimo intelektualne in organizacijske napore v ustrezno izrabo računalnika v vzgojno-izobraževalnem procesu.

Nekateri trdijo, da računalnik podpira mentalno lenobo in da se bo naše šolanje zreduciralo na učenje pritiskanja gumbov. Toda izkušnje s poučevanjem otrok ob uporabi računalnika kažejo ravno nasprotno. Računalnik odpira nove aktivnosti ter vzpodbuja radovednost in kreativno sposobnost otrok. Računalnik je izredno uspešno motivacijsko sredstvo, kar je še posebej pomembno, *če vemo, kaj pomeni za uspešen pouk to, da* učenci radi pridejo v šolo in se z veseljem lotijo dela. Računalnik postaja nepogrešljiv učni pripomoček, ki pa seveda bistveno spreminja metodo dela in njegovo vsebino. V šoli prehajamo od obravnave dejstev, ki je vezana na pomnenje obilice podatkov, k reševanju problema, ki zahteva kreativno razmišljanje. To je izredna nova kvaliteta, ki pa zahteva določeno preobrazbo posameznih predmetov, pa tudi dodaten napor učiteljev.

Lahko bi rekli, da po vsebinski in metodični plati računalnik v vzgojnoizobraževalnem procesu nima meja. Edina omejitev je pomanjkanje fantazije in nezadostno poznavanje otroške psihologije.

Prišli smo tudi do odgovora na vprašanje: "Ali bo računalnik *nadomestil učitelja?*". *V splošnem je odgovor: NE! Utrjevanje* snovi z računalnikom običajno poteka učinkoviteje. Učitelj pa je nepogrešljiv vodnik pri višjih oblikah izobraževanja, kot je uporaba znanja pri reševanju problema. Mirno lahko trdimo, da vloga in pomen učitelja rasteta skupaj z njegovo odgovornostjo. Seveda pa računalnik v šoli vpliva na obstoječe didaktične principe in izobraževalne pristope pri skrajševanju poti do intelektualne zrelosti učenca.

4. RAČUNALNIŠKA ZNANJA V ŠOLI

Do sedaj še še nismo lotili drugega vprašanja, ki smo si ga zastavili takoj na začetku: "Kakšna znanja o računalniku samem naj bi posredovali učencem?". Bistvo odgovora lahko najdemo v znani metafori: "Programiranje, druga pismenost".

Prvi del te metafore je nenavadna dejavnost, čeprav je programiranje že precej popularna stroka, ki zahteva specifične zmogljivosti in obsežno šolanje, drugi del pa je splošna vrlina, najosnovnejša lastnina sodobnega človeka. Vendar pismenost in programiranje slonita na tehničnih izumih, kot sta tisk in računalnik, ter se medsebojno dopolnjujeta. Pojav tiska v človekovi zgodovini pomeni ločitev akumulacije znanja od uporabe, kar je bistveno vplivalo na doseganje novega znanja. Tudi računalnik pospešuje razvoj novih sposobnosti, algoritmičnega mišljenja, abstraktnega sklepanja, akcije, čeprav se je samo programiranje v zgodovini civilizacije pojavilo daleč *pred prvim računalnikom. Računalnik je le stimuliral* programiranje, kot je knjiga stimulirala pismenost.

Pojav mikroprocesorjev je verjetno najbolj revolucionarno tehnično odkritje dvajsetega stoletja, saj bo v bodočnosti mikroprocesor del skoraj vsakega industrijskega izdelka. Tisoči poklicev bodo morali reorganizirati svoje delo zaradi uporabe računalnika. V prihodnjosti bo človek, ki ne bo naravoslovno-tehnično izobražen, pomanjkljivo izobražen. Znašel se bo v vlogi nepismenih srednjeveških baronov, ki so se hvalili, da imajo tajnike za pisanje in računanje. Danes zna vsakdo pisati in računati, isto se bo zgodilo s programiranjem. To je druga pismenost.

Smoter poučevanja računalniških znanj bi lahko povzeli takole: bistveno je spoznavanje osnovnih principov informatike, organizacije in upravljanja ter ustreznega mesta in vloge računalniške tehnike; poleg tega pa je treba spoznavati metode in tehnike za reševanje problemov s pomočjo računalnika, tj. programiranje.

Za opredmetenje teh smotrov se nam ponujata dve možnosti. Prva je, da bi računalniška znanja strnili v samostojen predmet, druga pa, da bi ta znanja vgradili v druge predmete.

Možnost, kjer bi računalniška znanja postala del različnih predmetov, kjer je to smiselno in potrebno - predvsem v metodičnem smislu, se zdi bolj naravna. Ta varianta zahteve preoblikovanje obstoječih predmetov in seveda dodatno izobraževanje učiteljev. *V svetu se tega sicer lotevajo, vendar* v dolgoročnem smislu. Zato pa se ponuja možnost posebnega predmeta kot praktično lažja in hitreje izvedljiva.

5. RAČUNALNIŠKE IZKUŠNJE V SLOVENSKIH ŠOLAH

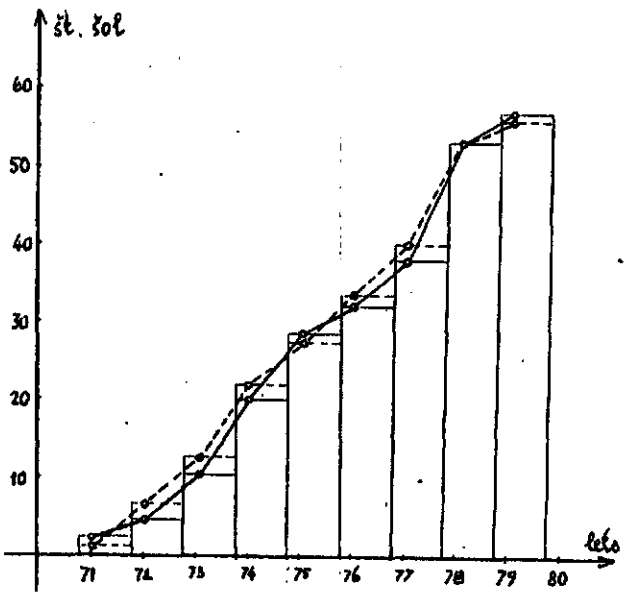
V nadaljevanju si bomo ogledali nekaj izkušenj, ki izhajajo iz pouka predmeta računalništvo v srednjih šolah v zadnjih *desetih letih v SR Sloveniji* [9,10,11]. *Od prvih začetkov* uvajanja predmeta v nekatere srednje šole v šolskem letu 1970/71 je obseg izvajanja stalno naraščal. Najprej so si sledili trije celoletni tečaji za učitelje (največ je bilo učiteljev matematike) v organizaciji Zavoda SR Slovenije za šolstvo, je povzročilo skokovito naraščanje pouka računalništva. Kasneje, v šolskem letu 1977/78, je bil organiziran še en tak tečaj, ki se je od prejšnjih razlikoval v programskem jeziku (PASCAL namesto FORTRANA).

V letih 1976 do 1980 je problematiko pouka računalništva v srednjem izobraževanju v SR Sloveniji spremljala interdisciplinarno sestavljena raziskovalna skupina v okviru projekta "Pouk računalništva v usmerjenem izobraževanju" pri RCPU, FNT.

Rezultat teh prizadevanj so bila predvsem tiskana gradiva: priročnik za učitelje [1], učbenik [2] in zbirka nalog [4]. V šolskem letu 1978/79 sta bili v srednjih šolah s poukom računalništva izvedeni dve anketi. Prva anketa v okviru omenjenega projekta je analizirala predvsem kadrovsko situacijo, druga anketa Zavoda SR Slovenije za šolstvo pa tudi možnosti in želje šol za praktično izvajanje pouka. V šolskem letu 1978/79 je po obeh anketah izvajalo pouk računalništva okrog 60 šol, vključeno pa je bilo približno 5.000 učencev (Sl. 1). Veliko šol je izkoriščalo za praktične vaje obstoječe možnosti na računalniku RCPU - FNT in kapacitete delovnih organizacij. Oprema zadnjih praviloma ni namenjena vzgoji in izobraževanju in marsikje so časovno-prostorski pogoji za poučevanje dokaj neugodni. Iz neenakih pogojev izvirajo težave pri poenotenju in zagotavljanju kvalitete pouka.

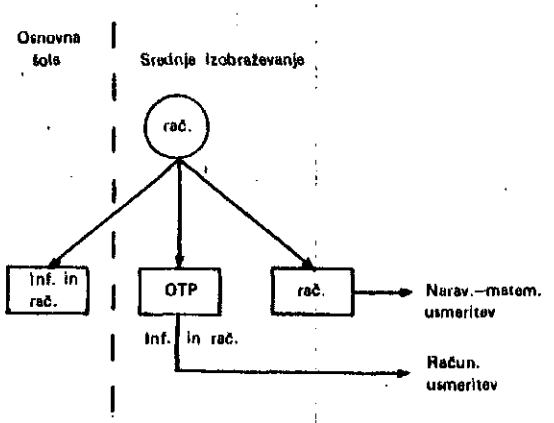
V sodelovanju s projektno skupino je bil na osnovi izkušenj [12] pripravljen učni načrt za računalništvo za štiriletne srednje šole, ki ga je sprejel Strokovni svet za vzgojo in izobraževanje SR Slovenije v juniju 1979 in še velja v sedanjih tehniških šolah in v gimnazijah, ki imajo računalništvo kot praktična znanja. Metoda priprave učnega načrta je bila izrazito oprta na številne učitelje, ki so se prostovoljno vključili v delo raziskovalne skupine. Obsežna analiza učnega načrta je zajela [12]:

- oceno vrednosti tem za doseganje vzgojnoizobraževalnih smotrov,
- čas za obravnavo tem,
- zahtevnost tem za učence,
- zahtevnost tem za učitelja.



Sl. 1. Število šol s predmetom Računalništvo je iz leta v leto naraščalo

Družbene potrebe so ob prehodu na usmerjeno izobraževanje narekovale vključevanje osnovnih znanj o informatiki in računalništvu kot obvezni fond znanja za vsakega učenca v predmet osnove tehnike in proizvodnje v skupni vzgojnoizobrazbeni osnovi. To je seveda vplivalo na nadgradnjo predmeta računalništvo na eni strani in na drugi strani na zasnovanje ustreznega fakultativnega predmeta v osnovni šoli (Sl. 2).



Sl. 2. Z reformo so računalniška znanja postala del Osnov tehnike in proizvodnje (OTP), preoblikoval se je predmet Računalništvo, ki se še ustrezno nadgrajuje v Računalniški in Naravoslovno-matematični usmeritvi, na nivoju osnovne šole pa se pripravlja poseben predmet.

Osnove tehnike in proizvodnje

V okviru osnov tehnike in proizvodnje se vsak srednješolec seznani z osnovnimi pojmi informatike, računalništva in kibernetike [13]. Poseben poudarek je na tipičnih primerih uporabe računalnika na raznih področjih od tehnike, administracije, upravljanja, procesne tehnike pa vse do umetne inteligence z robotiko. Bistvo predmeta je v obravnavi odnosa med človekom in računalniško tehniko. Učenci izvedo odgovore na vprašanja kot so:
- kje in kako si z računalnikom najučinkoviteje pomagamo,
- ali je avtomatizacija vedno na mestu,
- kakšne posledice bo imel hiter razvoj računalniške tehnike,
- kje so vzroki napak v avtomatski obdelavi podatkov,
- kakšna je odvisnost človeka od računalniške tehnike,
- do kakšne mere lahko zaupamo rezultatom, ki jih dobimo iz računalnika,
- kakšna je vloga informatike v procesu samoupravljanja,
- ali računalnik ogroža človeka?

Za predmet osnove tehnike in proizvodnje obstaja učbenik [6]. Poglavje Informatika in računalništvo pa je še posebej skrbno obdelano, saj ga spremlja didaktični komplet [7], ki ga sestavlja preko 30 grafoskopskih prosojnic, 40 diapozitivov in film "Pogled v računalnik".

Predmet računalništvo

V usmerjenem izobraževanju doživlja računalništvo nadaljnji razmah, saj je vključeno v večino programov srednjega izobraževanja na V. stopnji zahtevnosti. Odvisno od usmeritve namenjajo posamezni programi predmetu računalništvo od 35 do 140 ur pouka.

Programsko jedro predmeta obsega poleg uvoda še poglavja Informacija in njena predstavitev, Zgradba in delovanje računalnika, Reševanje problema z računalnikom ter Pregled uporabe računalnikov.

Programsko jedro je zgrajeno s posebnim poudarkom na metodah, ki jih prinašajo, oziroma vzpodbujajo računalniška znanja. Tu velja še posebej omeniti sistematično reševanje problemov in razvijanje algoritmičnih in programskih konceptov. Bistveni del pouka so praktične vaje učencev na računalniku, kjer učenci samostojno razvijajo programe, tj. programirajo v visokem programskem jeziku PASCAL.

Učni načrt za računalništvo z navodili prinaša napotke za uresničevanje vzgojnoizobraževalne vsebine. Poudarjeno je praktično reševanje konkretnih problemov na računalniku, individualizacija in diferenciacija pouka ter skupinsko delo. Tak pouk računalništva seveda poleg drugih didaktičnih pripomočkov zahteva zlasti ustrezno računalniško opremo šol.

Vzpodbudni rezultati računalniškega izobraževanja se kažejo predvsem v velikem zanimanju mladine za lastno delo na računalniku. Ne nazadnje se to odraža tudi na vsakoletnih tekmovanjih srednješolcev iz računalništva, kjer iz leta v leto raste število udeležencev skupaj s kvaliteto njihovih izdelkov.

Današnja mladina bo živela in delala v računalniškem svetu, tako okolje obdaja šolo že danes in računalniški utrip že prodira tudi v osnovno šolo. Ta čas načrtujemo fakultativni predmet informatika in računalništvo v 7. ali 8. razredu osnovnih šol. V osnovi je predmet politehnično zasnovan, njegov namen pa je nuditi osnovnošolcem, ki jih to zanima, ustrezna računalniška znanja in to na organiziran način, ki zagotavlja povezavo s srednješolskim izobraževanjem na tem področju.

Ob tem se zastavlja vprašanje, kako in s čim zajamčiti potrebno kvaliteto izvajanja pouka računalniških predmetov.

6. IZOBRAŽEVANJE UČITELJEV IN RAČUNALNIŠKA OPREMA ŠOL

Začetne kadrovske potrebe za izvajanje pouka predmeta računalništvo so pokrili štirje celoletni tečaji za učitelje pri Zavodu SR Slovenije za šolstvo. Kasneje so se vključili v pouk predmeta računalništvo predvsem mnogo mlajši učitelji matematike, ki so imeli računalništvo s programiranjem v visokošolskem študijskem programu. Vendar to ne zadošča potrebam. V načrtu je program za učitelja računalništva z več računalniškimi znanji.

Vsako leto potekajo na Zavodu SR Slovenije za šolstvo nekajdnevni seminarji, na katerih se učitelji seznanijo z vsebinskimi in metodičnimi novostmi na računalniškem področju.

Seminarsko izobraževanje učiteljev za pouk s pomočjo računalnika že izvaja RCPU pri FNT. Seveda pa bi moral predvsem pouk vsake metodike na pedagoški akademiji in univerzi vsebovati tudi uporabo računalnika pri poučevanju. Vzporedno z razvojem računalništva se mora razvijati tudi izobraževanje učiteljev, ki ne sme zaostajati za družbenimi potrebami.

Računalniška oprema slovenskih srednjih šol je zaenkrat zelo skromna. Le 10 % srednjih šol, ki izvajajo pouk računalništva, ima tudi svoje računalnike in to od preprostih mikroračunalnikov do večuporabniških sistemov. Ostale šole za praktično delo uporabljajo univerzitetne računalnike ali pa računalnike delovnih organizacij v bližnji okolici šole.

Opremljanja srednjih in kmalu tudi osnovnih šol z računalniki ni mogoče odlagati v nedogled. Problem je seveda računalniška oprema domačih proizvajalcev, ki je nekajkrat dražja kot podobna oprema na svetovnem trgu. Že večkrat je bila poudarjena potreba po širši družbeni akciji, s katero bi razbremenili ceno domače računalniške opreme za šolske potrebe. Hkrati pa bi bilo seveda potrebno to opremo standardizirati.

7. RAČUNALNIŠKO USMERJENE ŠOLE

Novost v srednjem izobraževanju je računalniška usmeritev in vzgojnoizobraževalni program računalništvo. V tej usmeritvi se izobražujejo računalniški tehniki za aparaturno opremo (hardware) in programerski tehniki (za software). V tej usmeritvi računalniška znanja predstavljajo osnovo vzgojnoizobraževalnega programa.

Poudarek na računalniških znanjih je tudi v smeri matematični tehnik v naravoslovno-matematični usmeritvi, kjer v izbirnem delu programa naravoslovno-matematična tehnologija v 3. in 4. letniku predmet računalništvo in programiranje usmerja učence v to področje.

8. ZAKLJUČEK

Kljub dokaj celovitemu sistemu izobraževanja na področju računalništva pa tako pri nas, kot tudi drugod po svetu, obstaja vrsta odprtih vprašanj vsebinske, metodične, tehnične in seveda finančne narave. Vendar je širši problem računalnika v šoli predvsem problem miselnosti ljudi na vseh nivojih, od staršev, učiteljev do načrtovalcev širše vzgojnoizobraževalne politike. Gre za resnično razumevanje mesta in vloge računalnika v šoli in za posledice širše kompjuterizacije človekovih aktivnosti. Očitno je namreč to, da že danes tudi v šoli brez računalnika zaostajamo za tistimi, ki ga imajo, jutri pa verjetno brez računalnika ne bomo mogli več naprej.

LITERATURA

1. Roblek, B. (urednik), Računalništvo: Gradivo s tečaja za učitelje, Zavod SRS za šolstvo, Ljubljana, 1973
2. Bratko, I., Rajkovič, V., Uvod v računalništvo, DZS, Ljubljana, 1975
3. Polya, G., Kako rešujemo probleme, DDU Univerzum, Ljubljana, 1976
4. Benkovič, J., Cokan, A., Martinec, M., Reinhardt, R., Roblek, B., Računalništvo - Zbirka nalog 1, DZS, Ljubljana, 1980
5. Wirth, N., Računalniško programiranje, DMFA, Ljubljana, 1980
6. Bratko, I., Rajkovič, V., Informatika in računalništvo - poglavje v učbeniku Osnove tehnike in proizvodnje, DZS, Ljubljana, 1981
7. Bratko, I., Rajkovič, V., Informatika in računalništvo - didaktični komplet (prosojnice, diapozitivi, film), DDU Univerzum, Ljubljana, 1981
8. Mohar, B., Zakrejšek, E., Uvod v programiranje, DMFA, Ljubljana, 1982
9. Bratko, I., Rajkovič, V., Roblek, B., What should secondary school students know about computers. Analysis of an experiment, IFIP 2nd World Conference on Computers in Education, Marseilles, 1975
10. Benkovič, J., Bratko, I., Kornhauser, A., Rajkovič, V., Roblek, B., Sirnik, I., Introducing fundamentals of cybernetics with automatic control at secondary school level, IFAC Symposium Trends in Automatic Control Education, Barcelona, 1977
11. Rajkovič, V., On the role of computer science subjects at the secondary school level, 3rd International Conference on Information Technology, Jerusalem, 1978
12. Benkovič, J., Kornhauser, A., Rajkovič, V., Primerjalna analiza pouka računalništva na srednji stopnji izobraževanja, Informatika, Vol. 4, No. 4, Ljubljana, 1980, pp. 25-33
13. Bratko, I., Rajkovič, V., Informatika in računalništvo v osnovah tehnike in proizvodnje, vzgoja in izobraževanje, Vol. 12, No. 5, Ljubljana, 1981, pp. 49-50
14. Cokan, A., Tretja svetovna konferenca o računalnikih v izobraževanju, Vzgoja in izobraževanje, Vol. 12, No. 5, Ljubljana, 1981
15. Cokan, A., Računalniško izobraževanje v Angliji, Vzgoja in izobraževanje, Vol. 13, No. 4, Ljubljana, 1982, pp. 54-56
16. Golden, F., Computer generation: A new breed of Whire kinds, Time, maj 3, 1982

INFORMATICA 4/1983

ARHITEKTURA MIKRORAČUNARSKIH MODULA NAMIJENJENIH ZA UPRAVLJANJE PROCESIMA

MARIO ŽAGAR

ZAVOD ZA REGULACIONU I SIGNALNU TEHNIKU, ELEKTROTEHNIČKI FAKULTET, ZAGREB

UDK: 681.519.7

U članku su razmatrani principi izgradnje mikroračunarskih modula za različite primjene u upravljanju procesima. Opisana je jedna ideja realizacije mikroračunarske konfiguracije pomoću standardnih modula. Pri tome su zadovoljeni uvjeti da se iz istih modula može realizirati sistem čije se karakteristike mijenjaju od minimalnih do maksimalnih. Prikazano je jedno konkretno rješenje.

ARCHITECTURE OF MICROCOMPUTER MODULES FOR PROCESS CONTROL
In this article the principles of building microcomputer modules for different process control purposes are discussed. One idea is described, realizing microcomputer configuration with standardized modules. Conditions for realizing microcomputer system from the minimal to the maximal characteristics with the same modules are satisfied. One practical solution is given.

UVOD

Moderni pravci razvoja na području projektiranja i izgradnje digitalnih računarskih sistema namijenjenih za direktno digitalno upravljanje različitim procesima uvjetovani su:

- revolucionarnim razvojem osnovnih elektroničkih, mikroprocesorskih komponenata kao opreme čiju snagu treba u vrlo kratkom vremenu, od pojave nove kompo - nente i njezinog osvajanja tržišta do razvoja složenih uređaja, optimalno iskoristiti i omogućiti joj da se dokaže,
- tradicionalnom konzervativnošću i nepovjerenjem u sve što je novo, nedovoljno provjereno i prihvaćeno što je često i opravdano,
- osnovnim zahtjevom da se svi novi ele - menti uklope u već postojeće stanje jer svaka izmjena zahtijeva dosta truda i financijskih izdataka,
- minimalnim korisničkim zahtjevima na početku projekta koji se s vremenom sve više povećavaju.

Navedeni uvjeti su činjenice koje treba prihvatiti i na njima bazirati nova rješenja.

Pojavom relativno jeftinih i pametnih mikroračunarskih komponenata kao logičkih cjelina iz kojih gradimo određenu mikroračunarsku konfiguraciju po vlastitoj volji, nameće se potreba za definiranjem i realizacijom takve konfiguracije koja će koncepcijom biti primjenjiva za različite tipove i načine povezivanja s procesima, a isto tako i međusobnog povezivanja s drugim mikroračunalima u cilju dobivanja sistema za raspodijeljeno procesiranje. Osnovni kriterij kojeg se pri tome treba pridržavati je modularnost i standardiziranost određenih mikroračunarskih cjelina.

ANALIZA MIKRORAČUNARSKIH MODULA

Ako pokušamo analizirati osnovne principe izgradnje modula mikroračunala, isključivši jednočipna mikroračunala pogodna za jednostavnije primjene, ali manje pogodna za složenije, možemo istaći dva osnovna pristupa:

a) na pojedinim fizičkim modulima, pločicama, realizirani su odgovarajući funkcionalni elementi. Tako npr. možemo projektirati CPU pločicu (procesor, napajanje, takt, inicijalizacija, pojačala i dr.), memorijsku PROM pločicu (programabilna memorija koja trajno pamti podatke), ulazno/izlaznu pločicu za komunikaciju s vanjskim svijetom i dr. Prednost takvog pristupa je neovisnost modula. Svaki modul mora zadovoljavati uvjete koje mu propisuje sabirnica preko koje su svi moduli povezani (Sl. 1).

Krenemo li tim putem, nećemo pogriješiti. Veliki broj svjetskih proizvođača mikroračunarske opreme ima takav pristup u izgradnji svojih sistema. Osnovni nedostatak tog pristupa (u nekim slučajevima to je i prednost) je raspodijeljenost funkcija na fizički odvojenim modulima. Ne postoji bitna kvalitativna razlika u modulima kod minimalne i maksimalne konfiguracije. Razlika je jedino u broju pojedinih modula. Za svaki modul poznate su njegove karakteristike npr. RAM modul 4Kx8, EPROM modul 8Kx8, U/I modul s 16 linija, itd. U ovom pristupu, određenu mikroračunarsku konfiguraciju dobijemo odabiranjem broja pojedinih modula koji su nam potrebni. Tako za rješenje određenog problema možemo imati: 1 x CPU modul, 4 x RAM modul, 2 x EPROM modul, 2 x U/I modul, čime smo dobili konfiguraciju od CPU, 16Kx8 RAM memorije, 16Kx8 EPROM memorije, 32 U/I linije.

b) Pristup u kojem se mikroračunarski elementi raspoređuju na pojedine module prema tome koliko se često pojavljuju kod određene razine složenosti problema. Tako je npr. za raspodijeljeno upravljanje procesima u realnom vremenu, već i kod najminimalnije konfiguracije, potrebno imati elemente za generiranje realnog vremena te komunikaciju s okolišem. Raspored elemenata i modula razlikovat će se od slučaja "a" , iako logičke funkcije možemo promatrati na isti način kao i prije dakle kao zasebne module. Fizički su oni ipak drugačije raspoređeni. Smanjuje se broj modula, a povećava raspon njihovih mogućnosti u cilju jednostavnijeg rješavanja različitih razina složenosti problema, a uz manje financijske izdatke.

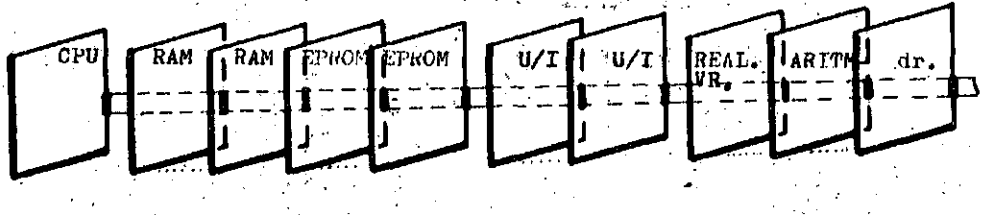
Pokušamo li propisati što takvi osnovni moduli moraju sadržavati, dolazimo u (ne)prilici da definiramo nešto s čime se svi sigurno neće složiti. To je u svakom slučaju jedan kompromis između postavljenih zadataka koje modul treba znati riješiti, veličine i složenosti modula, njegove cijene, univerzalnosti i dr.

U daljnjem tekstu bit će dan jedan od mogućih prijedloga koji zadovoljava veliku većinu slučajeva.

REALIZACIJA MODULA MIKRORAČUNALA

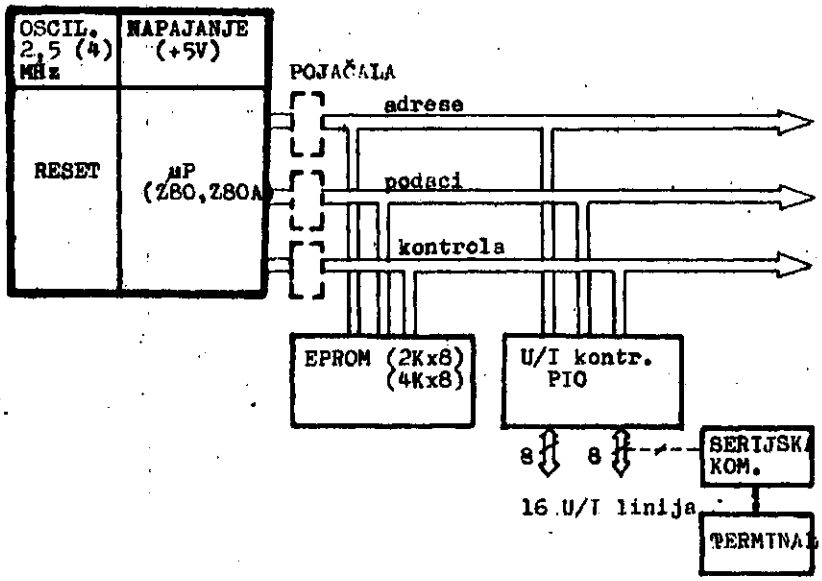
I. minimalna konfiguracija - "jezgra osnovnog modula" realizirana na dijelu štampane pločice

Svojstva "jezgre osnovnog modula" (sl. 2):
- mikroprocesor (Z8o, Z8oA)
- RESET logika
- napajanje (+5 V)
- oscilator (2,5 MHz, 4 MHz)



Slika 1

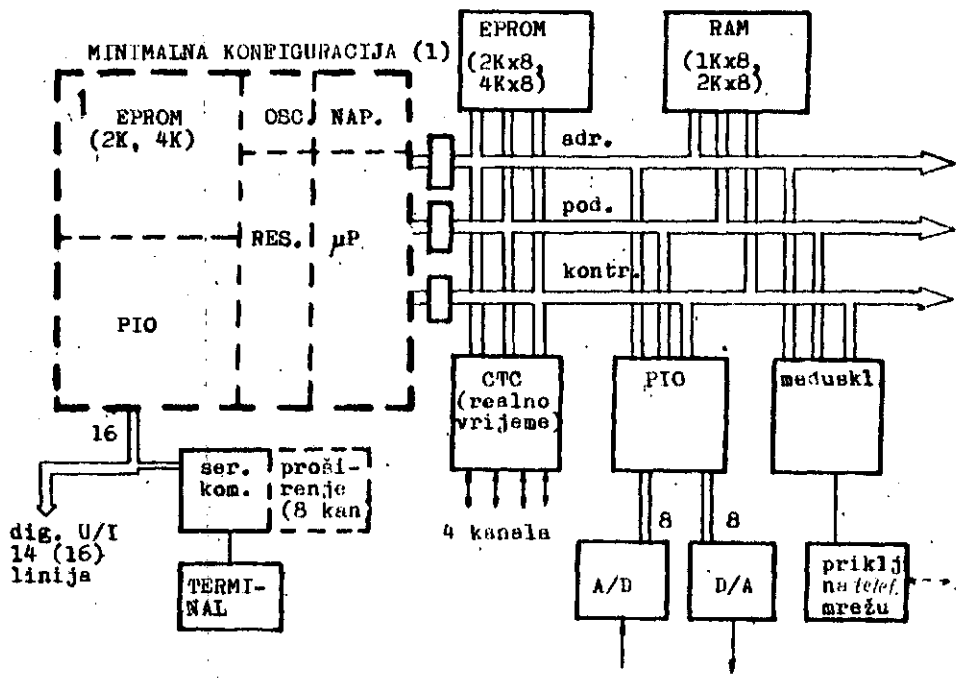
Funkcionalni moduli povezani preko sabirnice



Slika 2 - Jezgra osnovnog modula

- indikacija stanja mikroprocesora
- sklopovi za pojačavanje signala
- sklopovi za povezivanje s drugim procesorima
- lokalna memorija tipa EPROM (Erasable Programmable Read Only Memory) 2Kx8,4Kx8
- PIO (Parallel Input/Output) 16 U/I linija plus 2 linije za serijsku komunikaciju koja se rješava programski
- komunikacija preko terminala ili kalkulatorske tastature i pokaznika
- upravljački program - monitor i/ili programska podrška za konkretni zadatak.

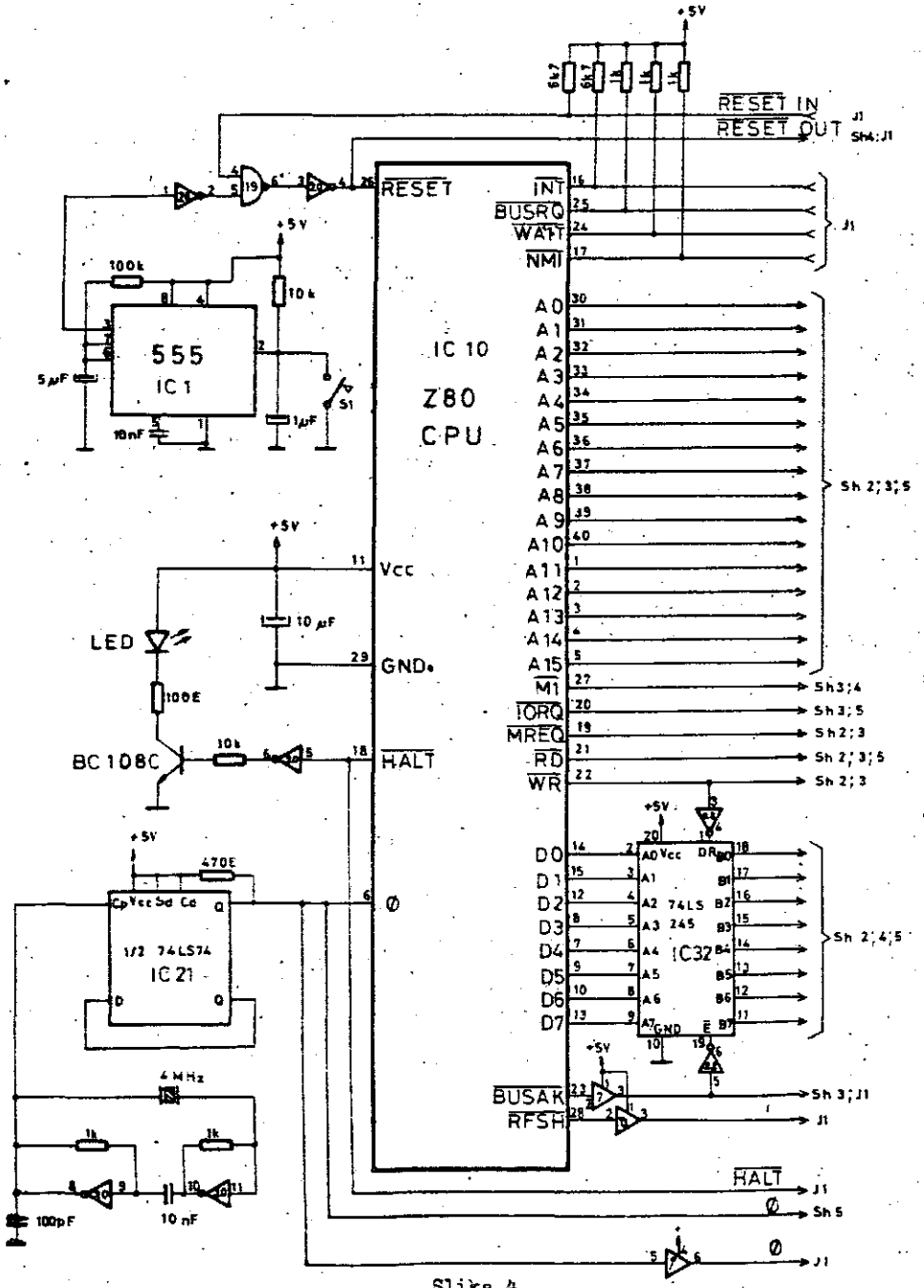
Dodatnim proširivanjem osnovne jezgre, približavamo se osnovnom modulu prikazanom na slici 3. Moguća je postepena nadogradnja komponenata i programa prema trenutnim potrebama od "jezgre osnovnog modula" prema "osnovnom modulu".



Slika 3

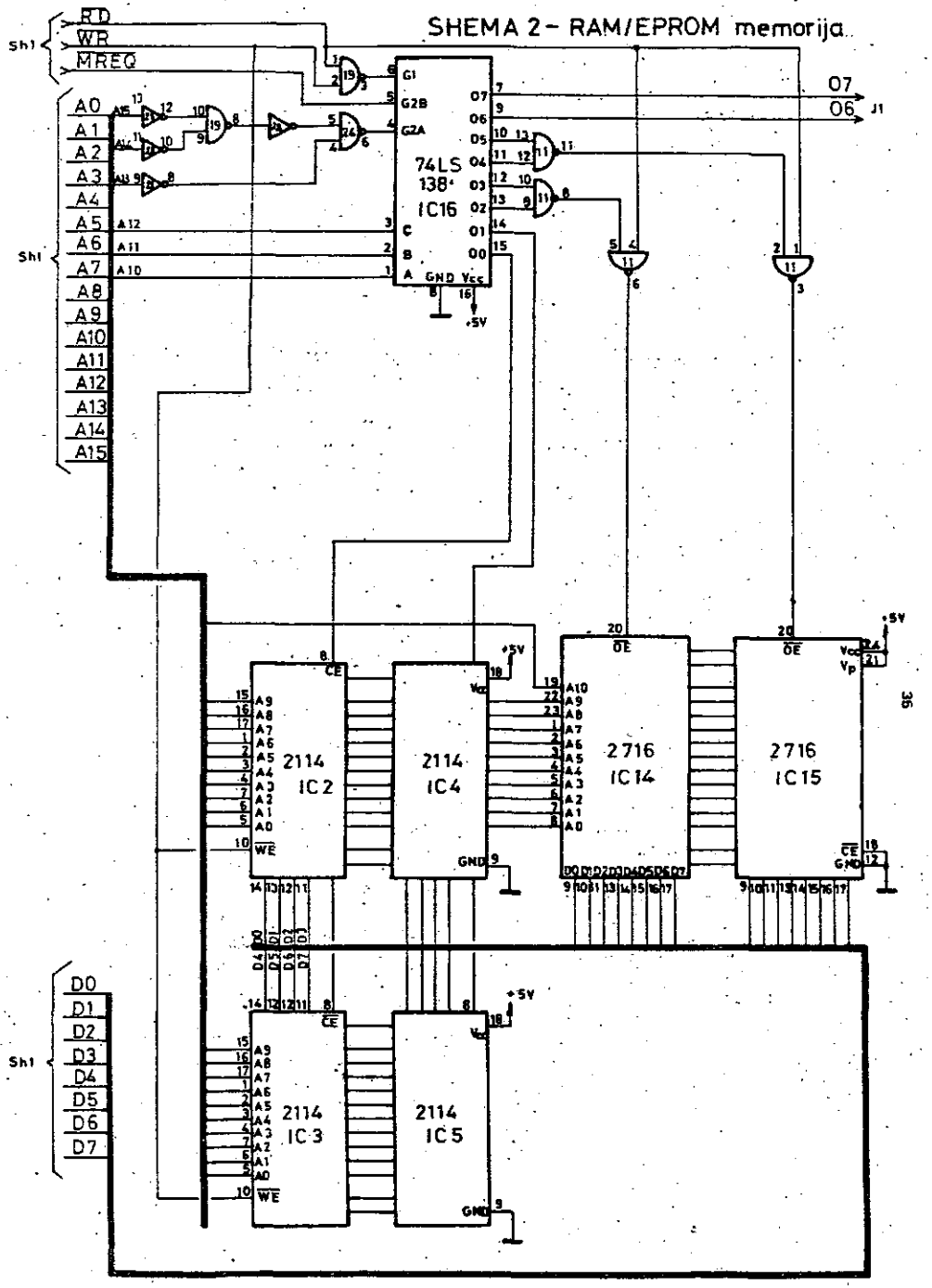
Osnovni modul

SHEMA 1– CPU



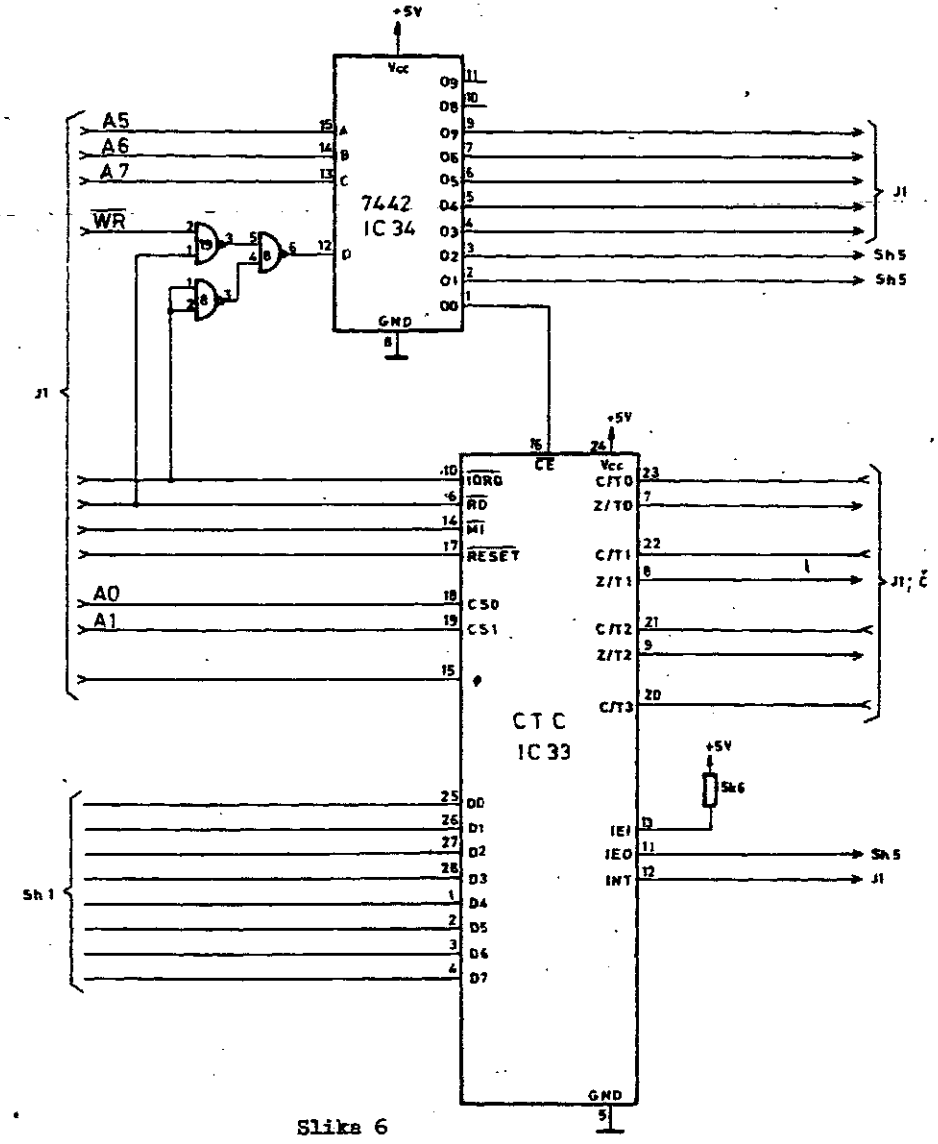
Slika 4

SHEMA 2– RAM/EPROM memorija



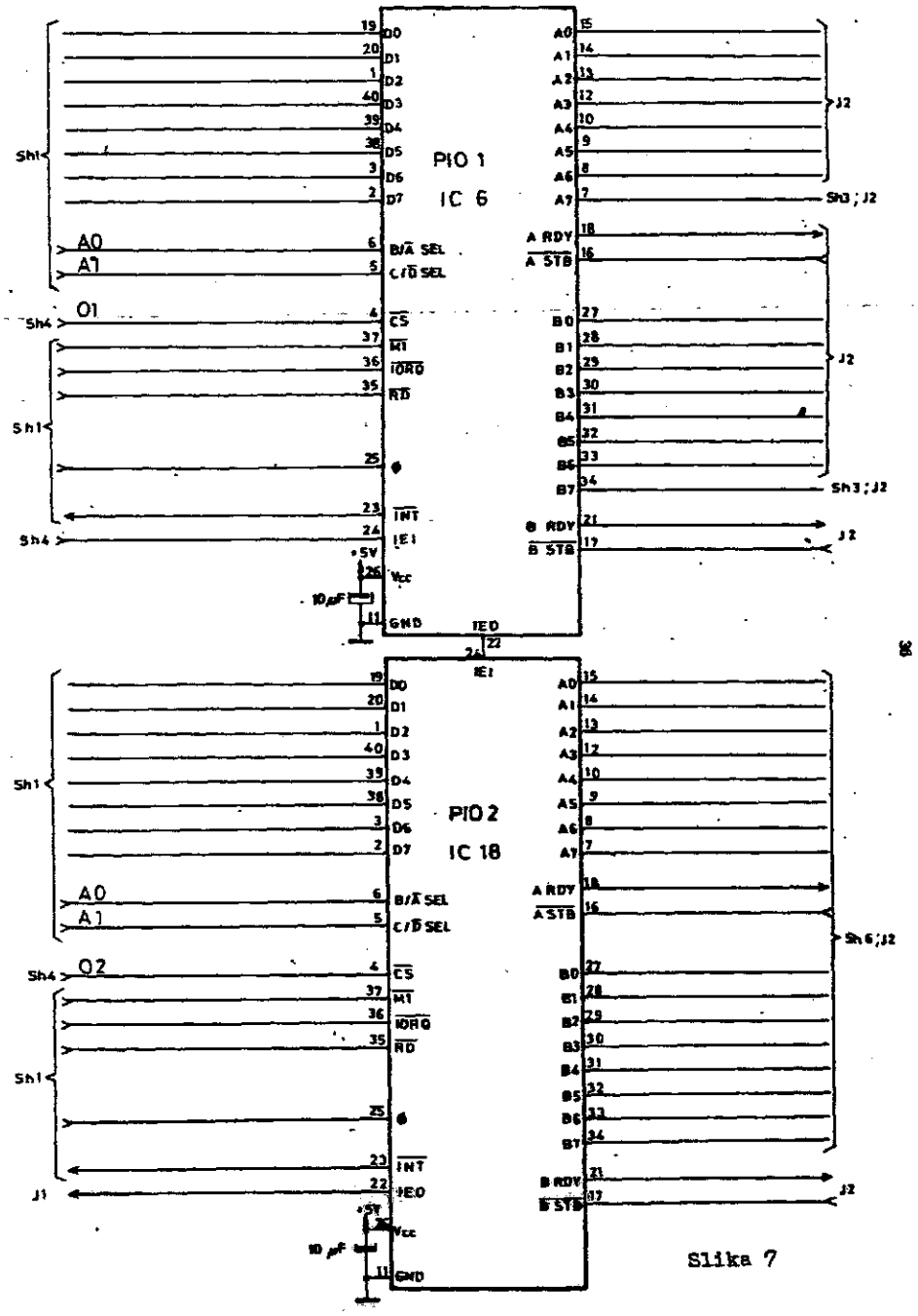
Slika 5

SHEMA 4 – CTC : I/O selektor



Slika 6

SHEMA 5 – PIO1, PIO2



Slika 7

II. Osnovna konfiguracija - "osnovni modul" realiziran na jednoj pločici dvostrukog EVRO formata

Osnovni modul sadržava "jezgru osnovnog modula" plus:

- CTC (brojilo/mjerilo vremenskih impulsa s 4 kanala za rad u realnom vremenu)
- PIO (16 U/I digitalnih linija, što zajedno s PIO jedinicom iz "jezgre osnovnog modula" čini 3o U/I linija plus serijski kanal riješen programski ili 16 (14) U/I linija plus analogni ulaz plus analogni izlaz
- lokalnu memoriju tipa RAM (2Kx8)
- mogućnost priključivanja standardnih U/I jedinica (terminala, tastature, pokaznika, štampača, kazetofona) i nestandardnih (A/D,D/A, telefonske mreže i dr.)
- jezgra operacionog sistema plus različiti dodatni programi za obradu u realnom vremenu, matematičke operacije, komunikaciju s drugim mikroračunalima
- memorijski prostor za spremanje specijalne programske podrške i podataka za rješavanje konkretnog zadatka.

Najbitniji elementi osnovnog modula prikazani su na slikama 4,5,6,7. Taj je modul dovoljan za rješavanje niza jednostavnijih funkcija upravljanja i može djelovati samostalno. Ako želimo proširiti postojeću konfiguraciju, jedan od načina je da definiramo dodatne module mikroračunala čijim priključivanjem će osnovna jezgra sistema postajati sve snažnija. Tu i dalje zadržavamo princip da moduli za proširivanje sadrže različite komponente mikroračunala.

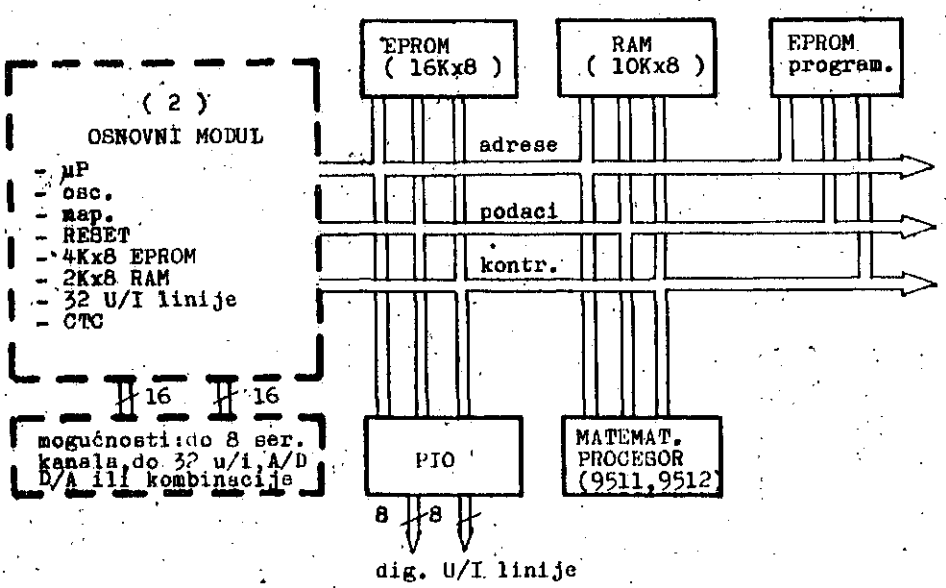
Praktički se sa 3 modula, kombinacijom i popunjavanjem tih modula sa programabilnim komponentama i inicijalizacijskim programiranjem komponenata može pokriti vrlo široko područje primjene, kako po složenosti tako i po tipu posla.

III. Mikroračunarska konfiguracija realizirana iz dva modula ("osnovni modul" plus "prvi dodatni modul")

Konfiguracija je realizirana na dvije pločice dvostrukog EVRO formata. Prvi dodatni modul prikazan je na slici 8.

Svojstva prvog dodatnog modula:

- 16K x 8 EPROM memorije (2716/2732)
- 1oK x 8 RAM statičke memorije
- EPROM programator za I2716/2732
- PIO jedinica sa 16 U/I linija
- matematički procesor (9511/9512).



Slika 8

Prvi dodatni modul (osnovni modul prikazan je crtkano)

Osnovni modul i prvi dodatni modul zajedno imaju velike sklopovske mogućnosti koje se iskorištavaju upisivanjem odgovarajućih programa za određene primjene u EPROM memoriju. Osnovne karakteristike cijele konfiguracije su : 32Kx8 memorije (EPROM + RAM) koja se može prilagođavati trenutnim potrebama (koraci po 2Kx8), brojilo/mjerilo vremenskih impulsa, matematički procesor (algebarske operacije i različite matematičke funkcije u nepomičnom i pomičnom zarezu te normalna i povećana preciznost), za potrebe gdje je programski matematički paket previše spor, EPROM programator koji može programirati EPROM memorije 2716/2732 bez dodatne intervencije korisnika (može se koristiti kao memorija za trajno pamćenje podataka).

Uz odgovarajuću programsku podršku (monitor, asembler, editor, linker i dr.) koja je smještena u EPROM memoriji, ova konfiguracija postaje sistem za razvoj drugih mikroračunarskih modula ili (uz promjenu programa) sistem konkretne namjene. Preko vanjskih jedinica te uz pomoć programske podrške ostvaruje se veza sa drugim mikroračunalima, tvoreći tako bazu za razvoj raspodijeljenih mikroračunarskih sistema za upravljanje procesima.

IV. Mikroračunarska konfiguracija realizirana s 3 modula ("osnovni modul" plus "prvi dodatni modul" plus "drugi dodatni modul" realizirani na 3 pločice dvostrukog EVRO formata)

Drugi dodatni modul prikazan je na sl.9. Svojstva drugog dodatnog modula:

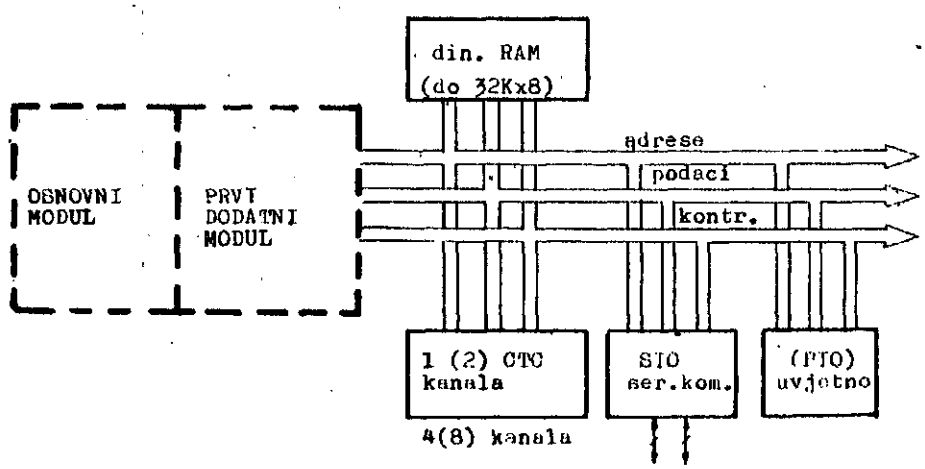
- 32K x 8 dinamičke RAM memorije (s promijenjivim adresnim područjem)
- 1 (2) CTC jedinice s 4 (8) kanala
- SIO jedinica s 2 serijska kanala (programsko rješavanje različitih protokola)
- PIO jedinica sa 16 U/I linija.

Uz osnovni modul mogu se koristiti dodatni moduli u bilo kojoj konfiguraciji (osnovni plus prvi, osnovni plus drugi, osnovni plus prvi plus drugi). Svojstva mikroračunarske konfiguracije sa sva 3 modula spadaju u gornju klasu svojstava današnjih 8-bitnih mikroračunala (Slika 10).

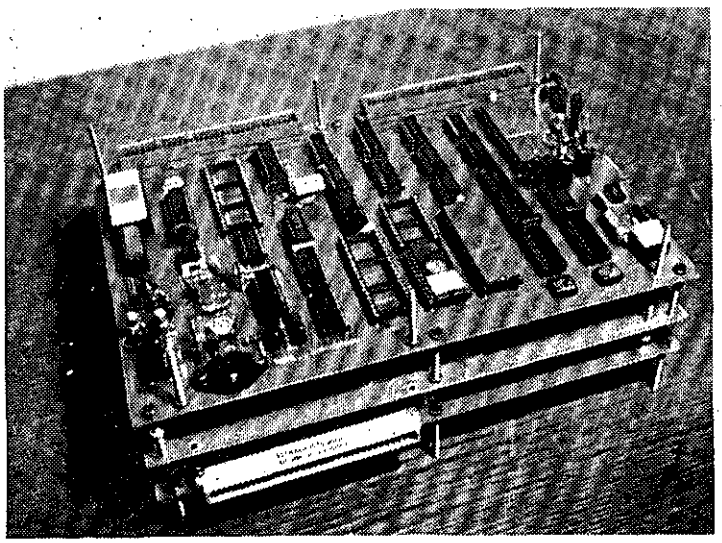
Osnovna svojstva mikroračunarske konfiguracije s 3 modula su:

- mikroprocesor Z80 (Z80A)
- 64Kx8 memorija (različite mogućnosti kombiniranja RAM/EPROM)
- 3x CTC (12 kanala za realno vrijeme)
- SIO (2 kanala serijske komunikacije (plus 8 kanala preko PIO jedinice)
- 3 (4) PIO (48(64) U/I linije)
- matematički procesor (9511/9512)
- A/D, D/A pretvorba
- EPROM programator (I2716/2732)
- priklj. kazetne jedinice
- priklj. za linijski štampač
- kalkulatorska tastatura i 7-segm.pokaznik.

S ovom konfiguracijom rješavaju se vrlo složeni problemi upravljanja. Memorijski prostor dozvoljava korištenje viših programskih jezika u kombinaciji s asemblerom na vremenski kritičnim odsječcima.



Slika 9
Drugi dodatni modul
(osnovni modul i prvi dodatni modul prikazani su crtkano)



Slika 10
Mikroračunarska konfiguracija s 3 modula

V. Dodatni moduli

Uz prethodno nabrojene module, priključuje se uz bilo koju konfiguraciju posebni modul čija je funkcija prilagođavanje veličina iz vanjskog svijeta u oblik pogodan za unos u mikroračunalo i obratno. Taj je modul realiziran na jednoj pločici dvostrukog EVRO formata, a komponente (pojačala, sklopke, elementi za galvansko odvajanje, LED diode, triaci i drugi energetski dijelovi) se popunjavaju prema primjeni za koju je mikroračunarska konfiguracija određena.

S opisanim modulima ova mikroračunarska konfiguracija nije zatvorena. Ona se prema potrebama može proširivati i sa dodatnom memorijom (npr. 512Kx8 lokacija) i sa U/I komponentama (maksimalno 256).

Bitna je karakteristika ovog pristupa da se i za najjednostavnije i najsloženije konfiguracije koriste iste komponente i moduli što zadovoljava niz primjena od rješavanja jedne jednostavne funkcije do raspodijeljenih sistema za rješavanje vrlo složenih zadataka.

ZAKLJUČAK

Osim kvalitete mikroprocesora koji se ugrađuje u određeni uređaj, sve se više traži i kvaliteta cijelog uređaja, a to znači dobra povezanost i raspodijeljenost uloga između mikroprocesora i dodatnih jedinica.

Važnu ulogu pri tome imaju programabilne jedinice koje omogućavaju projektiranje modularnih sistema čija konfiguracija može rasti od minimalne do maksimalne uz mogućnost projektiranja "po mjeri" i naknadnog proširenja konfiguracije.

Analiziran je jedan mogući pristup raspodijeli komponenata i modula mikroračunala namijenjenog upravljanju procesima tj. primjeni u realnom vremenu.

Specifičnost pristupa je prvenstveno u rasporedu različitih mikroprocesorskih komponenata u okviru pojedinih modula te njihovom prilagođavanju za široki raspon primjene u upravljanju proizvodnim procesima.

Mikroračunarske konfiguracije realizirane različitim kombinacijama istih modula i komponenata koriste se na različitim razinama složenosti zadatka, od rješavanja jednog problema (npr. pametnog mjernog pretvornika), lokalnih stanica za rješavanje određene grupe problema i povezivanja s drugim udaljenim stanicama do globalnog sistema koji vrši nadzor i generalne upravljačke funkcije nad svim podređenim stanicama.

Isti princip sadržan je i u razvoju programske opreme koja je također raspoređena po pojedinim modulima pa tako imamo osnovnu programsku podršku na "osnovnom modulu" koja se sastoji iz monitora i dodatnih programa za: simulaciju 8 serijskih kanala preko PIO jedinice za paralelnu komunikaciju, obradu realnog vremena, komunikaciju s kalkulatorskom tastaturom i 7-seg.

pokaznikom, 16-bitnu cjelobrojnu aritmetiku, upravljanje EPROM programatorom kao standardnim memorijskim područjem i dr.

Na dodatnim modulima smješteni su (prema potrebi) asembler, editor, linker ako se sistem koristi kao razvojni te aritmetika u pomičnom zarezu, pretvorba i formatiranje podataka za rad matematičkog procesora, A/D pretvorba, D/A pretvorba, različiti algoritmi upravljanja, prikaz realnog vremena na ekranu terminala, globalni prikaz stanja procesa, prikaz pojedinih parametara, povijesni prikaz događaja u procesu i dr.

Daljnji razvoj je ugradnja i povezivanje viših programskih jezika i asemblera u cilju poboljšanja karakteristika postojećeg sistema.

LITERATURA

1. J.M.Ayache, J.P.Courtiat,M.Diaz: "REBUS, A Fault-Tollerant Distributed System for Industrial Real-Time Control" IEEE Transactions on Computers, Vol . C-31,No.7, July 1982,(637-647)

2. - : Memory Data Book and Designers Guide, Mostek, 1979.

3. - : Microcomputer Components Data Book, Mostek, 1979.

4. - : Data Catalog 1981, Intel Corp., 1981.

5. - : Z80-CPU,PIO,CTC,SIO Technical Manuals, Zilog, 1978.

6. B.Blatnik i ostali: "Mikroračunarski sistem DELTA 323/M" Informatica 1/1982. (4-22)

7. - : AmZ8000 Family Data Book, Advanced MicroDevices, 1980.

8. M.Žagar: "Primjena mikroprocesorskih komponenata Z80 u proizvodnji računarske opreme", Informatica 3/1978.(33-42)

9. M.Žagar: "Primjena EPROM memorija u prikupljanju i obradi podataka dobivenih iz procesa" Informatica 1/1980. (42-46)

ALGOL 60 ZA SISTEM CP/M I

A. P. ŽELEZNIKAR

ISKRA DELTA, LJUBLJANA

UDK: 681.06 ALGOL60:519.682

ČLANEK OPISUJE JEZIK ALGOL 60 ZA OPERACIJSKI SISTEM CP/M. TA JEZIK JE ZANIMIV ZARADI VELIKE KOLIČINE OBSTOJEČIH ALGOLSKIH PROGRAMOV PA TUDI ZARADI MOŽNOSTI POUČEVANJA PROGRAMIRANJA V VISOKIH PROGRAMIRNIH JEZIKIH - STRUKTURIRANEGA IN NESTRUKTURIRANEGA. TA ČLANEK PRINAŠA V SVOJEM PRVEM DELU OPIS OMEJITEV IN RAZŠIRITEV JEZIKA RML ALGOL GLEDE NA ALGOL 60 IN KRATEK PRIROČNIK JEZIKA RML ALGOL. V PRIROČNIKU SO OPISANE KLJUČNE BESEDE IN IMENA, PREDDEKLARIRANA IMENA, ZGRADBA ALGOLSKEGA PROGRAMA, BLOKI IN DEKLARACIJE, PROGRAMSKA OBLIKA, PODATKOVNI TIPI, IMENA, SIMBOLI, POLJA, ENOSTAVNI IZRAZI, NIZI IN ZNAKOVNI LITERALI, PRIREDITVENI STAVKI, POGOJNI IZRAZI, POGOJNI STAVKI, FOR STAVKI, PRAZNI STAVKI, KOMENTARJI, OZNAČITVE, RAZDELILNIKI IN GOTO STAVKI, OZNAČITVENI IZRAZI, PROCEDURE, PROCEDURE S PARAMETRI, ŠTEVILSKI IN BOOLOVSKI PARAMETRI PRI KLICU Z VREDNOSTJO, KLIC SPREMENLJIVKE Z IMENOM (Z NAVEDBO), NIZNI IN RAZDELILNIŠKI PROCEDURNI PARAMETRI, OZNAČITVE IN PROCEDURE KOT PARAMETRI TER POVZETEK PROCEDURNIH ZNAČILNOSTI. V DRUGEM DELU BODO OPISANE ŠE VHODNE/IZHODNE PROCEDURE IN DRUGE ZNAČILNOSTI JEZIKA RML ALGOL. NA KONCU ČLANKA JE DODANA SINTAKSA JEZIKA RML ALGOL, NJEGOVE VHODNE/IZHODNE PROCEDURE IN NEKATERI PROGRAMIRNI PRIMERI.

A CP/M System Algol 60 Language I

THIS ARTICLE DEALS WITH AN ALGOL 60 LANGUAGE FOR CP/M OPERATING SYSTEM. ALGOL 60 IS BEING RELEVANT BECAUSE OF A SUBSTANTIAL QUANTITY OF EXISTING ALGOL PROGRAMS AND BECAUSE OF THE POSSIBILITY TO TEACH PROGRAMMING - STRUCTURED AND UNSTRUCTURED ONE - IN HIGH LEVEL PROGRAMMING LANGUAGES. IN THE FIRST PART, THIS ARTICLE DESCRIBES LIMITATIONS AND EXTENSIONS OF THE RML ALGOL CONSIDERING ALGOL 60 STANDARD AND GIVES A SHORT GUIDE HOW TO USE RML ALGOL. THE FOLLOWING IS PRESENTED: KEYWORDS AND IDENTIFIERS, PRE-DECLARED IDENTIFIERS, THE STRUCTURE OF AN ALGOL PROGRAM, BLOCKS AND DECLARATIONS, PROGRAM LAYOUT AND STYLE, DATA TYPES, IDENTIFIERS AND SYMBOLS, ARRAYS, SIMPLE EXPRESSIONS, STRINGS AND CHARACTER LITERALS, ASSIGNMENT STATEMENTS, CONDITIONAL EXPRESSIONS, CONDITIONAL STATEMENTS, FOR STATEMENTS, DUMMY STATEMENTS, COMMENTS, LABELS, SWITCHES AND GOTO STATEMENTS, DESIGNATIONAL EXPRESSIONS, PROCEDURES, PROCEDURES WITH PARAMETERS, NUMERICAL AND BOOLEAN PARAMETERS BY VALUE, VARIABLES CALLED BY NAME (REFERENCE), STRING AND SWITCH PROCEDURE PARAMETERS, LABELS AND PROCEDURES AS PARAMETERS, AND A SUMMARY OF POINTS ON PROCEDURES. IN THE SECOND PART OF THE ARTICLE THE INPUT/OUTPUT PROCEDURES OF RML ALGOL AND OTHER CHARACTERISTICS WILL BE DESCRIBED. FINALLY, SUPPLEMENTS FOR RML ALGOL SYNTAX, ITS INPUT/OUTPUT PROCEDURES AND SOME PROGRAMMING EXAMPLES ARE LISTED.

1. Uvod

Jezik Algol 60 je nastajal v 60-ih letih v posebnem odboru IFIP ((1)); tudi opisana različica Algola (sintaksa) bo izhajala iz poročila ((1)), temeljila pa bo tudi na priročniku jezika Algol 60 za sistem CP/M s procesorjem Z80 podjetja Research Machines, P.O.Box 75, Chapel Street, Oxford, England (0865) 49792 ((6)); ta jezik bomo imenovali RML Algol. Razlika med jezikoma Algol 60 in RML Algol bo minimalna in slednji bo tudi razširitev jezika Algol 60 z natančno opredelitvijo V/I procedur.

V 60-ih letih je časopis Communications of the ACM objavljal najrazličnejše algoritme, zapisane v obliki algolskih procedur, ki so pokrivali široko področje matematičnih in tehniških funkcij. Ti algoritmi so bili vselej utemeljeni z določenimi matematičnimi raziskavami in v veliki meri optimizirani. Povzetki in izboljšave teh algolskih algoritmov so bili objavljeni v posebnih publikacijah ((2, 3, 4)); več teh algoritmov bomo prikazali v jeziku RML Algol in jih preizkusili; objavljali jih bomo tudi v rubriki Uporabni programi časopisa Informatica.

```
BEGIN INTEGER in,out,c;

COMMENT this program converts from
the upper/lower case convention to
the upper case in quotes convention;

PROCEDURE seto(a);
VALUE a; INTEGER a; ioc(31);

INTEGER PROCEDURE swlist;
ioc(39);

PROCEDURE getc;
BEGIN c:=chin(in);
      IF c<0 OR c=&^Z THEN GOTO fin
END ;


PROCEDURE uout;
IF c>=97 AND c<=122 THEN chout(out,c-32)
ELSE chout(out,c);


      seto(swlist+13);
      text(10,"ALG"); {default extension}
a1:   ioc(2); in:=input;
      IF in<1 THEN GOTO a1;
      out:=output;
      IF out<1 THEN
      BEGIN close(in); GOTO a1;
      END ;

loop: getc;
      IF c=&" THEN
      BEGIN uout;
a2:           getc; uout;
              IF c=&** THEN
              BEGIN getc; uout;
              END ELSE
              IF c=&" THEN GOTO loop;
              GOTO a2;
      END ELSE
      IF c=&*C THEN
      BEGIN uout; GOTO loop END ELSE
      IF c=&& THEN
      BEGIN uout; getc; uout;
              IF c=&** OR c=&*^ THEN
              BEGIN getc; uout
              END ;
              GOTO loop;
      END ELSE
      IF c>=&A AND c<=&Z THEN
      BEGIN chout(out,&'); uout;
a3:           getc;
              IF c>=&A AND c<=&Z THEN
              BEGIN uout; GOTO a3
              END ;
              chout(out,&'); uout;
              GOTO loop;
      END ELSE uout;
      GOTO loop;

fin:  chout(out,&^Z); close(out);
      close(in); GOTO a1
END
FINISH
```

```
'BEGIN' 'INTEGER' IN,OUT,C;

'COMMENT' THIS PROGRAM CONVERTS FROM
THE UPPER/LOWER CASE CONVENTION TO
THE UPPER CASE IN QUOTES CONVENTION;

'PROCEDURE' SETO(A);
'VALUE' A; 'INTEGER' A; IOC(31);

'INTEGER' 'PROCEDURE' SWLIST;
IOC(39);

'PROCEDURE' GETC;
'BEGIN' C:=CHIN(IN);
      'IF' C<0 'OR' C=&^Z 'THEN' 'GOTO' FIN
'END' ;


'PROCEDURE' UOUT;
'IF' C>=97 'AND' C<=122 'THEN' CHOUT(OUT,C-32)
'ELSE' CHOUT(OUT,C);


      SETO(SWLIST+13);
      TEXT(10,"ALG"); {DEFAULT EXTENSION}
A1:   IOC(2); IN:=INPUT;
      'IF' IN<1 'THEN' 'GOTO' A1;
      OUT:=OUTPUT;
      'IF' OUT<1 'THEN'
      'BEGIN' CLOSE(IN); 'GOTO' A1;
      'END' ;

LOOP: GETC;
      'IF' C=&" 'THEN'
      'BEGIN' UOUT;
A2:           GETC; UOUT;
              'IF' C=&** 'THEN'
              'BEGIN' GETC; UOUT;
              'END' 'ELSE'
              'IF' C=&" 'THEN' 'GOTO' LOOP;
              'GOTO' A2;
      'END' 'ELSE'
      'IF' C=&*C 'THEN'
      'BEGIN' UOUT; 'GOTO' LOOP 'END' 'ELSE'
      'IF' C=&& 'THEN'
      'BEGIN' UOUT; GETC; UOUT;
              'IF' C=&** 'OR' C=&*^ 'THEN'
              'BEGIN' GETC; UOUT
              'END' ;
              'GOTO' LOOP;
      'END' 'ELSE'
      'IF' C>=&A 'AND' C<=&Z 'THEN'
      'BEGIN' CHOUT(OUT,&'); UOUT;
A3:           GETC;
              'IF' C>=&A 'AND' C<=&Z 'THEN'
              'BEGIN' UOUT; 'GOTO' A3
              'END' ;
              CHOUT(OUT,&'); UOUT;
              'GOTO' LOOP;
      'END' 'ELSE' UOUT;
      'GOTO' LOOP;

FIN:  CHOUT(OUT,&^Z); CLOSE(OUT);
      CLOSE(IN); 'GOTO' A1
'END'
'FINISH'
```

LISTA 1. TA LISTA PRIKAZUJE PROGRAM ZA PRETVORBO IZ NAČINA 2 (ZAPIS ALGOLSKEGA PROGRAMA Z VELIKIMI IN MALIMI ČRKAMI) V NAČIN 1 (ZAPIS ALGOLSKEGA PROGRAMA Z VELIKIMI ČRKAMI). V NAČINU 2 JE TA PROGRAM ZAPISAN V LEVEM STOLPCU IN V NAČINU 1 V DESNEM. V NAČINU 1 SO REZERVIRANE BESEDE ZAPRTE V ENOJNA NAREKOVAJA. IZ LISTE JE RAZVIDNO, DA JE NAČIN 2 PRIMERJNEJŠI (PREGLEDNEJŠI, LAŽJI ZA BRANJE) IN ZATO GA BOMO V TEM ČLANKU UPORABLJALI. NAČIN 1 (DESNI STOLPEC) JE BIL PREDVIDEN ZA UPORABO TELEPRINTERJA, KI IMA SAMO VELIKE (ALI SAMO MALE) ČRKE. V LISTI 2 JE PRIKAZAN INVERZNI ALGORITEM, KI PREPIŠE NAČIN 1 V NAČIN 2. OBA ALGORITMA STA TAKO LAHKO BISTVENA PRIPOMOČKA PRI PRETVORBI NAČINOV ZAPISOVANJA. PROGRAM NA TEJ LISTI KAŽE TUDI NEKATERE ZNAČILNOSTI PISANJA (OBLIKO IN STIL) PROGRAMOV V JEZIKU ALGOL 60 IN TUDI NEKATERE POSEBNOSTI JEZIKA RML ALGOL (NPR. &** PREDSTAVLJA ASCII ZNAK **, GLEJ TEKST ČLANKA). V TEM PROGRAMU SE POJAVLJAJO TUDI V/I PROCEDURE, ZNAČILNE ZA RML ALGOL (GETC, TEXT, CHOUT ITD.).

V tem prispevku opisane algolske programe bomo izvajali na operacijskem sistemu CP/M (računalniški sistem Partner proizvajalca Iskra Delta). Rezultate bomo večkrat primerjali s podatki v matematičnih tabelah ali pa jih bomo izračunavali s svinčnikom na papirju. Po potrebi bomo lahko prikazane algolske algoritme prepisali v druge programirne jezike (npr. v Pascal, PL/I, Ado, Basic itd.); v tem bo zajeta tudi širša koristnost obravnavanih programov. Seveda pa za nas pomembne problematike jezika Algol 60 ne bo moč opisati v enem samem članku.

Jezik Algol 60 je postal osnova za kasnejše programirne jezike, zlasti za nastanek jezika Pascal in Ada. Pascal je v bistvu omejil nekatere zamisli Algola (bloke) in dodal nove, s poudarkom na možnostih strukturiranega programiranja. Vendar je tudi v Algolu moč strukturirano programirati ((5)), potrebna pa je višja stopnja znanja kot za programiranje v Pascalu.

RML Algol je Algol 60 za male računalniške sisteme in je bil več let preizkušan na računalnikih PDP8 in PDP11. Sistem je sestavljen iz dveh delov: iz enoprehodnega prevajalnika za procesor Z80 in iz izvajalnega programa. Prevajalnik prevede izvirni algolski program v strojno neodvisen vmesni jezik, ki določa zaporedje subrutin in vsebuje subrutinske argumente. Vmesni jezik je zelo zgoščen in le 10 zlogov je v povprečju potrebnih za predstavitev algolskega stavka. Izvajalni program ima nalagalnik za prevajalniški izhod in druge rutine za izvajanje prevedenega algolskega programa.

Prevajalnik in izvajalni program zasedata le 12k zlogov (izvajata se v različnih časih: prevajalnem in izvajalnem). Za prevajanje zadostuje že 21k zlogov pomnilnika, za izvajanje pa 16k zlogov. Algolski programi, ki so bili razviti na sistemu RML Algol, se lahko uporabijo tudi na sistemih s procesorjem PDP11.

2. Splošne opombe o jeziku RML Algol

RML Algol ima glede na Algol 60 nekatere omejitve in razširitve. Omejitve so pretežno posledica enoprehodnega prevajalnika. Vse spremenljivke morajo biti deklarirane pred svojo uporabo. Razširitve obsegajo uvedbo podatkovnega tipa BYTE ARRAY, logične operatorje, operator MOD in vrsto vhodnih/izhodnih funkcij.

Omejitve so natanko tele:

- OWN spremenljivke niso implementirane
- večkratne prireditve niso dovoljene
- celoštevilske označitve ne obstajajo
- spremenljivke morajo biti deklarirane pred uporabo
- klic z imenom je omejen in dejanski parameter je ime spremenljivke (kot pri pozivu z navedbo v Fortranu)
- parametri polja se morajo klicati z imenom
- algolska "krepka" vejica ni implementirana
- večkrat so potrebna navodila za indikacijo tipov pri procedurnih parametrih in pogojnih stavkih
- le prvih šest znakov v imenih je bistvenih

Razširitve glede na Algol 60 pa so tele:

- uveden je podatkovni tip BYTE ARRAY
- uvedeni so operatorji MOD, !, DIFFER, MASK
- komentarji se lahko zaprejo v zavite oklepaje ({ , })
- procedurna imena so lahko rezultat označitvenih izrazov
- dodatne funkcije vsebujejo obdelavo nizov, neposredni diskovni V/I, bločni pomik, brisanje polj, grafiko itd. (glej Dodatek 2)

Sintaksa jezika RML Algol je opisana v Dodatku 1. Slovstvo, ki obravnava programiranje v jeziku Algol 60, je zbrano na koncu članka ((7 - 11)).

3. Kratek priročnik jezika RML Algol

3.1. Ključne besede in imena

Algol zahteva razločevanje ključnih besed in imen. Prevajalnik za RML Algol loči avtomatično dva načina:

Način 1. Algolski program je zapisan samo z velikimi črkami: vse ključne besede morajo biti v tem načinu zapisane med enojnima narekovajema. Tako imamo npr.:

'BEGIN' 'END' 'FOR' 'STEP' 'UNTIL' 'DO'

V listi 1 imamo algolski program UCASE.ALG, ki prevede izvirno zbirko, zapisano v načinu 2 (velike in male črke) v način 1 (vse črke so velike). V tej listi je na levi strani prikazan program UCASE.ALG, izpisan z velikimi in malimi črkami, na desni strani pa prevod zbirke UCASE.ALG s tem istim programom v program, ki je zapisan z velikimi črkami in ključnimi besedami v narekovajih.

Način 2. Algolski program je zapisan z malimi in velikimi črkami: vse ključne besede morajo biti zapisane z velikimi črkami, imena pa z malimi. Tako imamo npr.:

```
FOR ime := 1 STEP 2 UNTIL maks DO
```

V listi 2 imamo algolski program LCASE.ALG, ki prevede izvirno zbirko, zapisano v načinu 1 (velike črke) v način 2 (velike in male črke). Način 2 je priročnejši in bolj pregleden.

Prevajalnik predstavi notranje vsa imena z velikimi črkami. Ključne besede morajo biti medsebojno ločene: torej THEN GOTO in ne THENGOTO. V načinu 1 se število zapiše z velikim "E", torej kot 1.234E-5, v načinu 2 pa kot 1.234e-5, torej z malim "e"; notranje se "e" vselej bere kot veliki "E".

3.2. Predddeklarirana imena

Vrsta procedurnih imen bo razpoznanih brez deklariranja. Te bomo razumeli, kot da so deklarirana v bloku, ki obdaja program. Pri tem imamo vhodne/izhodne rutine. Te rutine bodo npr. tudi

```
loc(n);
```

Predddeklarirane so tudi standardne funkcije sin, cos, arctan, ln, exp, sqrt, abs, sign, entier, ki so opisane v Dodatku 2.

3.3. Zgradba algolskega programa

Jezik Algol je strukturiran. V oklepajih BEGIN in END so vklenjeni stavki in od teh oklepajev je odvisen vrstni red izvajanja. Sestavljen stavek ima deklaracije in blok in splošno velja za RML Algol

```
BEGIN s1; s2; ... ; sn END FINISH
```

Stavki s1; s2; ... ; sn so lahko zopet sestavljeni stavki in bloki. Konec programa je označen s FINISH.

Blok, v katerem je ime deklarirano, določa ob-

```
BEGIN INTEGER in,out,c;

COMMENT this program converts from
the upper case in quotes mode to the
upper/lower case convention;

PROCEDURE seto(a);
VALUE a; INTEGER a; ioc(31);

INTEGER PROCEDURE swlist;
ioc(39);

PROCEDURE getc;
BEGIN c:=chin(in);
      IF c<0 OR c=&^Z THEN GOTO fin
END ;



      seto(swlist+13);
      text(10,"ALG"); {default extension}
a1:   ioc(2); in:=input;
      IF in<1 THEN GOTO a1;
      out:=output;
      IF out<1 THEN
      BEGIN close(in); GOTO a1;
      END ;
loop: getc;
a2:   IF c=&' THEN
      BEGIN
a3:           getc;
              IF c=&' THEN
              BEGIN getc;
                    IF c>&*8 AND c#&; THEN
                    chout(out,&*8); GOTO a2;
              END ELSE
              IF c<&A OR c>&Z THEN GOTO a2
              ELSE chout(out,c);
              GOTO a3
      END ELSE
      IF c>=&A AND c<=&Z THEN  c:=c+32;
      chout(out,c);
      IF c=&& THEN
      BEGIN getc; chout(out,c);
              IF c=&** OR c=&*^ THEN
              BEGIN getc; chout(out,c);
              END
      END ELSE
      IF c=&" THEN
      BEGIN
a4:           getc; chout(out,c);
              IF c=&** THEN
              BEGIN getc; chout(out,c);
              END ELSE
              IF c=&" THEN GOTO loop;
              GOTO a4;
      END ;
      GOTO loop;
fin:  chout(out,&^Z); close(out);
      close(in); GOTO a1;
END
FINISH
```

LISTA 2. PROGRAM NA TEJ LISTI PRETVORI ZAPIS Z VELIKIMI ČRKAMI V ZAPIS Z MALIMI IN VELIKIMI ČRKAMI. TUDI TA PROGRAM JE HKRATI NAZOREN PRIMEREK ZAPISA PROGRAMA V JEZIKU ALGOL.

močje (vidljivost) tega imena. Izven bloka to ime nima pomena in ne zaseda pomnilniškega prostora. Dodeljevanje pomnilnega prostora v Algolu je dinamično, skladno z izvajanjem programa. Ekonomičnost dodeljevanja pomnilnika se lahko poveča z dinamičnimi mejami v deklaraciji polja (aritmetični izrazi). Enaka imena se lahko uporabljajo v različnih blokih (tudi v vgnezdenih).

3.4. Bloki in deklaracije

Spremenljivke morajo biti deklarirane v RML Algolu pred svojo uporabo in prisotnost teh deklaracij povzroči, da nastane iz sestavljenega stavka blok. Vse deklaracije morajo biti zapisane neposredno za BEGIN oziroma za naslednjo deklaracijo. Izjema so le označitve, ki se deklarirajo z imenom in dvopičjem pred označenim stavkom. Deklaracija enostavnih spremenljivk je npr.

```
TYPE ime1, ime2, ... , imeN;
```

kjer je TYPE lahko INTEGER, REAL ali BOOLEAN. Spremenljivke se uporabljajo v bloku, v katerem so deklarirane (do zaklepaja END). Procedure in označitve se lahko uporabijo pred svojimi deklaracijami, toda morajo biti v območju točke uporabe. Spremenljivke in stikala morajo biti deklarirana pred svojo uporabo.

Primer:

```
BEGIN REAL x, y, p; INTEGER i, p;
   s1; s2;
   BEGIN REAL x, z; s3;
   END
END
```

Stavka s1 in s2 ne moreta navajati z, ker je ta izven območja; stavek s3 ne more navajati prvega x, ker bo veljal v tem območju drugi x (prednost pred prvim x). Vendar s3 lahko navaja y in i. Ime p je deklarirano dvakrat in tu se bo pojavila napaka v prevajalnem času.

3.5. Programska oblika

Oblika algolskega programa je odvisna od programerja. Znaki 'nova vrstica', 'presledek' in 'tabuliranje' se upoštevajo le izjemoma, in sicer:

- presledki so bistveni v nizih
- ključne besede in dvoznakovni simboli ne smejo vsebovati zgoraj naštetih posebnih znakov; npr.

```
BEGIN     toda ne    BE GIN
a := b;   toda ne  a : = b;
a <= b    toda ne  a<  = b
```

Drugih omejitev ni.

Oblika algolskega programa naj bi jasno (vidno) prikazovala bločno strukturo programa, s stolpno poravnanima oklepajema BEGIN in END za posamezen blok. Označitve naj bi vselej začele novo vrstico na levi strani programa. Stavki so ločeni s podpičji, s ključnimi besedami in ne z znakom nove vrstice. Izrazi se lahko razprostirajo prek več vrstic. V vrstici je lahko tudi več krajših stavkov.

Imena naj ponazarjajo pomen podatka, ki ga izražajo. V imenih se lahko pojavljajo presledki, če to prispeva k jasnosti: prevajalnik jih ne upošteva. Pogosto nastopajoče spremenljivke in poljski indeksi se navadno označujejo z enočrkovnimi imeni, ko imamo npr.:

```
end of file := znak = &^Z;
FOR i:=spodnji STEP int UNTIL
                    zgornji DO ...
```

Komentarji naj se vključujejo na mestih, kjer je pojasnjevanje programa priporočljivo.

3.6. Podatkovni tipi, imena, simboli

Podatek v pomnilniku, ki ga je moč manipulirati, je številski ali logičen (boolovski). Številske vrednosti so realne ali celoštevilske. Cela števila nimajo ulomljenega dela in zaseda-

jo po dva zloga; realne vrednosti pa imajo mantiso in eksponent in zasedajo štiri zloge. Algolski sistem pretvarja števila iz enega v drugi tip, ko je to potrebno. Aritmetični izrazi so lahko mešane realne in celoštevilske vrednosti.

Številski in logični podatki se lahko pojavljajo v programu kot vrednosti literalov. Celoštevilski literal nima decimalne vejice in eksponentnega dela in njegove vrednosti so v intervalu (-32768, 32767). Realna števila imajo decimalno vejico ali decimalni eksponent, označen z 'E' ali z 'e' (odvisno od načina), ali oboje. Primeri so:

```
0.1     -2.345     1.2E3     25.7e-7
```

Realna števila uporabljajo en zlog za eksponent in tri zloge za mantiso in so v približnjem intervalu (E-38, E+38) s 6 do 7 decimalnimi številkami.

Logična literala sta ključni besedi TRUE in FALSE.

RML algolski program je sestavljen iz zaporedja simbolov, simboli so sestavljeni iz znakov. Urejevalne in posebne oblikovalne znake prevajalnik ne upošteva. Simboli se združujejo v enote, ki jih prevajalnik obravnava kot enote. Takšne enote (skupine) so številske konstante, ključne besede in imena. Samo prvi dve črki ključne besede sta bistveni. Imena določa programer za spremenljivke, označitve, stikala, polja in procedure. Dvoznakovni simboli v Algolu so prireditveni operator (:=), večje kot ali enako (>=) in manjše kot ali enako (<=). Prvi znak imena mora biti črka, dolžina imena ni omejena, vendar se upošteva le prvih 6 znakov. Tako sta imeni abc123 in abc1234 identični.

Imena spremenljivk in stikal morajo biti deklarirana pred njihovo uporabo. Označitve in procedure se lahko uporabljajo pred deklariranjem, ker je tip imena mogoče določiti iz programskega konteksta.

3.7. Polja

Indeksirane spremenljivke v Algolu so elementi polja in so lahko realne, celoštevilske in boolovske. Zlogovna spremenljivka (byte) je vselej indeksirana. Polja se deklarirajo na začetku bloka, in sicer v obliki

```
ARRAY ime [ai:ai] ;
```

Tu sta ai aritmetična izraza, ki določata indeksne meje, negativna in/ali pozitivna, le zgornja meja ne sme biti nižja od spodnje. Realni ai se zaokroži na najbližje celo število. Pri več indeksih imamo npr. tole:

```
INTEGER ARRAY cp [1:30,1:4];
BOOLEAN ARRAY bp1,bp2 [1:n,1:3],bp3 [-1:5];
              bp4,bp5 [1:2*n];
```

V poljih z deklariranimi spremenljivimi mejami morajo imeti spremenljivke določene vrednosti; te naj bi bile deklarirane v zunanjem bloku in njim prirejene vrednosti.

Pri uporabi polj je lahko aritmetičen izraz na mestu indeksa. Imamo tele možne primere:

```
cp [ 1,4] := 7;
cp [n,m] := cp [ cp [ n,2], cp[n,m]] ;
cp [n,3] := IF bp2 [ 3,1] AND bp3[ n] THEN
              cp [cp [ 3*n,1], IF bp3 [ 0] THEN
              2 ELSE n] ELSE n;
```

Kakšno je dodeljevanje pomnilnika poljem in preizkušanje poljskih mej? V večrazsežnostnih poljih se najhitreje spreminja zadnji indeks. Imejmo npr.

```
a [m:n,p:q]
```

Naslov felementa a [i,j] je dan z izrazom

```
osnova+obseg*((i-m)*(q-p+1)+(j-p))
```

kjer je obseg 1, 2 ali 4 v odvisnosti od tipa polja (boolovsko, zlogovno, celoštevilsko ali realno). V času izvajanja se preizkusi izračunani naslov, ali leži v dodeljenih pomnilniških mejah polja. Te meje niso neposredno povezane z deklariranimi mejami! Npr. element a [-1,15] je sprejemljiv za polje, ki je bilo deklarirano kot a [0:9,0:9] .

RML Algol uvaja kot novost tkim. BYTE polja (zlogovna polja). Ta polja omogočajo učinkovito uporabo pomnilnika pri manipulaciji z nizi ali z malimi celoštevilskimi vrednostmi (0 do 255). V izrazih se elementi zlogovnih polj obravnavajo kot cela števila in se lahko uporabljajo v celoštevilskih kontekstih. V izrazu se tak element uporablja kot celoštevilska vrednost, ki ima osem višjih bitov enakih 0. Tako bo

```
i := b [n]
```

vselej pozitivno za i v vrednostnem intervalu (0, 255).

Prireditev k elementu zlogovnega polja zahteva pretvorbo izraza na desni strani prireditvenega simbola v celo število in osem nižjih bitov se potem priredi, 8 višjih bitov celega števila pa se enostavno odreže. Tako dobimo pri

```
b [1] := -1; i := b [1]
```

vrednost 255 za i. Prevajalnik ločuje celoštevilska in zlogovna polja samo v deklaracijah; v drugih kontekstih se oba tipa lahko medsebojno uporabljata: to velja tudi za dejanske parametre. Npr.:

```
BEGIN INTEGER ARRAY i [0:100];
      BYTE    ARRAY b [0:100];

PROCEDURE xx(a); INTEGER ARRAY a;
BEGIN
      . . . . . .
END;
      xx(i); xx(b); . . .
```

je sprejemljivo.

3.8. Enostavni izrazi

Izraz je tisti del programa, ki da rezultat; ta je tipa REAL, INTEGER, BOOLEAN ali LABEL.

Realna operatorja sta potenciranje (znak ^) in realno deljenje (znak /). Aritmetični izrazi se izračunavajo z upoštevanjem operatorske prednosti od leve proti desni, če so prednosti enake. Oklepaji spreminjajo vrstni red izračunavanja izraza. Seznam aritmetičnih in logičnih operatorjev s pripadajočimi prednostmi (prioritetami) je tale:

operator	prednost	pomen
^	3	potenciranje (najvišja prednost)
*	2	množenje
/	2	realno deljenje
%	2	celoštevilsko deljenje
MOD	2	celoštevilski modul
+	1	seštevanje
-	1	odštevamje
MASK	1	logični 'in'
DIFFER	1	logični 'ekskluzivni ali'
!	1	logični 'ali' (najnižja prednost)

Operatorji MOD, MASK, DIFFER in ! so dodani in jih v Algolu 60 ni. Operatorji %, MOD, MASK, DIFFER in ! dajo celoštevilske rezultate. Rezultat celoštevilskega deljenja (%) je zaokrožen proti ničli. Rezultat celoštevilskega modula (MOD) je ostanek pri celoštevilskem deljenju in

```
i MOD 0
```

vrne vselej vrednost nič. Operaciji

```
i%0   in   x/0
```

pomenita deljenje z ničlo, ki povzroči napako v času izvajanja.

Logični operatorji MASK, DIFFER in ! predpostavljajo dva celoštevilska, 16-bitna argumenta in velja npr.

```
3 ! 5       = 7
3 MASK 5    = 1
3 DIFFER 5  = 6
```

Izrazi lahko vsebujejo mešanico celoštevilskih in realnih količin, pretvorba med tipi se pojavi samodejno v kontekstu izraza. Npr.:

```
BEGIN REAL x, y;
      INTEGER i, j;
      . . . . . .
      i := x*y; x := x*i; i := x^y;
      i := x^i; x := i%j;
      . . . . . .
```

so veljavne operacije. V pretvorbi iz realnega v celoštevilsko se rezultat zaokroži na vrednost najbližjega celega števila.

Boolovski izrazi so sestavljeni iz boolovskih spremenljivk, iz literalov TRUE in FALSE, aritmetičnih relacij, boolovskih procedur in boolovskih operatorjev. Ti operatorji so v zaporedju padajoče prednosti tile:

NOT	NOT b: ima vrednost FALSE, če ima b vrednost TRUE in obratno
AND	b AND c: ima vrednost TRUE, če sta b in c TRUE, sicer vrednost FALSE
OR	b OR c: ima vrednost TRUE, če je b ali c ali oba TRUE, sicer vrednost FALSE
IMPLIES	b IMPLIES c: ima vrednost FALSE, če je b TRUE in c FALSE, sicer vrednost TRUE
EQUIVALENT	b EQUIVALENT c: ima vrednost TRUE, če imata b in c enako pravilnostno vrednost, sicer vrednost FALSE

Relacijski operatorji so tile:

=	enako
>	večje kot
>=	večje kot ali enako
<	manjše kot
<=	manjše kot ali enako
#	neenako

Relacijski operatorji so dladni. Pazljivost je potrebna pri uporabi operatorjev = in #, če sta operanda tipa REAL. Tu imamo primerjavo bitov in operanda morata imeti enaka bitna vzorca. Zato je večkrat smotrnejša primerjava na majhno absolutno razliko. Npr.

```
IF x = y THEN ...
```

zamenjamo z

```
IF abs(x - y) < 0.0001 THEN ...
```

Pri spremembi vrstnega reda izračunavanja uporabimo lahko tudi oklepaje (kot pri aritmetičnih izrazih).

Primeri boolovskih izrazov so:

```
NOT x<5 OR bs1 AND (y=0 OR bs2)
x # (y-5)
+(x-5)^2  <= 20
```

kjer sta x in y številski spremenljivki ali proceduri, bs1 in bs2 pa boolovski spremenljivki ali proceduri. Znak + v srednji vrstici pove, da je izraz v oklepaju aritmetičen (ne logičen kot v prvi vrstici).

3.9. Nizi in znakovni literali

Niz je zaporedje znakov, zapisanih med dvojna narekovaja, npr.

```
text(1, "Dober dan, žalost");
```

Znaki v nizu z ASCII vrednostjo manjšo od 32 (20H) se ne upoštevajo. Med te znake spadajo znak vrnitve valja, tab, krmilni znaki, presledek pa je sprejemljiv. Znaka '*' in '^' imata poseben pomen in se obravnavata posebej.

V niz vstavimo krmilni znak z dvema znakoma, tako da pišemo ^x (učinek tega je znak control-X).

Oblikovalni znaki so tile:

*N	vstavi CR/LF v niz
*C	vstavi CR
*L	vstavi LF
*S	vstavi presledek (tudi sam znak presledka je dovoljen)
*T	vstavi TAB
*P	vstavi novo stran (formfeed)

Pri tem velja:

**	vstavi znak '*'
*^	vstavi znak '^'
*"	vstavi znak '"'

Imamo tale primer:

```
text(1, "Primer *"NIZA*"
        *Slahko izgleda ko
        t*NX*^2+Y*^2=");
```

Na konzoli se izpiše tole:

```
Primer "NIZA" lahko izgleda kot
X^2+Y^2=
```

Notranja predstavitev niza je zaporedje znakov, shranjenih kot zaporedje zlogov, ki končuje z znakom nič (00H).

Oglejmo si še znakovne literale. Literalno vrednost nekega znaka dobimo z uporabo znaka '&', ki mu sledi zadevni znak. Kadar želimo pretvoriti znakovno število v število med 0 in 9, uporabimo npr.:

```
i := chin(1)-&0;
```

ali če želimo izdati znak X

```
chout(1,&X);
```

To velja tudi za primere z znakoma '*' in '^', ko preizkušamo na znak control-Z

```
i := chin(dev);
IF i=&^Z THEN ...
```

ali na znak CR

```
IF i=&*C THEN ...
```

To pa ne velja za &*N in &*" (tu velja &"). Znakovni literali so celoštevilski tip.

3.10. Prireditveni stavki

Splošna oblika prireditvenega stavka je

```
spremenljivka := izraz;
```

Spremenljivki na levi strani se priredi vrednost izraza na desni. Prireditveni operator je dvoznakovni simbol (:=) in ne preprosto enačaj (=), ki je relacijski operator. V RML Algolu niso implementirane večkratne prireditve. Če je spremenljivka boolovska, mora biti tega tipa tudi izraz. Če je spremenljivka številska, je lahko izraz celoštevilski ali realen (kot rezultat); ta rezultat se bo po potrebi pretvoril v tip spremenljivke; realna števila se pri tej pretvorbi zaokrožijo na najbližje celo število. Imamo tele primere:

```
i  := 3+j;
x  := IF bi THEN j%5 ELSE 136.12;
bs := i#7;
```

3.11. Pogojni izrazi

Pogojni izraz lahko dobi več vrednosti v odvisnosti od rezultata boolovskega izraza. Splošna oblika pogojnega izraza je

```
IF bi THEN ei ELSE izraz
```

Tu je bi boolovski izraz (ki je lahko zopet pogojen), ei je enostaven izraz, ki ne sme biti pogojen, izraz pa je poljuben izraz. Ker je izraz lahko tudi pogojen izraz, dobimo npr.

```
IF bi THEN ei ELSE IF bi THEN ei ELSE izraz
```

Izrazi morajo biti vsi številski, vsi boolovski ali vsi označitveni (glej kasneje). Pogojni izraz postane nepogojen z uporabo oklepajev. V Algolu je dopustno tole:

```
IF IF bs1 THEN x#3 ELSE y=0 THEN
(IF bs2 THEN 25 ELSE 30) ELSE x+y
```

Prvi bi je pogojen. ai (aritmetičen izraz) v oklepajih je pogojen in bi brez oklepajev ne bil dopusten. Splošno pravilo v Algolu je, da IF ne sme slediti takoj za THEN. Ta prepoved velja zaradi možnosti nastanka dvoumnega koda. Primeri pogojnih izrazov so:

```
a := IF a>0 THEN a*a ELSE 0;
vel := IF a>b THEN a ELSE b;
maks := IF a>=b AND a>=c THEN a
        ELSE IF b>=a AND b>=c
             THEN b ELSE c;
a := IF IF x>0 THEN y>0 ELSE y<50
     THEN 3*x ELSE 0;
```

3.12. Pogojni stavki

Pogojni stavki imajo enako obliko kot pogojni izrazi, vendar ni potrebno, da imajo ELSE del. Očitno imamo dve obliki pogojnih stavkov:

```
IF bi THEN s1;
IF bi THEN s1 ELSE s2;
```

Stavek s1 ne sme biti pogojen, stavek s2 je pa lahko. V prvem primeru ne dobimo stavka, če je bi nepravilen (rezultat FALSE). Ker je s2 lahko pogojen, moremo drugo obliko neomejeno razširiti:

```
IF bi1 THEN s1 ELSE
        IF bi2 THEN s2 ELSE s3;
```

Primeri pogojnih stavkov so:

```
IF a>0 THEN vsota := vsota+a;
IF znak = &^Z THEN close(dev);
IF vzorec  maks THEN maks := vzorec
   ELSE IF vzorec<min THEN
   min := vzorec;
IF a>b AND c>d THEN
   BEGIN . . . END
ELSE
   BEGIN . . . END;
```

3.13. FOR stavki

FOR stavek omogoča ponovljeno izvajanje stavka z različnimi vrednostmi spremenljivke, ki je znana kot krmiljena spremenljivka. Splošna oblika FOR stavka je

```
FOR spr := efl,efl, ... efl DO s1
```

Stavek s1 je lahko pogojen. Krmiljena spremenljivka mora biti tipa REAL ali INTEGER, ne sme pa biti indeksirana. (Standardni Algol 60 dovoljuje tudi indeksirano spremenljivko.) efl pomeni "element FOR liste". Lista z enim samim elementom je dovoljena. Imamo tri vrste elementov liste, in sicer

- aritmetičen izraz
- STEP element in
- WHILE element

STEP element ima obliko

```
ai STEP ai2 UNTIL ai3
```

Po vsaki izvršitvi krmiljenega stavka se vrednost ai2 prišteje k spremenljivki. Pred vsako izvršitvijo stavka s1 (glej definicijo FOR stavka) se spremenljivka preizkusi na vrednost izraza ai3. Vrednosti izrazov ai2 in ai3 se med izvajanjem stavka s1 lahko spreminjata, zato moramo izrazoma pred njuno uporabo izračunati vrednost.

WHILE element ima obliko

```
ai WHILE bi
```

Pri vsaki ponovitvi se izraz ai izračuna in njegova vrednost se priredi spremenljivki. Potem se izračuna boolovski izraz bi in če je njegova vrednost TRUE, se stavek izvrši, sicer je element izčrpan. Primeri FOR stavkov pa so tile:

```
FOR i:=min STEP 1 UNTIL maks DO
   vsota := vsota + s[i];
FOR i:=1 STEP 1 UNTIL 1024 DO ...
FOR i:=1, 3, 99, j, -6, 11 DO ...
FOR x:=1, x*2 WHILE x<1025 DO ...
FOR i:=100 STEP -1 UNTIL -100 DO ...
FOR x:=0.1, 1, x*5 WHILE x<100,
   20 STEP -5 UNTIL 0 DO ...
```

FOR zanke so lahko vgnezdene poljubno globoko. Na primer

```
FOR i:=1 STEP 1 UNTIL maks DO
FOR j:=1 STEP 1 UNTIL i DO
   a := a + b[i,j]^2;
```

Matrično množenje pa je lahko tole:

```
FOR i:=1 STEP 1 UNTIL m DO
FOR j:=1 STEP 1 UNTIL n DO
BEGIN x:=0
      FOR k:=1 STEP 1 UNTIL p DO
          x:=x+a[i,k]*b[k,j];
      c[i,j] := x;
END;
```

Telo FOR zanke je lahko prazen stavek, ko čakamo na znak CR:

```
FOR i := chin(dev) WHILE i#*C DO;
```

Tudi zančna spremenljivka je lahko prazna:

```
FOR i:=0 WHILE test DO telo;
```

Tu je telo procedura ali blok, ki določa vrednost boolovske spremenljivke test ali pa je test boolovska procedura. V primeru

```
FOR i:=telo1 WHILE test DO telo2;
```

je telo1 lahko celoštevilska ali realna procedura in telo2 procedura ali blok; možnosti za zgraditev zanke so torej v Algolu 60 zares raznovrstne.

FOR zanka ni enostaven sravek in je ni mogoče poklicati v THEN delu pogojnega stavka brez uporabe oklepajev BEGIN in END; lahko pa se pokliče v ELSE delu brez teh oklepajev.

3.14. Prazni stavki

Prazen stavek se pojavlja tedaj, ko se ničesar ne pojavi pred ključnima besedama END ali ELSE ali pred znakom ';'. Primeri praznih stavkov so:

```
BEGIN END
IF b1 THEN;
;;
BEGIN s1; END
IF b1 THEN ELSE;
PROCEDURE prazno; ;
```

3.15. Komentarji

RML Algol dovoljuje tri vrste komentarjev. Poljubni simboli, ki se pojavljajo za END do prve pojavitve podpičja, END, ELSE ali FINISH, se smatrajo kot komentar in se ne upoštevajo. To so tkim. END komentarji. Imamo

```
END To se ne upošteva;
END To se tudi ne upošteva END
END To tudi ne ELSE
END To prav tako ne FINISH
```

Druga oblika komentarja je

```
COMMENT poljuben niz brez podpičja;
```

Ta oblika je dovoljena za podpičjem ali za besedo BEGIN. Enojni narekovaji se lahko v komentarju uporabljajo le paroma.

Tretja oblika komentarja je tekst v zavitih oklepajih in ti se smejo v komentarju uporabljati še paroma. Npr.

```
{to je {komentar}, ki se ne upošteva}
```

Standardni Algol 60 dopušča še komentarje v procedurnih pozivih in deklaracijah.

3.16. Označitve, razdelilniki in GOTO stavki

Vsak stavek je lahko označen z imenom in dvopičjem. Območje veljavnosti (vidljivosti) označitve je blok, v katerem se označitev pojavlja.

Programsko izvajanje se prenese na označeni stavek z GOTO stavkom. Primer:

```
BEGIN REAL x; s1; s2; GOTO ozn; s3;
ozn: s4
END;
```

Ni pa dovoljen primer, ko označitev ni v območju veljavnosti:

```
GOTO ozn;
BEGIN REAL x; s1; s2;
ozn: s3;
END;
```

Označitev v zunanjem bloku je dosegljiva iz notranjega bloka, npr.:

```
BEGIN REAL q;
ozn: s1; s2;
     BEGIN REAL y;
        s3; GOTO ozn;
     END
END
```

Poudariti velja, da sestavljen stavek ne postane blok, če se v njem pojavljajo označeni stavki (blok mora imeti neprazen deklaracijski del). Označitve v različnih sestavljenih stavkih istega bloka morajo imeti različna imena.

Razdelilnik (stikalo, pretikalo) je seznam (lista) označitev na začetku bloka. Vse označitve morajo biti v območju (v deklaraciji). Primer:

```
SWITCH s := ozn1, ozn2, ozn3;
```

Enostavna uporaba razdelilnika se doseže z GOTO stavkom:

```
GOTO s[ai];
```

Aritmetični izraz ai se izračuna in se uporabi kot indeks za listo označitev v deklaraciji. Če ima ai vrednost 2, je učinek tega stavek GOTO ozn2. Če ai preseže vrednost indeksov, se ta GOTO stavek vzame kot prazen stavek. Primer:

```
BEGIN SWITCH st := pr1, pr2, pr3;
poskusi: text(1, "primer stev=");
         GOTO st[chin(1)-&0 ];
         text(1, "*Nneveljavna vrednost");
         GOTO poskusi;
pr1:     . . . . . .
pr2:     . . . . . .
pr3:     . . . . . .
```

3.17. Označitveni izrazi

Označitveni izrazi so podobni aritmetičnim in boolovskim izrazom. Vrednosti označitvenih izrazov so označitve ali razdelilniški elementi. Polna definicija GOTO stavka je

```
GOTO oi;
```

kjer je oi označitveni izraz. Primer:

```
GOTO IF x - 0 THEN ozn1
     ELSE IF b THEN s[i+3]
     ELSE ozn2
```

Vrednost označitvenega izraza je lahko tudi naslov procedure, kot bo opisano v odstavku o procedurnih parametrih. Ta primer pomeni razširitev jezika Algol 60, v katerem so označitveni izrazi dovoljeni samo za označitve.

3.18. Procedure

Procedura je deklaracijski stavek na začetku bloka, ki pa se pri vstopu v blok ne izvaja. Procedura ima svoje ime in z imenom se pokliče v izvajanje. Enostavna procedura nima parametrov in ne da rezultata (eksplicitno). Npr.:

```
BEGIN
PROCEDURE storitev; s;
   s1; s2; storitev; s3; storitev
END
```

Tu je s stavek, ki se izvrši, ko se 'storitev' kot klic pojavi v programu. 's' je telo procedure 'storitev', ki se obravnava kot blok (procedure so bloki), čeprav je telo enostaven stavek (vobče je telo sestavljen stavek). Bločna struktura procedure ščiti telo pred vstopom vanj z GOTO stavkom, ko telo ni aktivno.

Procedura lahko da rezultat tipa REAL, INTEGER ali BOOLEAN. Takšna procedura je znana kot funkcija ali tipska procedura in njeno ime se lahko uporablja v izrazih. V telesu te procedure se vrnjena vrednost priredi imenu procedure. Izvajanje procedure se konča na njenem koncu ali s skokom GOTO iz nje. Procedura je lahko rekurzivna, če kliče sama sebe oziroma če se njeno ime pojavi na desni strani priredit venega stavka v njenem telesu. Imejmo tale primer:

```
BEGIN INTEGER i;
   INTEGER PROCEDURE j;
   IF i < 0 THEN GOTO slabo
   ELSE IF i = 1 THEN j := 0 ELSE
   BEGIN i := i-1;
         j := j+i
   END procedure j;
   i := 10; i := j;
slabo:
END
FINISH
```

Procedura j navaja spremenljivko i, ki je deklarirana v glavnem programu, v istem bloku kot procedura j. Deklaracija za i se more pojaviti, sicer bi prevajalnik predpostavljal, da je i še nedeklarirana boolovska (vrstici 3 in 4) ali celoštevilska (vrstica 6) procedura. Vrstica 5 bi bila napačna, ker bi moralo biti ime i levo od simbola := že deklarirano. Vselej mora obstajati pogoj, ki povzroči, da rekurzivna procedura izračuna rezultat ali izstopi brez rekurzije (kot npr. v vrsticah 3 in 4). Če procedura ne bi bila tako zgrajena ali če se i ne bi zmanjševal v vrstici 5, bi procedura klicala sama sebe, dokler ne bi izčrpala pomnilnega prostora. Označitev 'slabo' označuje prazen stavek. Telo procedure ni zaprto v oklepaja BEGIN in END, ker je telo pogojni stavek.

3.19. Procedure s parametri

Delovanje procedure je lahko odvisno od vrednosti, ki jih imajo njeni parametri v trenutku njenega poziva. Procedurna deklaracija ima listo formalnih parametrov, ki so imena, uporabljena v procedurnem telesu. Tip formalnega parametra je določen s specifikacijo, ki je podobna množici deklaracij za neindeksirane spremenljivke. Lista formalnih parametrov je zaprta v okrogle oklepaje in se nahaja tik za procedurnim imenom. Imena so ločena z vejicami. Imejmo npr.:

```
REAL PROCEDURE p(x,y,a,r,ozn);
VALUE y; REAL x,y;
REAL ARRAY a;
REAL PROCEDURE r;
LABEL ozn;
```

V standardnem Algolu 60 je dovoljen bolj zapleten tip parameterskega separatorja.

3.20. Številski in boolovski parametri pri klicu z vrednostjo

Parametri pri klicu procedure z vrednostjo so najenostavnejši. Pri klicu procedure se dejanski parametri izračunajo in vrednosti se posredujejo proceduri. Parameter procedure, klicane z vrednostjo, se obnaša kot spremenljivka, ki je bila deklarirana v proceduri; izjema je le začetna vrednost, ki se tej spremenljivki priredi ob vstopu v procedurno telo. Spremenljivka se med izvajanjem procedure vobče spreminja, toda njena nova vrednost je dostopna šele ob izstopu iz procedure. Imejmo tale primer:

```
PROCEDURE p(i,x,b); VALUE x,b,i;
REAL x; BOOLEAN b; INTEGER i;
BEGIN IF b THEN a:=x+i
      ELSE a:=x-i;
x:=2*a; a:=x+i;
END
```

Tu je a spremenljivka, definirana izven procedure. VALUE specifikacija se mora pojaviti pred deklaracijo tipov. RML Algol pretvarja vrednosti med tipoma REAL in INTEGER, če se dejanski in formalni parametri tipsko ne ujemajo. Druge tipske pretvorbe ne nastopajo. Možen klic gornje procedure je

```
p(1.23, 4.5*y, TRUE AND z > 0);
```

kjer sta x in z številski spremenljivki. Pomembno je TRUE pred relacijo z > 0, ker sicer prevajalnik ne bi imel podatka, da prevaja boolovski izraz. Vsi dejanski boolovski parametri morajo začeti z boolovskim imenom ali z enim izmed simbolov NOT, TRUE ali FALSE. Podobno mora biti že deklarirano ime, ki je prvi del dejanskega parametra, tako da prevajalnik lahko sklepa o tipu izraza. Ni potrebno, da je dejanski parameter zaprt v oklepaje (to so posebnosti RML Algola).

3.21. Klic spremenljivke z imenom

Procedurni parameter, ki ni bil specificiran z VALUE, se pokliče z imenom. V tem primeru se proceduri ne posreduje vrednost temveč naslov vrednosti (to je dejansko klic z navedbo). Dejanski parameter mora imeti natanko tip formalne spremenljivke. Vrednost spremenljivke se tu vselej spremeni tudi na zunanjem naslovu in ne samo na lokalnem naslovu v proceduri. Spremenljivke, klicane z imenom, se ne obnašajo več lokalno.

V RML Algolu se poljski parametri morajo klicati z imenom (navedbo). Dejanski parameter je ime polja brez indeksov. V procedurnem telesu se formalno ime polja uporablja z indeksi. Število indeksov mora biti enako številu indeksov v izvirni deklaraciji. RML prevajalnik dopušča izmenično uporabo zlogovnih (BYTE ARRAY) in celoštevilskih polj v procedurnih pozivih.

Imensko klicani objekti se lahko posredujejo med procedurami z uporabo liste parametrov. Uporabljajo se lahko neindeksirane spremenljivke, klicane z imenom kot krmiljene spremenljivke v FOR stavkih.

Imejmo tale primer:

```
BEGIN ARRAY polje[1:20];
REAL x; INTEGER i;

REAL PROCEDURE rp; rp := i^2 + 2;

PROCEDURE p(a,k,z); VALUE a;
REAL a; INTEGER k; ARRAY z;
FOR k:=1,2 DO z[k] := rp*a;

COMMENT glavni program;
     polje[i] := 12;
     p(polje[i],i,polje);
END
FINISH
```

Ko se pokliče procedura p, je bila vrednost parametra a inicializirana na 12 in k v proceduri p se je nastavil na 1 (na vrednost dejanskega parametra) v glavnem programu. Ko se pokliče rp

prvič, ima i vrednost 1 in rezultat, ki se priredi rp, je 3.0. Spremenljivki z [1] se priredi vrednost 3.0*12.0 in polje [1] postane 36.0. V naslednjem ciklu je i enako 2, rp je 6.0, toda a ima še vedno vrednost 12.0 in polje [2] postane 72.0.

3.22. Parametri niznih in razdelilniških procedur

Kadar je formalni parameter niz, je dejanski parameter lahko niz znakov med dvojnima narekovajema ali ime niza.

Uporaba razdelilniških (stikalnih) parametrov pa je tale: dejanski parameter je ime razdelilnika. Če se to ime uporablja v procedurnem telesu (z indeksom), je učinek enak onemu pri uporabi v bloku, v katerem je bil razdelilnik deklariran.

3.23. Označitve in procedure kot parametri

Prevajalniška obdelava parametrov tipa LABEL, PROCEDURE, REAL PROCEDURE, INTEGER PROCEDURE in BOOLEAN PROCEDURE je podobna že obravnavanim primerom. Dejanskemu parametru, ki je označitveni izraz, mora predhoditi tipska specifikacija (zaradi enoprehodnega prevajalnika). Podobno kot pri spremenljivkah, ki so klicane z imenom, pri poljih in razdelilnikih, se posreduje proceduri naslov označitve ali procedure. Dejanski parameter mora biti v območju vidljivosti v klicni točki, ni pa nujno, da je v območju klicane procedure; njegova uporaba ga dejansko postavi v območje.

Predddeklarirane funkcije in vhodna/izhodna imena se ne morejo uporabljati kot procedurni parametri, ko je bil formalni parameter deklariran kot procedura. Uporabiti se mora prazna procedura, ki pokliče preddeklarirano. Predddeklarirane procedure pa se lahko uporabljajo v izrazih, kjer je formalni parameter vrednostni. Tako je npr. dovoljeno sin(cos(3)).

RML algol ne razlikuje klicev z imenom in vrednostjo, če so parametri razdelilniki, nizi, označitve in procedure. Če je dejanski parameter označitveni izraz, se vrednost izračuna le ob vstopu v proceduro in ne pri vsakokratni uporabi parametra v procedurnem telesu. Klici vseh teh parametrov so klici z vrednostjo, čeprav prevajalnik ne zahteva njihove specifikacije.

Imejmo primer za dva procedurna začetka:

```
PROCEDURE p(s, ozn, rp, niz);
SWITCH s; STRING niz; LABEL ozn;
REAL PROCEDURE rp;

REAL PROCEDURE x(y, niz); VALUE y;
REAL y; STRING niz;
```

Možen poziv je npr.

```
p(razd, LABEL IF bi THEN ozn1
  ELSE ozn2, REAL PROCEDURE x, "ike")
```

V tem primeru je bil uporabljen označitveni izraz kot dejanski parameter tipa LABEL. Kot pri poljih in razdelilnikih se tudi pri realnih procedurah kot parametrih uporabljajo samo njihova imena. Vendar se parametri parametričnih procedur navedejo pri dejanskih pozivih. Npr.:

```
rp(rp(4, "xyz"), niz1)
```

3.24. Povzetek procedurnih značilnosti

Pri vstopu v proceduro se v Algolu dodeli pomnilni prostor dinamično proceduri za njene podatke skladno z deklaracijami. To pomeni, da so procedure v Algolu po naravi rekurzivne in je rekurzijska globina omejena le z razpoložljivim pomnilnim prostorom.

Procedurno telo je stavek, ki je lahko prazen, enostaven, sestavljen ali blok. V okviru bloka se lahko pojavijo še druge procedurne deklaracije, tako da je veljaven tale primer:

```
PROCEDURE ike;
BEGIN
   PROCEDURE ant;
   BEGIN
      PROCEDURE pet;
      BEGIN
         s1; . . .
      END;
      s2; . . .
   END;
   s3; . . .
END;
```

Območja procedur so določena z navadnimi pravili vidljivosti, tako da stavka s1 in s2 lahko dodelimo k ike, ant in pet, stavek s3 pa samo k ike in ant. Stavki v proceduri lahko navajajo poljubno spremenljivko, ki je v območju vidljivosti in ne samo tiste, ki se pojavljajo v listi procedurnih parametrov. V Fortranu bi bil potreben COMMON stavek. Iz procedure je mogoče tudi skakati z uporabo GOTO stavka na poljubno označitev, ki je v območju vidljivosti.

Slovstvo k prvemu delu

((1)) J.W.Backus et All: Revised Report on the Algorithmic Language ALGOL 60. (Edited by P. Naur). International Federation for Information Processing (IFIP) 1962.

((2)) M.I.Ageev, V.P.Alik, R.M.Galis: Algoritmy (1 - 50), Vypusk 2. Vyčislitel´nyj centr AN SSSR, Moskva 1966.

((3)) M.I.Ageev, V.P.Alik, Ju.I.Markov: Algoritmy (51 - 100), Vypusk 3. Vyčislitel´nyj centr AN SSSR, Moskva 1966.

((4)) M.I.Ageev, L.S.Krivonos, Ju.I.Markov: Algoritmy (101 - 150), Vypusk 4. Vyčislitel´nyj centr AN SSSR, Moskva 1967.

((5)) G.Walter: Strukturierte Programmierung mit ALGOL 60. R.Oldenbourg Verlag, Muenchen 1977.

((6)) Algol 60 for 280 Based Computers under CP/M. Research Machines, P.O.Box 75, Chapel Street, Oxford, England (0865) 49792.

((7)) R.Herschel: Anleitung zum praktischen Gebrauch von ALGOL 60. R.Oldenbourg Verlag, Muenchen 1971.

((8)) D.D.McCracken: A Guide to Algol Programming. John Wiley & Sons, New York.

((9)) E.Kaucher, R.Klatte, Ch.Ullrich: Hoehere Programmiersprachen ALGOL, FORTRAN, PASCAL. B.I.-Wissenschaftsverlag, Zuerich 1978.

((10)) R.Baumann: ALGOL-Manual der ALCOR-Gruppe. R.Oldenbourg Verlag, Muenchen 1969.

((11)) W.Heinrich, W.Stucky: Programmierung mit ALGOL 60. B.G.Teubner Studienskripten, Stuttgart 1978.

DODATEK 1. SINTAKSA JEZIKA RML ALGOL, KI ODSTOPA OD SINTAKSE JEZIKA ALGOL 60. SEMANTIKA VHODNIH IN IZHODNIH FUNKCIJ JE GLEDE NA ALGOL 60 V CELOTI SPREMENJENA.

```
========== SESTAVLJENI STAVKI IN BLOKI ==========

------------------- Programi --------------------

program ::= block FINISH |
                    compound_statement FINISH

---------- Bloki in sestavljeni stavki ----------

block ::= unlabelled_block | label : block
compound_statement ::= unlabelled_compound |
                    label : compound_statement
unlabelled_block ::= block_head ; compound_tail
unlabelled_compound ::= BEGIN compound_tail
block_head ::= BEGIN declaration |
                     block_head ; declaration
compound_tail ::= statement END |
                statement END comment_text ; |
                     statement ; compound_tail
statement ::= unconditional_statement |
          conditional_statement | for_statement
unconditional_statement ::= basic_statement |
                    compound_statement | block
basic_statement ::=
                unlabelled_basic_statement |
                     label : basic_statement
unlabelled_basic_statement ::=
       assignment_statement | go_to_statement |
          dummy_statement | procedure_statement


================ OSNOVNI STAVKI ================

---------------- Pogojni stavki ----------------

conditional_statement ::= if_statement |
                  if_statement ELSE statement |
                     if_clause for_statement |
                  label : conditional_statement
if_statement ::=
              if_clause unconditional_statement
unconditional_statement ::= basic_statement |
                    compound_statement | block
if_clause ::= IF Boolean_expression THEN

----------------- Zanemi stavki ----------------

for_statement ::= for_clause statement |
                         label : for_statement
for_clause ::= FOR variable := for_list DO
for_list ::= for_list_element |
                   for_list , for_list_element
for_list_element ::= arithmetic_expression |
                   arithmetic_expression STEP
                   arithmetic_expression UNTIL
                   arithmetic_expression |
                   arithmetic_expression WHILE
                            Boolean_expression

------------- Prireditveni stavki -------------

assignment_statement ::=
         left_part arithmetic_expression |
                   left_part Boolean_expression
left_part ::= variable := |
                        procedure_identifier :=

---------------- Skocni stavki ----------------

go_to_statement ::=
                   GOTO designational_expression

---------------- Prazni stavki ----------------

dummy_statement ::= empty


------------- Procedurni stavki --------------

procedure_statement ::=
      procedure_identifier actual_parameter_part
actual_parameter_part ::= empty |
                          ( actual_parameter_list )
actual_parameter_list ::= actual_parameter |
       actual_parameter_list parameter_delimiter
                                actual_parameter
parameter_delimiter ::= , | ) letter_string :(
letter_string ::= letter | letter_string letter
actual_parameter ::= string | expression |
          array_identifier | switch_identifier |
                               procedure_identifier
procedure_identifier ::= identifier |
                        designational_expression


================ DEKLARACIJE ================

declaration ::= type_declaration |
        array_declaration | switch_declaration |
                           procedure_declaration

------------- Tipske deklaracije -------------

type_declaration ::= type type_list
type ::= INTEGER | REAL | BOOLEAN
type_list ::= simple_variable |
                      simple_variable , type_list

------------- Deklaracije polj ---------------

array_declaration ::= ARRAY array_list |
                            type ARRAY array_list |
                              BYTE ARRAY array_list
array_list ::= array_segment |
                       array_list , array_segment
array_segment ::=
          array_identifier [ bound_pair_list ] |
                array_identifier , array_segment
bound_pair_list ::= bound_pair |
                    bound_pair_list , bound_pair
bound_pair ::= lower_bound : upper_bound
lower_bound ::= arithmetic_expression
upper_bound ::= arithmetic_expression

---------- Razdelilniske deklaracije ----------

switch_declaration ::=
         SWITCH switch_identifier := switch_list
switch_list ::= designational_expression |
          switch_list , designational_expression

----------- Procedurne deklaracije ------------

procedure_declaration ::=
   PROCEDURE procedure_heading procedure_body |
   type PROCEDURE
                procedure_heading procedure_body
procedure_body ::= statement | code
procedure_heading ::=
   procedure_identifier formal_parameter_part ;
                   value_part specification_part
specification_part ::= empty |
                  specifier identifier_list ; |
                   specification_part specifier
                                identifier_list ;
specifier ::= STRING | type | ARRAY |
      type ARRAY | LABEL | SWITCH | PROCEDURE |
                                   type PROCEDURE
value_part ::= VALUE identifier_list ; | empty
identifier_list ::= identifier |
                    identifier_list , identifier
formal_parameter_part ::= empty |
                        ( formal_parameter_list )
```

```
formal_parameter_list ::= formal_parameter |
      formal_parameter_list parameter_delimiter
                                formal_parameter
formal_parameter ::= identifier


==================== IZRAZI ====================

------------- Oznacitveni izrazi -------------

designational_expression ::=
              simple_designational_expression |
       if_clause simple_designational_expression
                    ELSE designational_expression
simple_designational_expression ::= label |
                              switch_designator |
                          designational_expression
switch_designator ::=
     switch_identifier [ subscript_expression ]
switch_identifier ::= identifier
label ::= identifier

-------------- Boolovski izrazi ---------------

Boolean_expression ::= simple_Boolean |
                       if_clause simple_Boolean
                        ELSE Boolean_expression
simple_Boolean ::= implication |
          simple_Boolean EQUIVALENT implication
implication ::= Boolean_term |
                implication IMPLIES Boolean_term
Boolean_term ::= Boolean_factor |
                  Boolean_term OR Boolean_factor
Boolean_factor ::= Boolean_secondary |
            Boolean_factor AND Boolean_secondary
Boolean_secondary ::= Boolean_primary |
                                NOT Boolean_primary
Boolean_primary ::= logical_value | variable |
                function_designator | relation |
                          ( Boolean_expression )
relation ::=          simple_arithmetic_expression
                           relational_operator
                    simple_arithmetic_expression
relational_operator ::= < | <= | = | >= | > | #

------------- Aritmeticni izrazi --------------

arithmetic_expression ::=
                  simple_arithmetic_expression |
          if_clause simple_arithmetic_expression
                      ELSE arithmetic_expression
if_clause ::= IF Boolean_expression THEN
simple_arithmetic_expression ::= term |
                           adding_operator term |
                     simple_arithmetic_expression
                              adding_operator term
term ::= factor |
                term multiplying_operator factor
factor ::= primary | factor ^ primary
primary ::= unsigned_number | variable |
                           function_designator |
                      ( arithmetic-expression )
multiplying_operator ::=  * | / | % | MOD
adding_operator ::=  + | - | MASK | DIFFER | !

---------- Funkcijski oznacevalniki -----------

function_designator ::=
     procedure_identifier actual_parameter_part
actual_parameter_part ::= empty |
                      ( actual_parameter_list )
actual_parameter_list ::= actual_parameter |
      actual_parameter_list parameter_delimiter
                                actual_parameter
parameter_delimiter ::= , | ) letter_string :(
letter_string ::= letter | letter_string letter
actual_parameter ::= string | expression |
          array_identifier | switch_identifier |
                           procedure_identifier
procedure_identifier ::= identifier

---------------- Spremenljivke ----------------

variable ::= simple_variable |
                                subscripted_variable
subscripted_variable ::=
            array_identifier [ subscript_list ]
array_identifier ::= identifier
subscript_list ::= subscript_expression |
          subscript_list , subscript_expression
subscript_expression ::= arithmetic_expression
simple_variable ::= variable_identifier
variable_identifier ::= identifier


=== SIMBOLI, IDENTIFIKATORJI, STEVILA, NIZI ===

-------------------- Nizi --------------------

string ::= " open-string "

------------------ Stevila -------------------

number ::= unsigned_number |
           + unsigned_number | - unsigned_number
unsigned_number ::= decimal_number |
                                  exponent_part |
                          decimal_number exponent_part
decimal_number ::= unsigned-integer |
                                 decimal_fraction |
                unsigned_integer decimal_fraction
exponent_part ::= E integer | e integer
decimal_fraction ::= . unsigned_integer
integer ::= unsigned_integer |
                               + unsigned_integer |
                                 - unsigned_integer
unsigned_integer ::= digit |
                             unsigned_integer digit

--------------- Identifikatorji ----------------

identifier ::= letter | identifier letter |
                                   identifier digit

(Samo prvih 6 znakov v identifikatorju je
bistvenih.)

---------------- Omejevalniki ------------------

specificator ::= STRING | LABEL | VALUE
declarator ::= BOOLEAN | INTEGER | REAL |
                    BYTE | ARRAY | SWITCH | PROCEDURE
bracket ::=  ( | ) | [ | ] | { | } | " |
                                        BEGIN | END
separator ::=  , | . | E | e | : | ; | := |  |
                   STEP | UNTIL | WHILE | COMMENT
sequential_operator ::= GOTO | IF | THEN |
                                  ELSE | FOR | DO
logical_operator ::= EQUIVALENT | IMPLIES |
             OR | AND | NOT | MASK | DIFFER | !
relational_operator ::=  < | <= | = | >= |
                                                > | #
arithmetic_operator ::=  + | - | * | / | % |
                                              ^ | MOD
operator ::= aritmetic_operator |
      relational_operator | logical_operator |
                              sequential_operator
delimiter ::= operator | separator | bracket |
                        declarator | specificator


----------------- Komentarji -------------------

comment ::= COMMENT any_sequence_not_
                        containing_semicolon ; |
                         END this_is_ignored ; |
                           END so_is_this ELSE |
                      END and_this_also FINISH |
                    { this_(comment)_is_ignored }

------ Crke, stevilke, logicne vrednosti ------

logical_value ::= TRUE | FALSE
digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
letter ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|
                                  s|t|u|v|w|x|y|z|
           A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|
                                  S|T|U|V|W|X|Y|Z
```

DODATEK 2. VHODNE/IZHODNE PROCEDURE JEZIKA RML ALGOL

========== VHODNE/IZHODNE PROCEDURE ==========

---------- Procedure ioc(n) ----------

n	ioc(n)
0 - 5	input/output selection
6 - 12	rwrite format control
13 - 15	output file options
16 - 17	interrupt option on disk I/O
18 - 19	read options
20 - 21	file extension options
22	reboot CP/M on completion
23 - 59	linked to procedures in ALIB.ALG
60	rerun program from start
61 - 62	RML graphics

---------- Standardne funkcije ----------

sin(x)	x is in radians
cos(x)	x is in radians
arctan(x)	the result is in radians in the range -pi/2 to +pi/2
ln(x)	natural logarithm
exp(x)	e to the power x
sqrt(x)	square root of x
abs(x)	absolute value of x
sign(x)	delivers -1, 0, or +1 according to whether x is negative, zero, or positive
entier(x)	returns the larger integer less or equal to x
x	is called by value and thus the actual parameter may be an expression

---------- Vhodne/izhodne procedure ----------

chin(dev)	read a char from device
chout(dev,ival)	outputs byte to dev
read(dev)	read a number from dev
read(dev,label)	read a file of unknown length
rwrite(dev,val,a,b)	outputs val to dev using a,b format
rwrite(dev,val)	exponent format with 6 dec digits
skip(dev)	outputs a CR/LF to device
text(dev,"string")	outputs a string to device
write(dev,ival)	prints ival as an integer to dev in decimal
write(dev,ival,radix)	prints in radix (0,1,2 for decimal, octal, hexadecimal)
findinput("string")	opens a file or device defined in string for input on stream device
findoutput("string")	analogous to findinput but for output
rblock(dev,a,b,n)	read n blocks from a disk file associated with stream dev, starting at block number b, writing the contents in memory at address a
wblock(dev,a,b,n)	writes n blocks to disk; parameters are the same as for rblock; value of wblock: = 0 successful write = 1 error in extending file = 2 end of disk file = 3 hard error = 255 no more directory space available
rewind(dev)	dev file is rewound for reading from the beginning
seti(a)	set the INPUT pointer to the address a
seto	set the OUTPUT pointer to the address a
ipoint	returns the current address of the INPUT pointer
opoint	returns the current address of the OUTPUT pointer
exflt(a,t)	extend the file control block list; a address of buffer to use t = 0 serial file access t # 0 random file access
fcblock(dev)	returns the address of the file control block associated with the file stream device
swlist	returns the address of the switch list
bios(n,bc)	performs a call through the BIOS jump vector where: n = entry in the jump table (0-14) bc = contets of BC register on entry bios returns contents of A register on exit
cpm(c,de)	performs a call to CP/M where c = C register on entry (0-27) de = DE register on entry; cpm returns contents of A register
rename	renames a file; if rename = -1 file was not found
newext(j,"XYZ")	a file of stream j is closed and its extension changed to XYZ

---------- Napaka v casu izvajanja ----------

error

---------- Knjiznicne procedure ----------

fspace	returns the number of bytes free
blmove(s,f,len)	block move of len bytes starting at address s to the block starting at address f
peek(a)	returns the byte value contained within the address given by a
poke(a,i)	sets the contents of address a to value i
in(c)	input from a port; executes an IN A,(C) instruction
out(c,a)	output to a port; executes an OUT (C),A instruction
dpb(u,t,s,a)	set up the disk parameters, u = unit number (0 to 3), t = track, s = sector, a = DMA address
rdisk	read the disk directly after a previous dpb call, returns the result
wdisk	write to disk using information set up by a previous call to dpb returning the result
parity(i)	returns TRUE if the character value of i has even parity else FALSE
shl(v,n)	shift integer v n places left
lsr(v,n)	logical shift right
asr(v,n)	arithmetic shift right
rotl(v,n)	rotate left
rotr(v,n)	rotate right
random	returns a pseudorandom number in the range 0 to 1

```
clarr(a,len)
            clear array area of length len  by-
            tes starting at address a
sloc("string")
            returns the address of the start of
            the string (string parameter)
tlen(s)     returns the length of  the  string
            whose address is at s
smatch(long,short)
            compares   two  strings  (short  in
            long); parameters are the addresses
            of the strings; if a match is found
            this  function returns the  address
            otherwise a zero
```

```
atext(dev,s)
            similar  to 'text' but s is the ad-
            dress of the string
emt(n)      graphic instruction
wait(n)     delay with n (for 10*n msec)
chpos(x,y)  coordinate position on the screen
point(x,y,z)
            plots a point on the screen of  in-
            tensity z
line(x1,y1,x2,y2,z)
            draws  a line from position (x1,y1)
            to position (x2,y2) of intensity z
```

DODATEK 3. PRIMERI ALGOLSKIH PROGRAMOV

```
BEGIN
COMMENT izpis sinusoide z vodoravno osjo;

   COMMENT deklaracija procedure za izpis znaka
           na poljubnem polozaju;
   PROCEDURE izpisi(zac,kon,znak,vmes,crlf);
      VALUE zac,kon,crlf;
      INTEGER zac,kon,crlf;
      STRING znak,vmes;
   BEGIN
      INTEGER tek;
      FOR tek:=zac STEP 1 UNTIL kon DO
          text(1,vmes);
      text(1,znak);
      IF crlf)0 THEN skip(1);
   END
   konec procedurne deklaracije;

   COMMENT glavni program;
   INTEGER tek;
   FOR tek:=99 STEP -11 UNTIL -99 DO
      IF tek=0 THEN
         BEGIN
         COMMENT izpis abscise in znakov;
         izpisi(1,1,"**"," ",0);
         izpisi(3,32,"**","-",0);
         izpisi(34,63,"**","-",1);
         END
      ELSE
         BEGIN
         REAL arcsin,y;
         INTEGER pol;
         y:=tek*0.01;
         arcsin:=arctan(y/sqrt(1-y*y));
         IF tek(0 THEN
            BEGIN
            pol:=-entier(arcsin*10);
            izpisi(1,pol+31,"**"," ",0);
            izpisi(pol+33,64-pol,"**"," ",1);
            END
         ELSE
            BEGIN
            pol:=entier(arcsin*10)+2;
            izpisi(1,pol-1,"**"," ",0);
            izpisi(pol,33-pol,"**"," ",1);
            END
         END
END
FINISH
```

```
BEGIN
   COMMENT izpis sinusoide z navpicno srednjo
         crto (osjo);

   COMMENT deklaracija procedure za izpis znaka
         na poljubnem polozaju;
   PROCEDURE izpis(zac,pol,znak,crlf);
      VALUE zac,pol,crlf;
      INTEGER zac,pol,crlf;
      STRING znak;
         BEGIN
            INTEGER tek,konec;
            konec:=pol-1;
            FOR tek:=zac STEP 1 UNTIL konec DO
                text(1," ");
            text(1,znak);
            IF crlf)0 THEN skip(1);
         END
   konec procedurne deklaracije;

   COMMENT glavni program;
   INTEGER korak;
   REAL korobs;
   korobs:=2*3.14/40;
   FOR korak:=0 STEP 1 UNTIL 40 DO
      BEGIN
        INTEGER tiskpl;
        REAL x;
        x:=korak*korobs;
        tiskpl:=entier(sin(x)*30)+40.5;
        IF tiskpl)40
        THEN BEGIN
               izpis(1,40,"!",0);
               izpis(41,tiskpl,"**",1);
             END
        ELSE
           IF tiskpl=40
           THEN izpis(1,40,"**",1)
           ELSE
              BEGIN
                 izpis(1,tiskpl,"**",0);
                 izpis(tiskpl+1,40,"!",1);
              END
      END
END
FINISH
```

INFORMATICA 4/1983

ULOGA RELACIJE APSORBCIJE U ALGORITMIMA INDUKCIJE I DEDUKCIJE

D. MARTINOVIĆ

MATEMATIČKI INSTITUT, BEOGRAD

UDK: 510.5:681.3

Relacija apsorbcije igra veliku ulogu u algoritmima koji se koriste prilikom mehaničkog dokazivanja teorema i formiranja hipoteza. Ona se koristi i kao relacija parcijalnog uredjenja u mreži $(C/\sim,\leqslant,\sqcap,\sqcup)$. Pokazuje se obavljanje rezolucije u navedenoj mreži, odnos izmedju neredukovanih i sastavaka sa faktorima, itd.

SUBSUMPTION RELATION IN MECHANICAL THEOREM PROVING AND HYPOTHESIS FORMATION. Subsumption relation plays a great role in the algorithms used in mechanical theorem proving and hypothesis formation. It is also a relation of partial order in the lattice $(C/\sim,\leqslant,\sqcap,\sqcup)$. It is showen how $\leqslant$ can be used in the resolution, what is the relation between nonreduced clauses and clauses with factors, etc.

Kada se na računaru obavljaju logička indukcija i dedukcija nad skupom formula Predikatskog računa I reda, uobičajeno je da se taj skup formula ekvivalentnim transformacijama prevede u skup C sastavaka-formula bez kvantifikatora, koje su disjunkcije konačnog broja literala, pa da se pomoću njih rešava zadati problem. Na sastavke iz dobijenog skupa se zatim, obično, primenjuju čisto sintaksna pravila izvodjenja, koja se na računaru jednostavno realizuju, ali koja imaju jako semantičko značenje.

Kako se sve ove operacije vrše nad skupom sastavaka, prirodno je posmatrati algebarsku strukturu skupa svih sastavaka nastalih u azbuci koju čine bar dva relacijska simbola, imena konstanti, promenljivih i funkcija. Po konvenciji, sastavke prikazujemo kao skupove literala.

Ne ulazeći u mukotrpno, ali dalekosežno ispitivanje algebarskih osobina ovog skupa, navodimo samo rezultat, uz neophodna objašnjenja i definicije: $(C/\sim,\leqslant,\sqcap,\sqcup)$ je mreža.

Relacija uredjenja u mreži je relacija apsorbcije (subsumption), koja se definiše na sledeći način:

Def.1.: $C_1 \leqslant C_2$ akko $(\exists\theta)\ C_1\theta \subseteq C_2$.//
Ukoliko su ovi sastavci jednoelementni (literali) umesto relacije podskup, stoji jednakost.

Ova binarna relacija je odlučiva, refleksivna i tranzitivna, pa je relacija kvazi uredjenja. Medjutim, ona nije antisimetrična, jer iz $C_1 \leqslant C_2$ i $C_2 \leqslant C_1$ ne sledi ni da su C_1 i C_2 varijante. To je zato što postoje tzv. neredukovani sastavci koji apsorbuju neki svoj pravi podskup. Na svu sreću, zahvaljujući odlučivosti relacije apsorbcije, može se formirati konačni algoritam redukcije, koji za svaki sastavak utvrdjuje da li je redukovan ili nije, pa ako nije, vrši njegovu redukciju. Kada u skup C sastavaka uvedemo ekvivalenciju na sledeći način $C_1 \sim C_2 \Leftrightarrow C_1 \leqslant C_2 \wedge C_2 \leqslant C_1$, ceo skup C se raspada na klase ekvivalencije, a kako su ekvivalentni redukovani sastavci medjusobno varijante, to za predstavnike ovih klasa uzimamo samo redukovane sastavke.

U skup $C/_\sim$ se uvodi uredjenje na sledeći način:

$$[C_1] \leqslant [C_2] \overset{\text{def}}{\Longleftrightarrow} (\exists C_1)(\forall C_2)(C_1 \in [C_1] \wedge C_2 \in [C_2] \Rightarrow C_1 \leqslant C_2)$$

i lako se pokazuje da je tek ovako definisana relacija, relacija parcijalnog uredjenja.

Pokažimo sada kako je uvedena operacija $\sqcap$:

Def.2: Najmanje opštom generalizacijom (least general generalisation) sastavaka C_1 i C_2, u oznaci $C_1 \sqcap C_2$, nazivamo sastavak koji zadovoljava sledeće uslove (1) $C_1 \sqcap C_2 \leqslant C_1 \wedge C_1 \sqcap C_2 \leqslant C_2$ i (2) za svako D iz $\mathcal{C}$ za koje važi $D \leqslant C_1 \wedge D \leqslant C_2$ važi i $D \leqslant C_1 \sqcap C_2$. //

Nećemo, zbog dužine, navoditi algoritam generalizacije skupa sastavaka (može se naći u (2)), recimo samo da svaki konačan skup $\{C_1,\dots,C_n\}$ sastavaka ima generalizaciju akko ima bar jednu selekciju $L_1,\dots,L_n$, gde su $L_i \in C_i$ literali sa istim znakom i predikatskim simbolom.

U skup $\mathcal{C}/_\sim$ se uvodi operacija generalizacije na sledeći način: $[C_1] \sqcap [C_2] = [C_1 \sqcap C_2]$.

Sve do ovog stupnja, uspešno su se koristila rešenja vrlo slična onima koja su se odnosila na skupove atoma, ali kada se dodje do koraka u kome je potrebno pronaći operaciju dualnu sa $\sqcap$, analogije sa skupom atoma prestaju. Tamo je korišćena operacija najopštije unifikacije (most general unification), za koju nema sličnog rešenja u skupu $\mathcal{C}(\mathcal{C}/_\sim)$. No, kako se relacija apsorbcije u nekim slučajevima svodi na relaciju "biti podskup", prirodno je iskoristiti skupovno prikazivanje sastavaka i dati sledeću definiciju:

Def.3: Najopštiji zajednički slučaj sastavaka C_1 i C_2, u oznaci $C_1 \sqcup C_2$, je sastavak $C_1 \cup C_2'$, gde je C_2' varijanta od C_2 koja sa C_1 nema zajedničkih promenljivih.//

Već je jasno da se proširenje ove operacije na skup $\mathcal{C}/_\sim$ uvodi ovako: $[C_1] \sqcup [C_2] = [C_1 \sqcup C_2]$.

Ovim je bar delimično pokazano, kako je formirana mreža $(\mathcal{C}/_\sim, \leqslant, \sqcap, \sqcup)$ koja je, recimo i to, nemodularna i nekomplementirana.

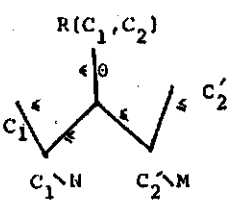
Polumreža $(\mathcal{C}/_\sim, \leqslant, \sqcap)$ je konstruisana radi potreba obavljanja indukcije na računarima. Polazeći od zadatog skupa S sastavaka, primenom algoritma generalizacije, formiraju se hipoteze zasnovane na S.

Dualna operacija $\sqcup$ je uvedena najviše zato da bi se zaokružile algebarske osobine polumreže $(\mathcal{C}/_\sim, \leqslant, \sqcap)$, ali je interesantno da ona može da nadje svoje mesto u procedurama koje se primenjuju prilikom dokazivanja teorema na računaru.

Ovde ćemo se pozabaviti samo rezolucijom, kao jednom od metoda primenjivanih prilikom mehaničkog dokazivanja teorema.

Def.4: Binarnom rezolventom sastavaka C_1 i C_2, nazivamo sastavak $((C_1 \setminus N) \cup (C_2' \setminus M))\theta$, gde je C_2' varijanta od C_2 takva da sa C_1 nema zajedničkih promenljivih, a N i $\neg$ M su medjusobno unifabilni literali sa najopštijim unifikatorom θ.//

Odmah se vidi da je po sredi uniranje "okrnjenih" polaznih sastavaka bez zajedničkih promenljivih, a to je upravo operacija uniranja sastavaka u mreži. Dalje, rezolucijom se uklanjaju komplementarni parovi literala, jer oni u uniji čine logičku jedinicu. Napomenimo, da su prilikom



konstruisanja mreže sastavaka logička pravila bila potisnuta od strane algebarskih, ali su implicitno bila prisutna. Tako, recimo, iz $C_1 \leqslant C_2 \rightarrow C_1 \Rightarrow C_2$, a rezolventa je posledica sastavaka iz kojih je nastala.

Posmatraćemo, zato, podskup od $\mathcal{C}$, koji ćemo označiti sa **1**, koji čine sastavci koji su disjunkcije dva komplementarna literala. Nazvali smo ga jediničnim akupom, jer pretstavlja skup logičkih jedinica. Tada možemo reći sledeće:
$(C_1 \cup C_2)\theta = R(C_1,C_2) \cup 1$, $1 \in \mathbf{1}$. Štaviše, možemo posmatrati i sastavke označene kao 1', koji primenom mgu atoma koje sadrže, postaju sastavci iz **1**.

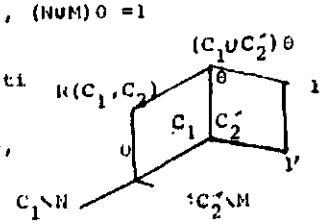
Primer: $\Gamma=\{L(a,f(b)), L(x,y), \neg L(a,f(z))\}$,
$\theta=\{(a,x),(f(b),y),(b,z)\}, \theta=mgu\{L(a,f(b)), L(x,y), L(a,f(z))\}$, $\Gamma\theta=1=\{L(a,f(b)), \neg L(a,f(b))\}$

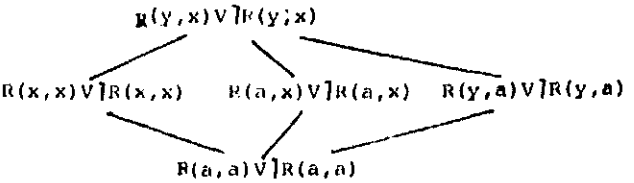
Tako odnose izmedju sastavaka koji učestvuju u rezoluciji možemo prikazati na sledeći način:

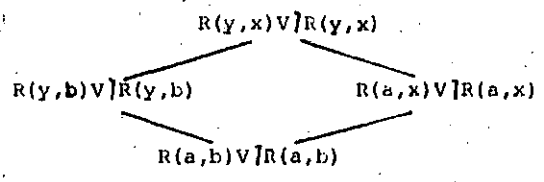
$C_1 \cup C_2' = (C_1 \setminus N) \cup (C_2' \setminus M) \cup 1'$, $N \cup M = 1'$

$(C_1 \cup C_2')\theta = R(C_1,C_2) \cup 1$, $(N \cup M)\theta = 1$

Skup 1 se može posmatrati i kao skup valjanih formula odredjene vrste, takvih da su neke od njih medjusobno u relaciji, a neke ne, kao što se vidi iz sledećeg primera:







Skup **1** je zatvoren u odnosu na primenu supstitucije -sa svakim sastavkom iz **1**, skupu **1** pripadaju i svi slučajevi tog sastavka. To je sasvim razumljivo, jer iz $C_1\theta=C_2 \wedge C_1 \Rightarrow C_2$, dakle i C_2 je valjana formula, a kako je sastavljena od dva literala različito označena (to se primenom supstitucije ne može promeniti), onda je i $C_2 \in$**1**. Ovaj skup je takodje zatvoren u odnosu na operaciju generalizacije, ukoliko sadrži samo jedan relacijski simbol, jer onda svaki njegov podskup ima selekciju, a iz algoritma generalizacije se vidi da se kao rezultat dobija takodje sastavak iz **1**.

Osim na navedeni način, rezolucija se može definisati i ovako:

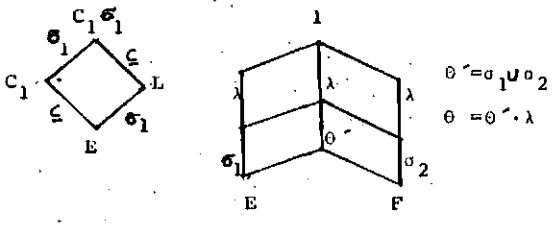
Def.5: Ako je E unifabilan podskup od C sa σ kao svojim mgu, onda je $C\sigma$ faktor od C. //

Def.6: Rezolventom sastavaka C_1 i C_2 nazivamo
1) binarnu rezolventu ovih sastavaka,
2) -"- faktora od C_1 i sastavka C_2,
3) -"- sastavka C_1 i faktora od C_2,
4) -"- faktora od C_1 i faktora od C_2.//

Ovde vidimo da se rezolucijom možemo osloboditi čitavog skupa literala (a ne samo dva literala kao do sada) iz unije dva sastavka.

Primer: $E\subseteq C_1$, $\sigma_1=mgu E$; $F\subseteq C_2, \sigma_2=mgu F$;

$\lambda=mgu(E\sigma_1, \neg F\sigma_2)$, $R(C_1,C_2)=((C_1\sigma_1\setminus E\sigma_1)\cup(C_2\sigma_2\setminus F\sigma_2))\lambda$



Ovde je $E\cup F=1$, jer postoji supstitucija θ koja ih prevodi u 1: $(E\sigma_1\lambda, \neg F\sigma_2\lambda)=(E\theta, \neg F\theta)$ gde je $\theta=(\sigma_1\cup\sigma_2)\lambda$ uz pretpostavku da C_1 i C_2 nemaju zajedničkih promenljivih. Da bi se mogla izvršiti rezolucija tipa 4), moraju $E\sigma_1$ i $\neg F\sigma_2$ biti unifabilni skupovi, tj. mora postojati λ , takva da je $E\sigma_1\cdot\lambda=\neg F\sigma_2\cdot\lambda$, što se grafički može prikazati na navedeni način.

Kako se u definicijama redukovanih i faktorisanih sastavaka posmatraju sastavci i njihovi podskupovi, interesantno je videti kakav odnos postoji izmedju pojmova "biti redukovan" i "imati faktore".

TVE.1: Ako sastavak C nema faktore, onda je redukovan. //

Dokaz: Ovde ćemo dokazati obrnuto tvrdjenje, tj. da C ima faktore ako nije redukovan.

Iz $E\subsetneq C$ i $C\sigma E \Rightarrow (\exists\theta)$ $C\theta\subseteq E$, tj. da θ preslikava neke podskupove od C u elemente iz C. Neka je $\{L_1,\ldots,L_n,M_1,\ldots,M_m\}\subseteq C$ i $\{M_1,\ldots,M_m\}\subseteq E$ i za neko θ, $\{L_1,\ldots,M_m\}\theta\subseteq\{M_1,\ldots,M_m\}$. Tada zbog $n+m>m$ što je ispunjeno kada je E pravi podskup od C, postoji bar jedan niz indeksa $\{i_1,\ldots,i_k,j\}$, $\{i_1,\ldots,i_k\}\subseteq\{1,\ldots,n+m\}$ i $j\in\{1,\ldots,m\}$, takav da je $N\theta = M_j$, gde je N neki k-elementni podskup od C. No, tada je N unifabilan skup sa θ kao svojim mgu, pa je $C\theta$ faktor od C.

Ako postoji supstitucija θ koja nije jedinična, takva da preslikava sastavak C na sebe samoga, isti zaključak važi: tada mora postojati podskup $\{N_1,\ldots,N_k\}$ od C, takav da je $N_1\theta = N_2, N_2\theta = N_3, \ldots, N_k\theta = N_1$, dok su ostali literali u C posle primene θ ostali nepokretni. No, odavde zaključujemo da je $N_1\subseteq\ldots\subseteq N_k\subseteq N_1$, tj. $N_1\sim\ldots\sim N_k$ tj. da su ovi literali alfabetske varijante, a u tom slučaju se bar svaka dva od njih mogu unifikovati, pa postoji faktor od C.
Primer: $C=\{P(x,y),P(y,z),P(x,z)\}$ postoji supstitucija $\theta=\{(y,x),(z,y),(x,z)\}$ takva da je $C\theta = C$ i $P(x,y)\theta = P(y,z); P(y,z)\theta=P(x,z); P(x,z)\theta=P(x,y)$. Ovde je čak ceo C unifabilan; recimo sa $\sigma=\{(z,x),(z,y)\}$, $C\sigma=\{P(z,z)\}$.

Znači da pod navedenim pretpostavkama uvek postoji unifabilan podskup od C, pa dakle i faktor od C. Pokazali smo da je osobina posedovanja faktora, mnogo šira od osobine "biti neredukovan".

U prethodno navedenoj definiciji rezolucije, se pod 1) govori o nalaženju rezolvente faktorisanog sastavka, ali se ne kaže koji faktor treba uzeti. Naime, sastavak u opštem slučaju može imati više faktora, jer ima više unifabilnih skupova, štaviše i kada je reč o samo jednom unifabilnom podskupu, sastavak može imati više faktora, jer su i svi podskupovi unifabilnog skupa takodje unifabilni i mogu imati različite mgu. Zbog toga je važno poznavati odnos koji postoji izmedju različitih rezolventi dva sastavka. Pos-

matraćemo zato jedan od slučajeva koji mogu nastati, jer se i ostali na sličan način mogu obradjivati.

TVE. 2: Neka su $E_1,\dots,E_k$ unifabilni podskupovi od C, takvi da je $E_1 \subsetneq E_2 \subsetneq \dots \subsetneq E_k$, $\sigma_i = mgu(E_i)$, takvi da σ_i ne unifikuje neki nadskup od E_i u C ni za jedno $i \in k$. Neka su $R(C\sigma_i, D)$ binarne rezolvente faktora od C i sastavka D, takve da se iz D uvek otklanja isti literal, a iz $C\sigma_i$ uklanja se $E_i\sigma_i$, tada važi sledeće:
ili je $R(C\sigma_i, D) \leq R(C\sigma_{i-1}, D)$, ili $R(C\sigma_i, D)$ apsorbuje slučaj nekog faktora od $C\sigma_{i-1}$ sa sastavkom D.

Dokaz: Neka je $E_1 \subsetneq \dots \subsetneq E_k$, iz $\sigma_i = mgu(E_i)$ sledi $\sigma_i = u(E_{i-1})$, pa postoji λ_{i-1} takvo da je $\sigma_i = \sigma_{i-1} \cdot \lambda_{i-1}$, $i=1,\dots,k$. Zato je $C\sigma_1 \leq C\sigma_2 \leq \dots \leq C\sigma_k$, a iz $E_{i-1} \subsetneq E_i \Rightarrow E_{i-1}\sigma_i \subsetneq E_i\sigma_i \leftrightarrow E_{i-1}\sigma_i\lambda_{i-1} \subsetneq E_i\sigma_i \leftrightarrow N_{i-1}\lambda_{i-1} = N_i \leftrightarrow N_{i-1} \leq N_i$, $i=1,\dots,k$.

$R(C\sigma_i, D) = ((C\sigma_i \setminus E_i\sigma_i) \cup (D \setminus M))\theta_i$, gde je $\theta_i = mgu(N_i, \neg M)$

$C\sigma_i \setminus E_i\sigma_i = C\sigma_i \setminus E_{i-1}\sigma_i = C\sigma_{i-1}\lambda_{i-1} \setminus E_{i-1}\sigma_{i-1}\lambda_{i-1} \subsetneq (C\sigma_{i-1} \setminus E_{i-1}\sigma_{i-1})\lambda_{i-1}$. Ne umanjujući opštost pretpostavljamo da $C\sigma_i$ i D nemaju zajedničkih promenljivih, zbog čega možemo pisati:

$(C\sigma_i \setminus E_i\sigma_i) \cup (D \setminus M) \leq (C\sigma_{i-1} \setminus E_{i-1}\sigma_{i-1})\lambda_{i-1} \cup (D \setminus M) =$

$= ((C\sigma_{i-1} \setminus E_{i-1}\sigma_{i-1}) \cup (D \setminus M))\lambda_{i-1}$, pa je i

$R(C\sigma_i, D) \leq ((C\sigma_{i-1} \setminus E_{i-1}\sigma_{i-1}) \cup (D \setminus M))\lambda_{i-1}\theta_i$

Nadjimo odnos izmedju θ_i i θ_{i-1}:
$\theta_i = mgu(N_i, \neg M) = mgu(N_{i-1}\lambda_{i-1}, \neg M)$, $\theta_{i-1} = mgu(N_{i-1}, \neg M)$
Neka se N_{i-1} i M mogu prikazati na sledeći način:

$N_{i-1} = F_1 t_1 F_2 \dots F_k t_k F_{k+1}$, $M = F_1 s_1 \dots F_k s_k F_{k+1}$, gde u F_i nema promenljivih iz N_{i-1} i M (jer prema pretpostavci $C\sigma_i$ i D nemaju zajedničkih promenljivih), a parovi (t_i, s_i) su osnov za formiranje skupa razlike, potrebnog za obavljanje algoritma unifikacije. Tako se dobija da je $\theta_{i-1} = ((t_i, s_i) \vee (s_i, t_i))$, zavisno do toga da li je s_i (t_i) ispred t_i (s_i) u alfabetskom smislu). Ovde je, naravno, uvek druga komponenta u paru, ime promenljive.

$N_{i-1}\lambda_{i-1} = F_1 t_1 \lambda_{i-1} F_2 \dots F_k t_k \lambda_{i-1} F_{k+1}$, pa se θ_i može pretstaviti na sledeći način:
$\theta_i = ((t_i\lambda_{i-1}, s_i)/(t_i, s_i) \in \theta_{i-1}) \cup ((s_j, t_j\lambda_{i-1})/t_j\lambda_{i-1})$ je promenljiva i $(s_j, t_j) \in \theta_{i-1}) \cup \Psi$, gde Ψ unifikuje ostale terme oblika $t_i\lambda_{i-1}$ sa s_i. Kako je $(t_j\lambda_{i-1}, t_j) \in \lambda_{i-1}$, kada su i t_j i $t_j\lambda_{i-1}$ promenljive, možemo pisati $(t_j, t_j\lambda_{i-1}) \in \lambda_{i-1}^{-1}$, pa je $(s_j, t_j\lambda_{i-1}) \in \lambda^{-1} \cdot \theta_{i-1}$. Dakle, mogu važiti sledeće relacije:

$1^{\circ}\ \theta_i \subsetneq \theta_{i-1} \cdot \lambda_{i-1}$; $2^{\circ}\ \theta_i \subsetneq \lambda_{i-1}^{-1} \cdot \theta_{i-1}$;

$3^{\circ}\ \theta_i \subsetneq \theta_{i-1} \cdot \lambda_{i-1} \cup \lambda_{i-1} \cdot \theta_{i-1} \cup \Psi$.

U zavisnosti od toga kakvo je θ_i može se reći da:

1) $R(C\sigma_i, D) \leq R(C\sigma_{i-1}, D)$ (slučaj 2°)

2) $R(C\sigma_i, D)$ apsorbuje slučaj rezolvente faktora od $C\sigma_{i-1}$ (jer je λ_{i-1} mgu skupa $(N_{i-1}) \cup (E_i \setminus E_{i-1})$, čime smo dokazali tvrdjenje.

Slično zaključujemo kada dozvolimo da se i u D formiraju različiti faktori medjusobno uredjeni relacijom $\leq$.

Prema tome nije svejedno koji se faktor uzima prilikom obavljanja rezolucije, ali ako se poznaje odnos koji važi izmedju različitih rezolventi dva sastavka, u slučaju da nadjena rezolventa iz nekih razloga ne odgovara primenjenoj proceduri dokazivanja, lako može biti zamenjena nekom drugom rezolventom a da se ne remeti kompletnost i saglasnost postupka.

Takodje se istražuju mogućnosti primene nekih drugih vrsta zamena rezolventi koje se pojavljuju tokom traganja za dokazom. U svim zamenama ključnu ulogu igra relacija apsorbcije, jer se uvek apsorbovani sastavak zamenjuje ovim koji ga apsorbuje. Od toga kako će se i u kojim slučajevima obavljati ovakve zamene ne zavisi samo efikasnost procedure dokazivanja, već i njena kompletnost, pa istraživanje u oblasti dokazivanja teorema na računaru treba da obuhvata i mogućnosti upotrebe relacije apsorbcije.

LITERATURA:

1. Chang,C.,Lee,R.: Symbolic Logic and Mechanical Theorem Proving, Academic Press, 1973.
2. Plotkin,G.: A note on inductive generalisation, Machine Intelligence, vol.5,Meltzer, B.,Michie, D.,eds.
3. Plotkin,G.: A further note on inductive generalisation, Machine Intelligence, vol.6, Meltzer, B.,Michie, D., eds.

INFORMATICA 4/1983

NEPOSREDNO IZVAJANJE VISOKIH PROGRAMSKIH JEZIKOV NA OBJEKTNI ARHITEKTURI

V. ŽUMER,
P. KOKOL,
B. STIGLIC

UDK: 681.3.06

VISOKA TEHNIŠKA ŠOLA, MARIBOR, JUGOSLAVIJA
ISKRA, AVTOMATIKA, LJUBLJANA

V članku podajamo algoritem za LR razpoznavanje in neposredno izvajanje visokih programskih jezikov. Za zanesljivo in učinkovito izvajanje je predlagana objektna arhitektura. Prav tako so nakazane rešitve za izvajanje krmilnih stavkov in povezovanje parametrov.

DIRECT EXECUTION OF HLL ON OBJECT ARHITECTURE. The algorithm for LR parsing and direct execution of HLL is given. The object architecture for reliable and efficient execution is proposed. Solutions for control statements and parameter passing are also included.

1. UVOD

Z razvojem visokih programskih jezikov se je dvignil nivo programskega okolja sočasno s tem pa zahteva po interaktivnosti dela. Idealno interaktiven sistem je takšen, ki omogoča neposredno izvajanje kateregakoli programskega jezika in kjer med uporabnikovim nivojem in računalniško materialno opremo ni semantične vrzeli. Iz literature poznamo že nekaj poskusov sistemov za neposredno izvajanje. Ena glavnih slabosti teh sistemov je, da so narejeni samo za en jezik. V članku podajamo koncept sistema za neposredno izvajanje, ki je primeren za vse programske jezike. Samo pregledovanje in razpoznavanje se da enostavno rešiti univerzalno za vse jezike, če ob upoštevanju regularne gramatike in LR gramatike vnesemo ustrezne tabele. Te tabele vsebujejo lastnosti posameznega jezika in krmilijo univerzalne programe za pregledovalnik in za razpoznavnik. Ena največjih ovir za učinkovitost neposrednega izvajanja visokega programskega jezika je vsakokratno pregledovanje in razpoznavanje krmilnih stavkov. Z objektno orientirano arhitekturo obravnavamo vso informacijo enotno, torej tako podatkovne strukture, kot stavke programa. Pri tem so krmilne strukture predstavljene kot objekti, ki omogočajo hitrejše izvajanje struktur, ki se ponavljajo.

Izvajanje poteka s klicom procedur, ki najprej opravijo povezovanje in dinamične preverjanje. Te operacije so učinkovite, ker je semantika informacije skrita že v arhitekturi. Po povezovanju in preverjanju se opravijo še ostale potrebne operacije kot so n.pr. aritmetične operacije, klicanje procedur, izvajanje krmilnih stavkov, hkratno izvajanje itd.

S sodobnimi VLSI vezji, mikroprogramiranimi sistemi, programiranimi logičnimi polji in pomnilniškimi elementi je implementacija takega sistema tudi praktično opravičena.

2. RAZPOZNAVNIK

V splošnem je razpoznavnik sestavljen iz dveh delov: iz tabel in iz programa, ki te tabele upravlja /1/. Za različne gramatike se spreminjajo le tabele. Izkazalo se je, da je za neposredno izvajanje visokih programskih jezikov najbolj primerno LR razpoznavanje /2/. LR (left to right) razpoznavniki so takšni, ki razpoznavajo vhodni tekst iz leve proti desni in skonstruirajo najbolj desno izpeljavo.

Razpoznavnik ima sklad, vhod in tabelo. Vhod se bere iz leve proti desni, simbol za simbolom. Sklad vsebuje niz $s_0\ s_1\ s_2\ s_3 \ldots s_m$, kjer je s_m na vrhu sklada, s_i pa se imenuje stanje in vsebuje vse informacije o stanjih v skladu pod seboj.

Tabela je sestavljena iz dveh delov, iz funkcij ACTION in GOTO. Funkcija ACTION (s_m, a_i) ima lahko naslednje vrednosti

1. pomakni (shift) s
2. reduciraj $A \rightarrow \beta$
3. sprejmi
4. napaka

Funkcija GOTO (s_m, a_i) generira novo stanje s in predstavlja tabelo prehajanja stanj deterministističnega končnega avtomata, katerega vhodna abeceda so simboli gramatike.

Konfiguracija LR razpoznavnika je dvojica:

$$(s_0\ s_1\ s_2 \ldots s_m,\ a_i\ a_{i+1} \ldots a_n\ \$)$$

Prva komponenta je vsebina sklada, druga komponenta pa še ne prebran del vhodnega teksta. Naslednja akcija razpoznavnika je odvisna od trenutnega vhodnega simbola a_i in stanja na vrhu sklada s_m. Konfiguracija po tej akciji je glede na štiri vrednosti funkcije ACTION naslednje:

1. Če je funkcija ACTION (s_m, a_i) = pomakni s, razpoznavnik izvede pomik in dobimo naslednjo konfiguracijo

   $$(s_0\ s_1\ s_2 \ldots s_m\ a_i,\ a_{i+1} \ldots a_n\ \$)$$

2. Če je funkcija ACTION (s_m, a_i) = reduciraj

$A \rightarrow \beta$, razpoznavnik reducira niz β v neterminal A in dobimo naslednjo konfiguracijo:

$(s_0\ s_1\ s_2 \cdots s_{m-r}\ s, a_i\ a_{i+1} \cdots a_n\ S)$

Stanje s dobimo iz funkcije GOTO (s_{m-r}, A), kjer je r dolžina niza β , to je desne strani produkcije.

3. Če je funkcija ACTION (s_m, a_i) = sprejmi, je razpoznavanje končano.

4. Če je funkcije ACTION (s_m, a_i) = napaka, je razpoznavnik odkril napako in se pokličejo ustrezni podprogrami.

Naveden postopek za razpoznavanje z neposrednim izvajanjem je dan z naslednjim algoritmom.

```
Program RAZPOZNAVNIK(tabele,vhodni text);
(* NAJDI je procedura za iskanje po LR tabeli, izhod: ACTION in N *)
(* SCANER je presledovalnik *)
begin preberi vhodni tekst; I=0;          (* I je kazalec na vrh sklada *)
  S(I):=0; SCANER; NAJDI;                 (* S je sklad *)
  while ACTION ni sprejmi do
    while ACTION ni reduciraj do
      SCANER; NAJDI;
      if simbol je identifikator do push simbol v DS; (* DS delovni sklad *)
        if faza deklaracije do vpisi simbol v simbolno tabelo
        else ni faza deklaracije do poisci simbol iz simbolne tabele end
      end;
      if ACTION je napaka do javi in popravi napako
      else ACTION je pomakni do I:=I+1; S(I):=N end
    end;
    (* ACTION je reduciraj *)
    I:=I-R;                               (* R je dolzina produkcije *)
    NAJDI; I:=I+1; S(I):=N;               (* GO TO *)
    if ni napak do generiraj sliko ali izvajaj end;
  end;
  izpisi rezultate ali eventuelne napake
end.
```

Algoritem vsebuje pretvorbo krmilnih stavkov v slike oz. v reverzno poljsko obliko. Vsi ostali stavki se izvajajo neposredno kot so zapisani v vhodnem tekstu. Na tak način dosežemo največjo možno hitrost izvajanja vhodnega teksta. Prednost danega algoritma je, da se istočasno z razpoznavanjem opravlja tudi delna generacija vmesnega teksta in izvajanje. Tako je omogočena največja možna paralelnost dela, ki jo lahko realiziramo z več paralelnimi procesorji oz. s prekrivalno tehniko. Naveden algoritem je univerzalen za vse programske jezike, saj so značilnosti posameznega jezika vnešene v tabelah.

Algoritem vsebuje dve vgnezdeni zanki. Zunanja zanka se ponavlja tako dolgo, dokler ne dobi funkcija ACTION vrednosti sprejmi. To pomeni, da je razpoznavanje in izvajanje programa končano. V tem primeru se izpišejo rezultati in eventuelne napake.

V notranji zanki pa se izvaja akcija pomakni s ali napaka tako dolgo, dokler funkcija ACTION ne dobi vrednosti reduciraj $A \rightarrow \beta$. V tem primeru se izvrši redukcija ter izvajanje ali generacija slike.

Da se izognemo pri vgnezdenih krmilnih stavkih in rekurzivno deklariranih procedurah rekurzivnim slikam, se v tem primeru generirajo samostojne slike. Vstopi in naslovivseh slik so v glavni sekljalni tabeli. S tem dosežemo enoten dostop do vseh slik in večjo učinkovitost izvajanja.

3. IZVAJANJE

Neposredno izvajanje visokih programskih jezikov zahteva pozno povezovanje objektov. Če pa želimo, da je takšno izvajanje tudi učinkovito in zanesljivo, je potrebno opremiti objekte s semantiko /3/. Ko funkcija ACTION doseže vrednost reduciraj $A \rightarrow \beta$, razpoznavnik generira ali pokliče procedure za izvajanje. Te procedure lahko opravijo tudi semantično analizo kot n.pr. primernost tipa, meje indeksov polj, pravilnost parametrov itd.

Največ težav pri neposrednem izvajanju nastopi pri krmilnih stavkih, ker jih ne moremo izvesti neposredno /4/. Zato pri razpoznavanju generiramo slike krmilnih stavkov. Pri vgnezdenih krmilnih stavkih omogočimo pravilno izvajanje z dodatnim skladom, ki vsebuje adrese vrnitve zunanje slike. Ker ima vsak krmilni stavek in vsaka procedura svojo sliko z vstopom v glavni sekljalni tabeli, je izvajanje krmilnih stavkov enako izvajanju procedur. Na tak način je tudi učinkovito rešen problem rekurzije.

Z upoštevanjem objektne arhitekture se zelo poenostavi povezovanje parametrov pri klicu procedur. Dejanski parametri so v delovnem skladu in za vsak parameter se zgradi objekt. Nato se v delovnem skladu zamenja ime parametra z naslovom objekta tega parametra. Formalni parametri so v sliki procedure, ki se je generirala pri deklaraciji procedure. Zamenjava formalnega parametra z dejanskim je samo postavitev ustreznega kazalca na objekt dejanskega parametra.

4. ZAKLJUČEK

Naveden koncept neposrednega izvajanja visokih programskih jezikov se od dosedaj znanih razlikuje z učinkovitim in zanesljivim delovanjem. Poleg tega koncept ni vezan na en sam jezik, ampak je univerzalen za vse programske jezike.

LITERATURA

1. A.V. Aho, J.D. Ullman, Principles of Compiler Design, Adison-Wesley 1977

2. V. Žumer idr., Raziskava mikroprogramirane arhitekture za implementacijo visokega programskega jezika, raz. naloga, VTŠ, Maribor 1982

3. V. Žumer idr., Objektne arhitekture na osnovi semantične zgradbe informacije, INFORMATICA '83, Ljubljana 1983

4. P. Kokol idr., Neposredno izvajanje visokega programskega jezika, INFORMATICA '81, Ljubljana 1981

PROGRAMSKA PODRŠKA MIKRORAČUNARSKOG SISTEMA ZA UPRAVLJANJE, KONTROLU I DIJAGNOSTIKU STANJA ALATNOG STROJA

K. M. BOŠNJAK,
P. MARIĆ,
R. DEJANOVIĆ

UDK: 681.3:621.9

ELEKTROTEHNIČKI FAKULTET, BANJALUKA
JUGOSLAVIJA

U radu je data programska podrška koja je razvijena za mikroračunarski sistem koji vrši funkciju upravljanja alatnim strojem PEE 250. Programi su testirani i verificirani na pogodnom simulatoru.

MICROCOMPUTER SYSTEM SOFTWARE FOR THE CONTROL, CHECKING AND DIAGNOSTIC OF MACHINE TOOL. The paper deals with software which is developed for microcomputer system for the control machine tool PEE250. Programs are tested and tneir verification is done on the simulator.

1. UVOD

Računarsko upravljanje industrijskim procesima se sve više širi uz stalno smanjenje cijene hardware-a računara za rad u realnom vremenu a istovremeno se povećava njihova fleksibilnost i pouzdanost. Medjutim, cijena i složenost software-a i programiranja rastu mnogo brže,što ima za posljedicu da je ukupna cijena, zatim zakašnjenja,te slabosti instalacije računarskog upravljanja, problemi koji su uglavnom vezani za software. Zbog toga posebnu pažnju treba posvetiti razvoju programske podrške za računarske sisteme koji upravljaju industrijskim postrojenjima.

Za mikroračunarsko upravljanje alatnim strojevima odgovorna je programska podrška koju će u realnom vremenu provoditi mikroračunarska sklopovska struktura i prema kojoj će biti utvrdjivano stanje na davačima te aktiviranje u pravom trenutku odredjenih izvršnih elemenata. O programskoj podršci ovisi koliko se i kako provodi kontrola upravljačkih komponenata, kako se dijagnosticiraju kritična stanja i kako se izbjegavaju potencijalne teže posljedice utvrdjenih grešaka.

Algoritam automatskog upravljanja alatnim strojevima može se podijeliti na niz različitih stanja stroja. Stanje stroja odredjeno je stanjima senzora, davača odnosno stanjima ulaznih parametara upravljačkog uredjaja. Promjena stanja stroja, odnosno promjena stanja nekog od ulaznih parametara, ostvaruje se akcijom preko izvršnih elemenata tj. izvršenom promjenom izlaznih parametara upravljačkog uredjaja.

2. PRINCIP RADA ALATNOG STROJA

Alatni stroj se sastoji od pogonskog dijela (elektromotor sa odgovarajućom dodatnom opremom), radnog mehanizma (čiji je osnovni dio potiskivač), uredjaja za prenos pogonske energije sa elektromotora na potiskivač (spojnica, kočnica, elektropneumatski ventil i zamajac) i davača položaja potiskivača. Rotaciono kretanje osovine elektromotora se preko odgovarajućih uredjaja pretvara u translatorno kretanje potiskivača. Zato je položaj potiskivača, preko davača položaja pritiskivača, odredjen odgovarajućim uglom. Jedan ciklus kretanja potiskivača od gornje mrtve tačke (GMT) do donje mrtve tačke (DMT) i nazad,do GMT odgovara promjeni ugla od 360^0. Uglu od 0^0 ili 360^0 odgovara položaj potiskivača u GMT, a uglu 180^0 položaj potiskivača u DMT. Ovaj ciklus predstavlja radnu cjelinu koja izvrši obradu jednog uzorka materijala. Maksimalna sila potiska potiskivača pri kretanju je 2500 kN.

Način rada stroja obuhvata tri radna režima, koji se mogu odabrati odgovarajućim prekidačem.

2.1. Podešavanje

U režimu podešavanja se samo odabire početni položaj potiskivača, koji uglavnom zavisi od alata na potiskivaču. Pokretanje potiskivača se može vršiti dvoručno sa dva prekidača koji se moraju aktivirati u vremenskom razmaku od najviše 0.5 s. Pokretanje se može vršiti i nožnim prekidačem. Potiskivač se kreće sve dok se ne isključi bar jedan prekidač dvoručnog upravljanja ili nožni prekidač.

2.2. Pojedinačni rad

U ovom slučaju se aktivnost stroja odvija u pojedinačnim ciklusima.(promjena ugla od 0^0 do 360^0). Na početku ciklusa potiskivač se mora nalaziti u GMT (početni položaj). Pokretanje potiskivača je isto kao u režimu podešavanja. Potiskivač se do DMT kreće samo ako su uključena oba prekidača dvoručnog upravljanja ili nožni prekidač. Nakon dolaska u DMT potiskivač nastavlja kretanje u drugom smjeru, bez obzira da li su prekidači dvoručnog ili nožnog upravljanja uključeni. Na ovaj način se omogućuje rukovaocu alatnim strojem da izvadi obradjeni dio i postavi sledeći komad materijala. Nakon dolaska u GMT potiskivač staje neovisno o tome da li su prekidači dvoručnog ili nožnog upravljanja uključeni. Ovim je jedan radni ciklus završen. Za početak sledećeg ciklusa uslov je da se potiskivač nalazi u zoni GMT i da su prekidači dvoručnog upravljanja (ili nožni prekidač) isključeni. Ako su ovi uslovi ispunjeni stroj je spreman za naredni ciklus koji se aktivira kao i prethodni.

2.3. Automatski rad

U automatskom radu se potiskivač kreće neprekidno od GMT do DMT i obrnuto. Ovaj način rada se može aktivirati samo dvoručno. Pri tome je kretanje potiskivača do DMT isto kao u pojedinačnom radu. U trenutku kada potiskivač dodje u DMT mogu se prekidači dvoručnog upravljanja deaktivirati. Potiskivač nastavlja kretanje do GMT a zatim nastavlja kretanje u drugom smjeru bez potrebe da se novi ciklus posebno aktivira. Ovo automatsko ponavljanje ciklusa se odvija dalje neovisno o rukovaocu strojem sve dok se ne aktivira prekidač za isključenje automatskog rada. Nakon aktiviranja ovog prekidača potiskivač se pri prvom narednom dolasku u GMT zaustavlja, čime je završen režim automatskog rada.

2.4. Rad u liniji

Pored navedena tri načina rada, stroj ima mogućnost rada u liniji. Ovo je moguće jedino kada je stroj u automatskom radu i uz dodatak dodatne opreme. Stroj se može povezati sa drugim istim ili drugačijim strojevima za obradu metala, kao i samo sa uredjajem za dovodjenje materijala. Ovi dodatni uredjaji za povezivanje strojeva se istovremeno koriste za sinhronizaciju rada kompletnog postrojenja.

2.5. Mjere sigurnosti

U radu razmatranog stroja se zbog velikih radnih površina i sile potiska, mora posebno voditi računa o mjerama sigurnosti. Najvažnije su svakako zaštita rukovaoca stroja kao i osnovnih dijelova stroja (alat na potiskivaču i energetski dio).

U cilju zaštite rukovaoca je neophodno obezbjediti takav način upravljanja strojem da se eliminiše mogućnost bilo kakvih povreda rukovaoca, kao i da se propiše kompletan postupak u radu sa strojem. Sa stanovišta sigurnosti je najkritičnija faza kretanja potiskivača prema DMT.Neophodno je zato obezbjediti da se rukovaoc u tom vremenu ne može naći u radnom prostoru stroja. Ovo je obezbjedjeno načinom rada stroja. Rad na stroju sa nožnim uključivanjem ili s automatskim radom dozvoljen je samo ako je postavljena sigurna zaštita pristupa radnom prostoru stroja (pokriven ili zaštićen alat, fiksne ili pokretne mrežice i slično). Dvoručno aktiviranje omogućuje rad bez ovih zaštitnih uredjaja. Medjutim, u ovom režimu se potiskivač kreće prema dole samo ako su pritisnuta oba dugmeta dvoručnog aktiviranja. Ona su tako postavljena da se obadva ne mogu aktivirati jednom rukom. Ukoliko se bilo koje od njih otpusti potiskivač odmah staje. Pored navedenog je rukovaoc u mogućnosti da u slučaju potrebe u svakom trenutku potpuno zaustavi rad stroja aktiviranjem odgovarajućih prekidača.

U cilju obezbjedjenja ovakvog načina funkcionisanja stroja moraju se koristiti elementi velike pouzdanosti, a oni najbitniji i u paralelnom radu. Takodje se stalno prate "kritične" veličine. Mjeri se temperatura namotaja elektromotora i u slučaju pregrijavanja stroj odmah staje. Takodje se neprekidno moraju kontrolisati: pritisak u pneumatskoj instalaciji stroja, upravljački napon, ispravnost siguronosne podluške (koja se aktivira u slučaju preopterećenja stroja) i slično. Sve ovo obezbjedjuje zaštitu bitnih dijelova stroja, a time i dodatnu zaštitu rukovaoca.

Za stroj je tačno propisan standardni način rukovanja koji u normalnim uslovima eliminiše mogućnost povreda i uništenje bitnih dijelova stroja.

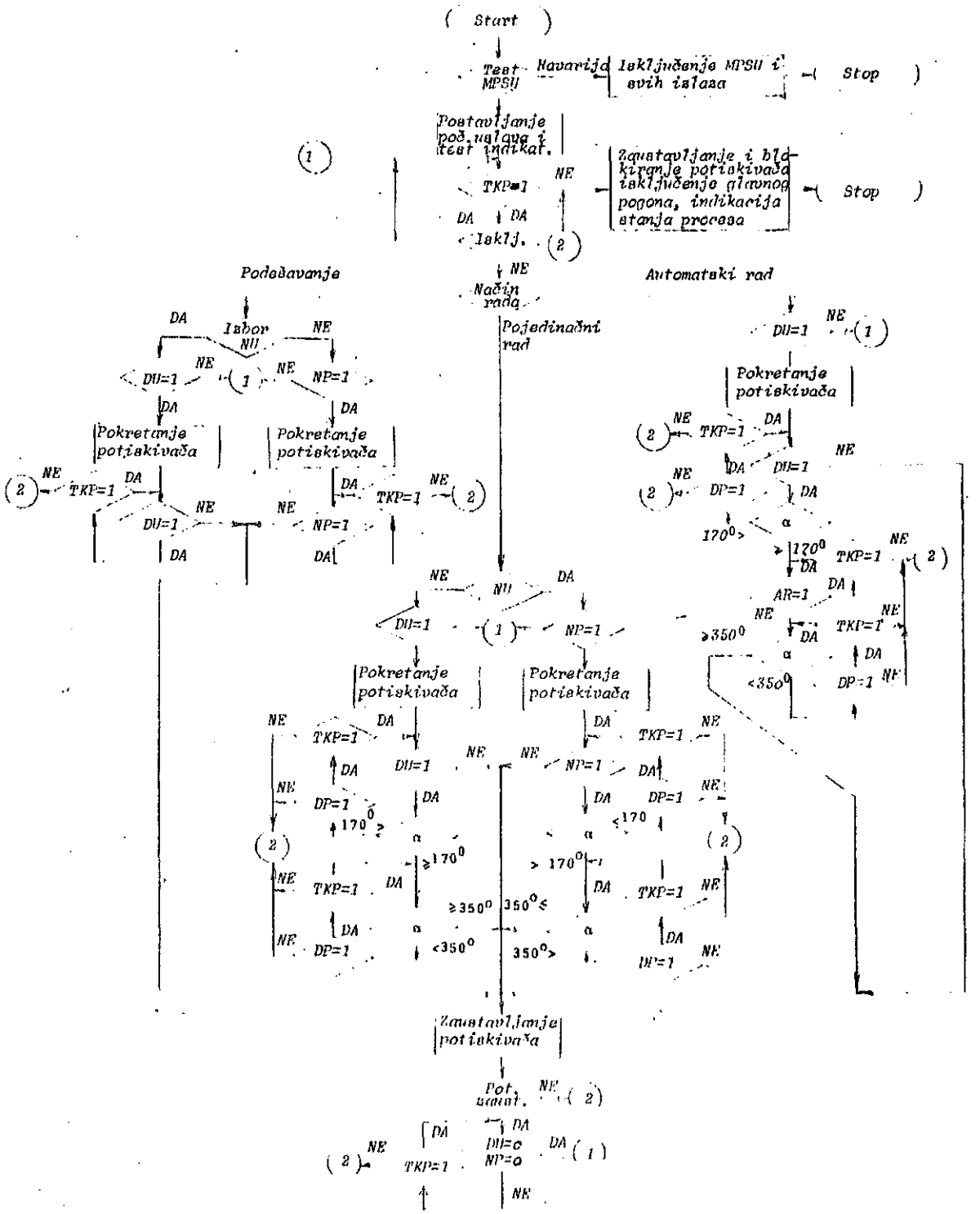
3. PRISTUP PROJEKTOVANJU PROGRAMSKE PODRŠKE

Mikroračunarski upravljački uredjaj ostvaruje algoritam upravljanja izvodeći program vodjenja procesa iz stanja u stanje prema tačno odredjenom algoritmu. Stanje procesa utvrdjuje se ispitivanjem ulaznih parametara (ulaznih linija mikroračunara) nakon čega se odredjuje sledeća akcija koju, prema algoritmu upravljanja, treba izvršiti da bi se proces doveo u sledeće stanje.

Izvršenje odredjene akcije (aktiviranje odredjenog izvršnog elementa) inicira se preko odredjenog izlaznog parametra (izlazne linije mikroraćunara).

Na samom početku izvodjenja programa treba,pr-

tavljaju početne vrijednosti radnih konstanti. Utvrdjuje se ispravnost rada RAM memorije, izlaznih registara i vremenskih pokazivača. Upravljački program počinje se izvoditi samo ako je utvrdjena ispravnost komponenata. U protivnom



Slika 1.

ogramabilni dio mikroraćunarske konfiguracije inicirati za odredjeni rad. Programiraju se ulazni i izlazni registri, vremenski pokazivači, zahtjevi za prekid i status riječi te pos-

se dojavljuje utvrdjena neispravnost u obliku šifre greške dok je onemogućena aktivnost izvršnih elemenata.

Programska podrška za provodjenje algoritma up-

ravljanja počinje utvrdjivanjem trenutnog stanja, odnosno provjerom radnih uslova za zadani mod rada. Ako su ispunjeni uslovi za akciju inicira se njeno izvodjenje. Prema dijagramu upravljanja poznato je koji se ulazni parametri moraju promijeniti izvršavanjem inicirane akcije, a poznato je i približno vrijeme u kojem se očekivana promjena ulaznih parametara mora dogoditi. Zato u nastavku upravljački program izlazi u odsječak za utvrdjivanje promjene ulaznih parametara, uz kontrolu pojave očekivanih dogadjaja u očekivanim vremenima.

3.1. Razvoj algoritma i programa za upravljanje konkretnim alatnim strojem

Razvoj programske podrške je zasnovan na prethodno uradjenom kinematskom dijagramu stanja. Neophodno je bilo obezbjediti programsku podršku koja će omogućiti potreban način funkcionisanja stroja uz uvažavanje njegovih kinematskih karakteristika.

Na početku izvodjenja programa se, nakon iniciranja programabilnog dijela, utvrdjuje ispravnost bitnih komponenata mikroračunarske konfiguracije. Ove komponente se dalje u toku izvodjenja programa neprekidno testiraju i u slučaju neispravnosti blokira rad stroja, a neispravnost dojavljuje u obliku šifre greške.

Provodjenje algoritma upravlja počinje utvrdjivanjem stanja stroja. Stanje stroja je odredjeno vrijednostima pojedinih parametara i željenim načinom rada. U skladu sa tim se iniciraju odgovarajući izlazi. Nakon ovoga se ponovo odredjuje stanje stroja. Pri tome se istovremeno dobiva informacija o uticaju pojedinih izlaza mikroračunarske konfiguracije na stanje stroja. Ovo omogućuje da se stroj dovede u željeno stanje i istovremeno kontroliše ispravnost funkcionisanja svih sklopova. U slučaju neočekivanog ponašanja pojedinih parametara, koje upućuje na nepravilan rad stroja, stroj se zaustavlja. Istovremeno se dojavljuje šifra parametra koja omogućuje da se relativno jednostavno tačno odredi mjesto i karakter neispravnosti. Dalja aktivnost stroja je potpuno onemogućena sve dok se ne otkloni neispravnost.

Algoritam automatskog upravljanja alatnim strojem je dat dijagramom toka na slici 1. Značenje skraćenih oznaka na slici je sledeće:

DU - prekidači dvoručnog upravljanja (DU=1 u slučaju da su oba prekidača uključena i to u roku od 0,5 s.);

NP - nožni prekidač nožnog upravljanja (za NP=1 prekidač uključen);

DP - davač položaja pritiskivača (DP=1 - davač ispravan);

α - ugao zakreta osovine davača položaja;

TKP - testiranje kritičnih parametara (TKP=1 - parametri su u dozvoljenim granicama);

AR - prekidač za zaustavljanje automatskog rada (za AR=0 automatski rad je završen);

NU - prekidač za izbor nožnog upravljanja.

4. ZAKLJUČAK

Osnovni zadatak sistema upravljanja pomoću mikroračunara je izdavanje izlaznih naredbi za stroj, na osnovi prije unešenog programa i stanja ulaznih veličina koje sistem za upravljanje stalno ispituje. Istovremeno sistem ispituje samog sebe. U slučaju kvara na objektu upravljanja ili na računarskom sistemu automatski dolazi do signalizacije greške te do njene lokalizacije. Na bilo koju utvrdjenu neispravnost trenutno se reaguje odgovarajućom zaštitnom akcijom. Svaka zaštitna akcija praćena je dojavom greške te njenom lokalizacijom.

Pošto cijena i složenost software-a i programiranja rastu mnogo brže od cijene hardware-a računarskih sistema za rad u realnom vremenu to je vrlo važno posvetiti pažnju razvoju ekonomične programske podrške.

LITERATURA

1. Leventhal,L., Walsh,C.: Microcomputer experimentation with the INTEL SDK 85, Prentice-Hall, Inc. New York 1980.
2. Herbert Y.Shang, Eric Manning and Geruot Metze: Fault Diagnosis of Digital Systems 1972.
3. INTEL: Component Data Catalog 1982.
4. Applying Microprocessors, New Hardware, Software and Applications, Electronics Book series, Mcgrow-Hill, New York 1976.
5. Microcomputer D.A.T.A.BOOK Edition 5, December 1978.
6. PEE 250/II Konstrukciona dokumentacija.

INTELIGENTNO TIPALO S SPOSOBNOSTJO RAZPOZNAVANJA VZORCEV

SAŠA PREŠERN

UDK: 681.3:681.52

INSTITUT JOŽEF STEFAN, UNIVERZA EDVARDA KARDELJA, LJUBLJANA, JAMOVA 39

Zahtevni senzorski sistemi, ki jih danes razvijajo v razvojnih laboratorijih po svetu in pri nas vsebujejo senzor le kot osnovno sredstvo za detekcijo signalov nekega procesa. Kako zbirati signale, da bo zbiranje in kontrola procesa učinkovita? Kako vključiti računalnik v senzorski sistem? Kako organizirati bazo senzorskih podatkov, da bo računalniško vodeni senzorski sistem omogočil razpoznavanje zahtevnih vzorcev? Ali lahko reprezentacija znanja poveča sposobnost računalniško vodenega senzorskega sistema? Članek odgovarja na ta vprašanja, ki so temeljnega pomena pri računalniško podprtih senzorskih sistemih in jih predstavi na primeru inteligentnega tipala, ki je sposobno razpoznati industrijske predmete in posamezne vzorce na predmetu.

AN INTELLIGENT TACTILE SENSOR WITH A CAPABILITY OF PATTERN RECOGNITION: In advanced sensing systems which are developed nowdays, represents a sensor only a basic device for signal detection. How to collect signals in order to make a sensing system and a process control efficient? How to organize sensor data base, so that a computer supported sensing system is capable to recognize complex patterns? Is it possible to increase the capability of a computer supported sensing system by knowledge representation? This paper gives answers to these important questions and demonstrates them on an intelligent tactile sensor that can recognise complex industrial objects, identify a seam location and track a three dimensional seam trajectory for an arc welding robot.

1. UVOD

Senzorski sistemi postajajo na področju avtomatike in kibernetike vse pogosteje mikroračunalniško vodeni sistemi. Kompleksni robotski in industrijski procesi zahtevajo povečano preciznost merskih in kontrolnih sistemov, ki omogočajo večjo prilagodljivost robotov in avtomatov v spremenljivem okolju. Mehansko tipalo, ki je podprto z mikroračunalniškim sistemom, je eden od učinkovitih senzorskih sistemov, ki jih danes uporabljajo v robotiki [PRE81A , BOL81 , LAG79].

V tem članku predstavljamo nekaj perečih in tesno povezanih problemov pri mikroračunalniškem vodenju senzorskih sistemov in jih ilustriramo na mikroračunalniško vodenem senzorskem sistemu s tipalom PRE81B . Sistem je namenjen opravljanju zahtevnih operacij razpoznavanja predmetov ali posameznih vzorcev na predmetu ter s tem v zvezi avtomatskem planiranju ustreznih akcij. Prav vključitev mikroračunalnikov v senzorski sistem, ki omogoča nov pogled na teorijo senzorskih sistemov, pa postavlja pred raziskovalca nova vprašanja v zvezi z mikroračunalniškim vodenjem, ki je podprto z bazo znanja in bazo podatkov. Podrobno bomo predstavili simbolni opis predmetov ali vzorcev, ki jih razpoznavamo s senzorskim sistemom ter reprezentacijo znanja, na čemer temelji avtomatsko planiranje akcij pri razpoznavanju.

V mikroračunalniško vodenih senzorskih sistemih uporabimo poleg numeričnih kontrolnih algoritmov še možnosti, ki nam jih nudi uporaba mikroračunalnika. Tako bomo videli, da je sestavni del kontrolnega postopka tudi:

- simbolni opis (predmeta, vzorca, okolice, ...)
- baza podatkov in
- reprezentacija znanja.

Posebno vlogo v hierarhični modularni organizaciji mikroračunalniško vodenih merilnih sistemov je treba posvetiti tudi metodam iskanja, ki omogočajo učinkovito primerjanje podatkov iz senzorja z bazo podatkov in iskanje prave rešitve.

Na posameznih stopnjah razpoznavanja predmeta ali vzorca izvršimo primerjanje in iskanje po drevesu, ki vsebuje strukturo problema. Pomemben je tudi plan dela, torej strategija zajemanja vzorcev, da bo identifikacija čim učinkovitejša. Oglejmo si rešitev te problematike na senzorskem sistemu s tipalom, ki je namenjen robotskemu varjenju.

Kontrolni algoritem vodi avtomatski varilni sistem. Na podlagi baze znanja o postopku tipanja in varjenja, ter informacij o trenutnem stanju, v katerem se nahaja varilni sistem, se izbere ustrezna faza kontrolnega algoritma.

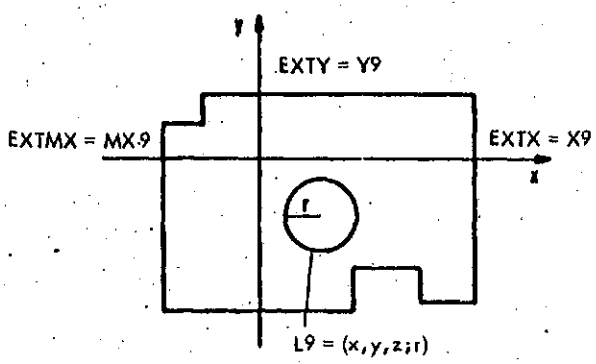
2. SIMBOLNI OPIS IN BAZA PODATKOV

Glavna naloga večine računalniško podprtih senzorskih sistemov je bodisi razpoznavanje predmeta in njegove lokacije ali pa razpoznavanje določenega dela na danem predmetu (to je reže, ki jo bo robot varil, izvrtine kamor se vstavi drugi predmet itd.). Da lahko računalniško vodeni senzorski sistem razpozna predmet, je potrebno izdelati simbolni opis prostorskih lastnosti predmeta. Če je tak opis pravilno združen s strategijo zajemanja vzorcev, lahko na preprost način reprezentiramo predmet ter s tem omogočimo učinkovitejše iskanje po drevesu ter hitrejšo identifikacijo predmeta.

Simbolni opis posameznega predmeta P sestoji iz strukturirane reprezentacije njegovih sestavnih delov. Ta reprezentacija je množica fizikalnih lastnosti predmeta Fi, i = 0,1,... ..., M, na primer (Slika 1):

PREDMET(i): površina P = Pi
ekstremna x točka EXTX = Xi
ekstremna -x točka EXMX = MXi
ekstremna y točka EXTX = Yi
ekstremna -y točka EXTMY = MYi
višina V = Vi
dolžina D = Di
izvrtina L = (Li1, Li2, ..., Lik)
k je št. izvrtin
izboklina C = (Ci1, Ci2, ..., Cij)
j je št. izboklin (1)

(V bodoče bomo zaradi enostavnosti privzeli omejitev, da imamo za dan predmet največ eno izvrtino in eno izboklino.)



Slika 1. Oblika predmeta in njegov simbolni opis.

Simbolni opis predmetov je shranjen v trajni bazi podatkov in ima obliko preprostih izrazov (simple expression) kot na primer

L4(x,y,z;r) (2)

kar pomeni, da ima predmet številka štiri izvrtino na lokaciji (x,y,z) s premerom r.

To je splošno znanje sistema in je shranjeno v trajnem spominu. Vsaka stopnja trajnega spomina vsebuje razred fizikalnih lastnosti predmetov (Slika 2), kot na primer razred površin predmetov, razred ekstremnih točk predmeta v x smeri, lokacijo izvrtine, itd.

Končna konjunkcija preprostih izrazov za dan predmet je simbolni opis predmeta kot na primer

PREDMET9 := X9 ∧ MX9 ∧ Y9 (3)

kar pomeni, da je predmet številka devet opisan s tremi količinami, ki jih vidimo na sliki 1.

Katere elemente simbolnega opisa bomo pri dejanskem razpoznavanju vključili v bazo podatkov, je seveda odvisno od vrste senzorja, ki ga uporabljamo. Tako bo površina ali bolje ploščina tlorisa pri računalniškem vidu koristen podatek, pri uporabi tipala pa odvečen. Obratno pa na primer velja za višino, ki je koristen podatek pri uporabi tipala in odvečen pri uporabi kamere, ki zazna le tloris predmeta.

Baza podatkov sestoji iz množice simbolnih opisov vseh predmetov.

STOPNJA	RAZRED	VREDNOST (n predmetov)
I	P	P1, P2, ..., Pn
II	EXTX	X1, X2, ..., Xn
III	EXTMX	MX1, MX2, ..., MXn
IV	EXTY	Y1, Y2, ..., Yn
V	EXTMY	MY1, MY2, ..., MYn
VI	V	V1, V2, ..., Vn
VII	D	D1, D2, ..., Dn
VIII	L	L1 = (x,y,z;r), ..., Lk = (x,y,z;r)
IX	C	C1 = (x,y,z;r), ..., Cj = (x,y,z;r)

Slika 2: Baza podatkov za razpoznavanje predmetov in posameznih vzorcev s pomočjo senzorja.

Podatke o prostorskih lastnostih predmeta organiziramo in reduciramo v stopnjah. Baza podatkov za razpoznavanje predmetov in posameznih vzorcev na predmetu s pomočjo inteligentnega tipala je organizirana kot hierarhična zbirka opisov predmetovih lastnosti, kjer vsaka stopnja hierarhije predstavlja različno geometrijsko lastnost predmeta.

Najprej služi senzor za detekcijo najbolj značilnih lokacij na površini predmeta, ki jih je tudi najenostavneje detektirati, kot na primer ekstremno točko predmeta v smeri x, dolžino predmeta, lokacijo izvrtine itd. Potem nastopi šele faza detaljnega razpoznavanja posameznih specifičnih lokacij na površini predmeta. Celoten ta korak kjer podatke o predmetu zberemo in zreduciramo imenujemo transformacija iz signalne reprezentacije v simbolno reprezentacijo predmeta. Vsak predmet je opisan s specifično simbolno reprezentacijo in tvori bazo podatkov za mikroračunalniško vodeni senzorski sistem.

3. REPREZENTACIJA ZNANJA

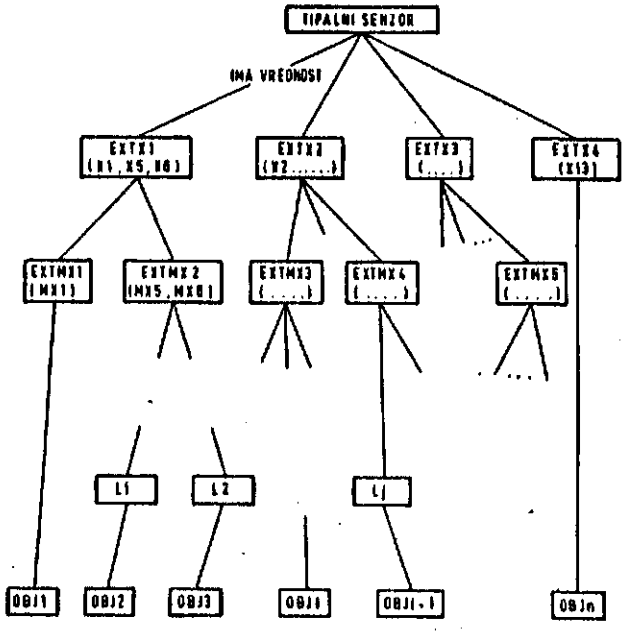
Pomemben problem pri opisu in interpretaciji predmetov je reprezentacija in uporaba vsega potrebnega in razpoložljivega znanja [NIL80]. Številne alternativne reprezentacije znanja, kot na primer mreže, okvirji (frames), produkcijska pravila in proceduralni opis lahko ponavadi podpirajo hierarhično strukturirano bazo znanja.

Podsistem, ki vsebuje bazo znanja našega tipalnega senzorskega sistema je podan kot semantična mreža z večnivojsko hierarhijo (Slika 3). Izvori znanja so podprogrami za gradnjo modela predmeta, ki ga tvori množica karakterističnih lastnosti (dolžina, ekstremna točka v x smeri, lokacija izvrtine, ...). Za razpoznavanje objektov in reže, ki naj jo robot vari, služi baza znanja, ki jo uporabi robot po tem, ko interpretira podatke iz senzorja. Opis predmetov v simbolni obliki in odnosi med posameznimi vozli v semantični mreži sestavlja bazo znanja, ki je bolj kompaktna kot pa zgolj podatki iz senzorja, torej zajema manj prostora v spominu in omogoča dobro strategijo iskanja po drevesu.

Vozli mreže znanja predstavljajo lastnosti predmetov (razrede simbolnega opisa) medtem, ko medsebojne povezave med vozli predstavljajo odnose med vozli.

Semantična mreža, ki predstavlja bazo znanja za razpoznavanje predmetov je na ta način strukturirana z namenom, da bi omogočila učinkovito in hitro iskanje. Vozli v zgornjem hierarhičnem nivoju vsebujejo tiste merske podatke, ki jih je možno določiti z najmanj računalniškega časa in v primeru uporabe tipalnega senzorja z najmanj premikanja robotove roke, ki drži tipalo. Včasih lahko že z merjenjem teh osnovnih podatkov identificiramo predmet ali pa vsaj drastično zmanjšamo preostanek drevesa za iskanje. V primeru, da predmet še ni identificiran je potrebna izločitev še drugih značilnosti na spodnjih hierarhičnih nivojih, ki nato omogočajo

enolično identifikacijo predmeta. Obdelava podatkov iz senzorja na spodnjih hierarhičnih nivojih sematične mreže zahteva več računalniškega časa in več spomina.



Slika 3: Baza znanja, podana kot semantična mreža z večnivojsko hierarhijo.

Poglejmo kako tvorimo semantično mrežo, ki predstavlja bazo znanja. Za vsako fizikalno količino F tvorimo AND-OR graf, ki ima sledeče lastnosti

i) vsebuje vrednosti dane fizikalne količine Fi za vse predmete i ($a \leq i \leq n$ pri čemer je n število predmetov);

ii) vse vrednosti, ki so znotraj intervala Fi + e in Fi − e so povezane z AND;

iii) vrednosti v AND grupah so urejene po velikosti;

iv) vrednosti OR so urejene po velikosti kot na primer:

$$F = \{Fa \vee Fb \vee (Fc \wedge Fd \wedge \ldots \wedge Fn)\}; \qquad (4)$$

$$Fa < Fb; \quad Fc < Fd < \ldots < Fn .$$

Velikost intervala, ki je določen z e se nastavlja s pomočjo kontrolne funkcije P in je zelo pomembna za hitro in enolično razpoznavanje. Več o tem bomo spregovorili kasneje, toda že sedaj vidimo, da je e tem večji čim bolj nenatančen je senzor in čim več je pri razpoznavanju prisotnega šuma.

4. PRIMERJANJE IN ISKANJE PO DREVESU

Pri primerjanju simbolnih reprezentacij predmetov z modelom, z namenom, da bi enolično interpretirali podatke iz senzorja ter identificirali predmet vodi v kombinatorsko iskanje. Poznamo številne kontrolne strukture, ki omejujejo kombinatorsko naraščanje: relaksacija, produkcijski sistemi, žarkovno iskanje (beam search [RED78]) itd. Implicitno zajema vsak od teh modelov tudi ustrezno strategijo iskanja: iskanje v globino, žarkovno iskanje itd.

Pri problemu razpoznavanja predmetov in iskanju reže s pomočjo senzorja (tipala) uporabimo tako obliko primerjanja in iskanja, ki je hkrati:

i) zadostna za razpoznavanje predmeta v motenem industrijskem okolju in

ii) učinkovita v smislu, da je struktura drevesa pri iskanju odvisna od kompleksnosti predmeta, ki ga razpoznavamo in njegove različnosti od ostalih predmetov, ki so vključeni v bazo podatkov.

To nam omogoča žarkovno iskanje. Podatke iz senzorja primerjamo z mrežo znanja ter s tem generiramo množico preostalih predmetov, ki imajo podobno vrednost fizikalne količine, ki jo je podal senzor. Zaradi prisotnosti šuma se realni predmeti delno razlikujejo od pripadajočih simbolnih modelov. Torej lahko problem razpoznavanja predmetov s senzorji obravnavamo kot problem iskanja pri drevesu v prisotnosti motenj.

Z uporabo žarkovne heuristike najdemo optimalno pot skozi graf tako, da na vsakem nivoju oklestimo vse tiste potomce, ki ne sodijo v snop rešitev, ki se le malo razlikujejo od najboljše poti. Tako zmanjšamo eksponencialno naraščanje števila poti skozi graf ter se hkrati izognemo potrebi po vračanju po grafu (backtracking) ali kakšnemu iterativnemu postopku iskanja, ki bi bil ob uporabi drugačne strategije iskanja potreben zaradi šuma ali pa zaradi majhnih translatornih ali rotacijskih premikov realnega predmeta glede na model.

Pomembna lastnost računalniško vodenega senzorskega sistema za razpoznavanje predmetov je dinamična gradnja simbolne reprezentacije objekta med samim postopkom iskanja. Za vsak naslednji nivo v mreži znanja je potrebno dodatno zbiranje in računalniška obdelava signalov iz senzorja. Če je možno razpoznati predmet na prvem nivoju mreže znanja, je iskanje končano in ni potrebna nobena nadaljnja računalniška obdelava (predmet N na sliki 3). Samo v primeru, če na tem nivoju predmet ni bil razpoznan, se nadaljuje razpoznavanje in obdelava podatkov na naslednjem nivoju. Ta način dinamične gradnje simbolnega modela predmeta pomeni prihranek časa pri obdelavi podatkov, ker spodnji nivo mreže znanja zahteva več računalniškega časa za obdelavo podatkov iz senzorja. Zato so performanse opisanega računalniško vodenega senzorskega sistema boljše kot pa performanse sistema, ki ne uporablja večnivojske mreže znanja in dinamične gradnje simbolne reprezentacije predmeta.

Na vsakem nivoju mreže znanja, se simbolna reprezentacija predmeta, ki je shranjena v začasnem spominu, primerja z vsemi relevantnimi podatki v bazi podatkov za ta nivo. Funkcija podobnosti Pi se izračuna za vse predmete i v bazi podatkov, ki so potomci danega vozla

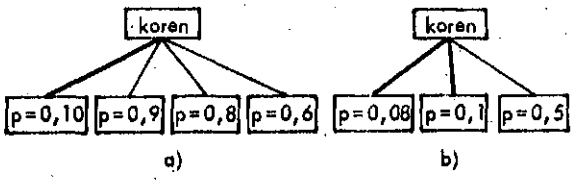
$$Pi = \frac{A - Bi}{Bi} \quad 1 \leq i \leq M ,$$

kjer je

A - podatki iz senzorja za določen predmet (shranjeni v začasnem spominu);
Bi - podatki v bazi podatkov za predmet i (trajni spomin);
M - število elementov v bazi podatkov, ki so povezani s prejšnjim vozlom, to je število potomcev vozla i).

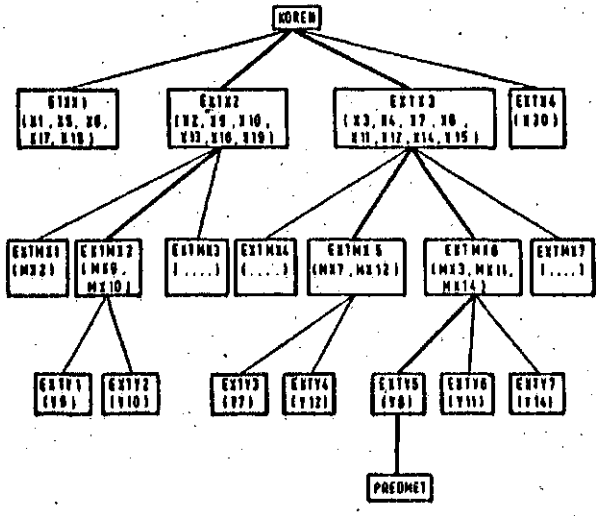
Vse vozle, ki imajo vrednost funkcije podobnosti P večjo kot p (p = 0.15) oklestimo (pogoj urejenosti (iv) nam pospeši klestenje) in iskanje se nadaljuje na naslednjem nivoju z reducirano mrežo znanju.

Na sliki 4 vidimo, da smo v primeru a oklestili 3/4 drevesa, v primeru b pa 1/3 drevesa.



Slika 4: Klestenje drevesa, ki temelji na funkciji podobnosti P.

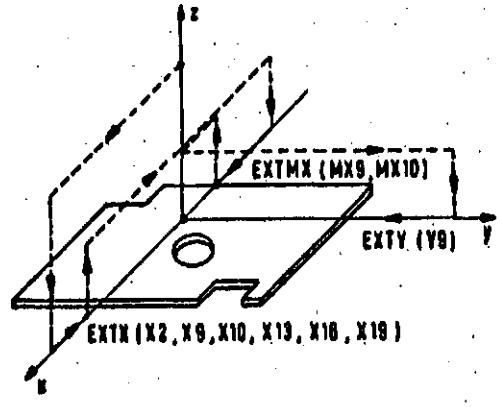
Klestenje temelji na izračunu vrednosti funkcije podobnosti P. Tako je vrednost funkcije P element, ki služi za gradnjo iskalnega drevesa in določa množico potomcev, ki jih bomo obravnavali na naslednjem nivoju (Slika 5).



Slika 5: Iskalno drevo za razpoznavanje predmetov, ki uporabljajo funkcijo podobnosti P na vsakem nivoju.

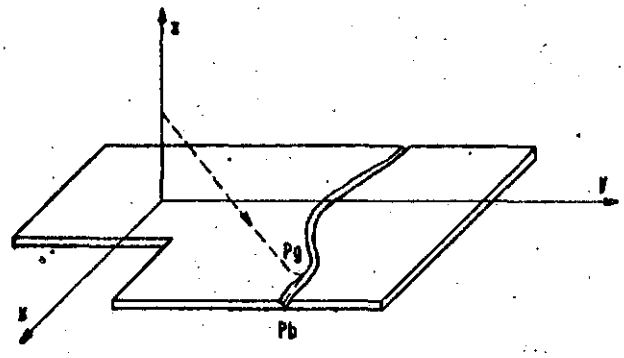
5. DELOVANJE INTELIGENTNEGA TIPALA V REALNEM ČASU

Ko sprejme robot predmet, ki naj ga zavari, je treba najprej identificirati lokacijo reže. Čeprav je idealna lokacija reže shranjena v robotovem spominu, je dejanska lokacija reže pogosto premaknjena. Pri večini industrijskih aplikacij v robotskem varjenju razpoznavanje predmetov ni velik problem ker ponavadi poznamo tip predmeta, ki naj ga robot zavari. Toda strategija identifikacije reže in klasifikacije predmetov je podobna v smislu, da s tipalom robot "otipa" različne lokacije na predmetu in zbira informacije postopoma. (Slika 6). Tako na primer robot razpozna neznani predmet tako, da najprej izmeri EXTX. Po tej metodi je neznani predmet lahko eden od sledečih { predmet 2, predmet 9, predmet 10, predmet 13, predmet 16, predmet 19}. Po meritvi EXTMX je neznani predmet samo še eden od { predmet 9, predmet 10}. Po meritvi EXTY pa je jasno, da gre za predmet 9.



Slika 6: Grafičen prikaz gibanja robotove roke med postopkom identifikacije predmeta.

Ko je predmet, ki ga je treba variti poznan (bodisi, da ga robot identificira sam ali pa sprejme to informacijo) začne robot s strategijo za identifikacijo reže. Ta strategija zavisi od vrste predmeta in od orientacije površine predmeta. Robot premakne tipalo k površini predmeta, blizu reže. Potem vleče tipalo tako, da igla drsi po površini predmeta vse dokler ne registrira s tipalom reže (točka Pg na sliki 7). Nato se tipalo pomakne proti začetku reže (točka Pb na sliki 7) in ga registrira z značilnim nenadnim odmikom tipalne igle. Sedaj je natanko znana lokacija začetka reže.



Slika 7: Grafičen prikaz gibanja tipala med procesom identifikacije reže.

Ko je predmet identificiran in so znane koordinate začetka reže, je naloga tipalu detektirati odmike dejanske trajektorije od idealne. Na podlagi izhoda iz tipala upravlja mikroračunalnik pozicioniranje varilne šobe.

Sistem je trenutno v razvojni fazi. Tipalo je pritrjeno na avtomat za varjenje s tremi prostostnimi stopnjami.
To sistem uporabljamo za laboratorijsko testiranje avtomatskega adaptivnega sledenja tridimenzionalnih trajektorij. Pričakujemo, da bomo končno verzijo sistema z inteligentnim tipalom uporabili za razpoznavanje 20 do 40 predmetov ali lokacij na posameznem večjem predmetu in za sledenje večline tridimenzionalnih trajektorij.

6. ZAKLJUČEK

Namen raziskave obsega reševanje perečih problemov mikroračunalniško vodenih senzorskih sistemov, ki podpirajo delovanje zahtevnih robotskih procesov. Rezultati raziskav so ilustrirani na primerih senzorskega sistema s tipalom. Računalniško podprt senzorski sistem za uporabo v robotiki zahteva pristop, ki je problemsko orientiran za razpoznavanje predmetov in identifikacijo reže. Samo na ta način je lahko kontrolni sistem za tipalo majhen in učinkovit in ga je mogoče implementirati na mikroračunalniku.

Predlagali smo hierarhično organizacijo baze podatkov za dane predmete. Uporabili smo optimalni žarkovni algoritem primerjanja in iskanja, ki je zlasti primeren za razpoznavanje predmetov v industrijskem okolju. Metoda reševanja posameznih problemov in celoten sistem, ki smo ga opisali je mogoče razširiti na večje število predmetov, kolikor jih pač lahko robot streže.

Sledenje reže z opisanim sistemom smo testirali v laboratoriju, ne pa še med postopkom varjenja ali v električno motenem industrijskem okolju, kjer bomo morali uporabiti dodatno zaščito proti elektromagnetnim motnjam.

Za postopek razpoznavanja predmetov in avtomatskega iskanja začetka reže je potrebno imeti tipalo, ki vsebuje poleg dveh pasivnih prostostnih stopenj tudi eno aktivno notranjo prostostno stopnjo. Tako tipalo je trenutno v izdelavi v našem laboratoriju. S tem tipalom bomo povečali število možnih strategij "tipanja" in hitrost posameznih faz tipalnega postopka. Z novim aktivnim tipalom bomo tudi preizkusili opisane strategije razpoznavanja predmetov in avtomatskega iskanja začetka reže.

ZAHVALA. Zahvaljujemo se Razvojnemu institutu Iskra za uporabo modularnega avtomata za varjenje.

7. LITERATURA

[BOL81] J.G. Ballinger: Using a Tactile Sensor to Guide a Robotic Welding Machine, Sensor Review, No. 3, July 1981.

[LAG79] B. Lagerlof, Current Industrial Robot Applications for Arc Welding, Proc. 9th Int. Symp. on Industrial Robots, Washington D.C., 1979, pp. 99-106.

[NIL80] N.J. Nilson: Principles of Artificial Intelligence, Tioga Pub. Co., 1980.

[PRE81A] S. Prešern et al.: Tactile Sensing System with Sensory Feed-back Control for Industrial Arc Welding Robots, Proc. 1st Int. Conf. on Robot Vision and Sensory Control, Stratford-upon-Avon, 1981.

[PRE81B] S. Prešern, I. Ozimek in M. Špegel: Mikroprocesorsko vodenje senzorskega sistema za robotsko varjenje, Informatica 4/81, pp. 33-35, 1981.

[RED78] D.R. Reddy in S. Rubin: Representation of Three Dimensional Objects, Technical Report, Computer Science Department, Carnagie-Mellon University, 1978.

MIKRORAČUNALNIŠKI SISTEM ZA VODENJE VISOKOREGALNIH SKLADIŠČ

JANEZ PETELINKAR

UDK: 658.78.011.56:681.3

ISKRA AVTOMATIKA, TOZD PROJEKTIRANJE IN GRADNJA SISTEMOV, LJUBLJANA

Microcomputer system for control of high-level-shelf stores

Abstract

The article gives the layout of microcomputer system for control of storage with high-level shelves in tire- factory VAZ Togliatti in USSR.

Microcomputer system is assembled of terminal station on shelf-lifts and central microcomputer which takes care for work coordination and connection with purchaser computer from which the demands for delivery of tires from production are transmitted.

The system enables automatic operation of the store where the purchaser computer at the beginning of the shift transmitts the data about necessary quantities and sort of tires however microcomputer system independenty assures a correct material supply from the store.

Microcomputer system solves also the problems about supply and delivery of material from the store and enables a supervision of the whole system by operating register. Microcomputer system has two modes of operation, automatic and manual one being performed by operational keyboard. Due to its reliability the central microcomputer has a stand-by. Data about the arrangement of material in the store are for reliability reasons sent also into the purchaser computer. This enables the reestablishment of system operation at every moment.

Mikroračunalniški sistem za vodenje visokoregalnih skladišč

Povzetek:

Članek podaja koncept mikroračunalniškega sistema za vodenje procesa skladiščenja v visokoregalnem skladišču gum v tovarni VAZ Togliatti v SZ.

Mikroračunalniški sistem sestavljajo končne postaje na regalnih dvigalih in centralni mikroračunalnik, ki skrbi za koordinacijo dela in povezavo z računalnikom kupca preko katerega prihajajo zahteve za izdajo gum proizvodnji. Sistem omogoča avtomatsko delovanje skladišča, kjer računalnik kupca ob začetku izmene pošlje podatke o potrebni količini in vrsti gum za proizvodnjo v izmeni. Mikroračunalniški sistem nato samostojno skrbi za pravilni odtek materiala iz skladišča.

Mikroračunalniški sistem rešuje tudi probleme hkratnega pritoka in odtoka materiala v skladišče ter preko obratovalnega protokola omogoča nadzor nad celotnim sistemom. Mikroračunalniški sistem pozna dva načina obratovanja, popolnoma avtomatsko obratovanje ter ročno upravljanje preko funkcijske tastature. Zaradi zanesljivosti delovanja je centralni mikroračunalnik podvojen. Razpored materiala v skladišču pa zaradi povečanja zanesljivosti pošilja tudi v računalnik kupca. S tem je v vsakem trenutku omogočena ponovna vzpostavitev operativnega stanja sistema.

1. Naloge računalniškega sistema v avtomatiziranem skladišču

1.1. Naloge centralnega (oskrbovalnega) računalnika v skladišču

Odvisno od velikosti in tipa skladišča, organizacije celotnega skladiščnega poslovanja in nivoja avtomatizacije skladiščnih operacij se naloge oskrbovalnega računalnika močno razlikujejo, vendar jih je možno deliti v dve skupini: obvezne naloge in fakultativne.

Pri obsežnejšem skladišču mora oskrbovalni računalnik opravljati naslednje naloge:

- vodenje praznih skladiščnih lokacij po vrstah skladiščne transportne enote (gabaritih),
- vodenje zaloge na posameznih skladiščnih lokacijah s šiframi vskladiščenega materijala,
- vodenje skupnih zalog posameznih vskladiščenih artiklov,
- vodenje datumov sprejema posameznega artikla v skladišče,
- izpis skladiščnih dokumentov (izdajnic, dobavnic).

Lista fakultativnih nalog je obsežnejša:

- vodenje cene posameznih artiklov,
- evidenca in prednost izdaje z načetih skladiščno - transportnih enot,
- rok uporabnosti posameznih artiklov (skadenca),
- vodenje evidence o posebnih lastnostih blaga (napake, posebne izvedbe, barve in sl.),
- določitev izbora skladiščno - transportne enote, ki se isti dan komisionira za več komisionov in se tako popolnoma izprazni,
- predpisovanje oz. izbor predpisane skladiščno - transportne enote za skladiščenje posameznih artiklov,
- način skladanja artiklov na skladiščno - transportne enote po vnaprej predpisanem vzorcu,
- optimiranje skladiščnih manipulacij transportne opreme z uporabo dvojnih ciklov pri povratnih načinih skladiščenja,
- izpis komisionirnega lista po načelu strategije najkrajših poti mehanskih pomagal,
- izpis nalogov za vnos sprejetega blaga,
- izpis nalogov za iznos blaga pri skladiščih z ročnim krmiljenjem mehaniziranih naprav,
- izbor alternativnih dosegljivih skladiščnih enot v primeru okvare posameznih mehaniziranih pomagal,
- signalizacija kompletiranja posameznih naročil,
- adresiranje pošiljk.

1. 2. Naloge mikroračunalniškega krmilnega sistema

V odviru vodenja procesa skladiščenja prevzema mikroračunalniški sistem naslednje naloge:

- komuniciranje z nadrejenim računalnikom zaradi prevzema nalog in pošiljanja posebnih informacij o spremembah stanja v skladišču. Naloge je možno posredovati posamično (posamezen hod dvigala) ali skupno za daljše časovno obdobje (običajno bo to izmena). Povratne informacije o izvršenih nalogah omogočajo arhiviranje posnetka stanj skladišča v nadrejenem računalniku. To je pomembno za samo materialno poslovanje skladišča in hkrati povečuje zanesljivost delovanja mikroračunalniškega sistema (možnost obnovitve stanja ob morebitnem izpadu mikroračunalniškega sistema).

- vodenje regalnih dvigal na osnovi dobljene naloge iz nadrejenega računalnika ali naloga, ki ga vnese operater preko funkcijske tastature. Vodenje izvajajo končne postaje (mikroračunalnik ali prosto-programirni sistem z zmožnostjo komuniciranja s centralnim mikroračunalnikom) na osnovi ukaza iz centralne postaje.

- vodenje transportnega sistema na osnovi dobljenega naloga iz nadrejenega računalnika ali iz funkcijske tastature. Vodenje se izvaja preko končnih postaj ali neposredno preko digitalnih izhodov.

- nadzor nad stanjem regalnih dvigal in transportnega sistema. Nadzor obsega prikaz trenutnih pozicij regalnih dvigal na numeričnem prikazovalniku in eventuelno prisotnih alarmnih signalizacij na sinoptiki in v obratovalnem protokolu. Obratovalni protokol lahko obenem vsebuje tudi zapis vseh opravljenih nalogov. Stanja regalnih dvigal zajemajo končne postaje na regalnih dvigalih. Stanje transportnega sistema zajemajo neposredno digitalni vhodi ali končne postaje. Stanje transpotrnega sistema je prikazano na sinoptiki. Alarmne signalizacije so protokolirane.

2. Zgradba sistema

Oskrbovalni del sistema je odvisen od namembnosti skladišča (število skladiščenih artiklov, število manipulacij). Njegovo konfiguracijo je potrebno določiti v vsakem primeru posebej. V primeru, ko gre za skladišče z majhnim številom artiklov (do 100) je možno tudi na tem nivoju uporabiti ustrezno konfiguriran mikroračunalnik.

Krmilni del se konceptualno ne spreminja. V odvisnosti od velikosti skladišča in števila upravljanih podsistemov se spreminja število končnih postaj in število digitalnih vhodov/izhodov. Periferija sistema je odvisna od zahtev kupca.

Končna postaja (mikroračunalnika ali PPS) predvideva zajem cca 60 bitov signalizacij (vključno s pozicijo) in izdaja do 60 komandnih bitov (možna razširitev z dodatnimi vhodno/izhodnimi moduli). Opremljena je z 32 kB pomnilnika (RAM ali ROM) in modulom za komunikacijo s centralno postajo (od 50 Bd do 600 Bd).

Centralna postaja skladišnega sistema omogoča priklop do 20 končnih postaj (regalnih dvigal) v stand-by režimu ali do 10 končnih postaj v hot stand-by režimu. Opremljena je z 2 x 64kB pomnilnika (RAM in ROM).

Centralna postaja skladišnega sistema je s končimi postajami povezana v zvezdasti konfiguraciji preko induktivnih zank. Povezava krmilnega in oskrbovalnega računalnika je serijska asinhrona po protokolu RS 232 ali RS 422.

Protokolirno-posluževalni podsistem omogoča prikaz informacij na izbranem mediju (pisalnik, alfa/numerični terminal) in izbrani način posluževanja (funkcijska tastatura, alfa/numerična tastatura).

Centralna postaja transportnega sistema zbira podatke iz transportnega sistema preko serijske povezave s končnimi postajami ali/in preko digitalnih vhodov (odvisno od števila informacija in njihove prostorske porazdelitve).

Celotni krmilni sistem je zasnovan modularno in je zgrajen iz komponent družine MOTOROLA 6800.

Krmilni sistem je distribuiran. Stopnja distribuiranosti je odvisna od velikosti sistema in zahtev uporabnika. V vsakem primeru pa je vsako regalno dvigalo opremljeno s svojo končno postajo.

Obravnavani sistem s svojo modularnostjo omogoča široko paleto izvedb, ki se med seboj razlikujejo po periferni opremi, številu samostojnih mikroprocesorskih enot ter po vrsti in obsegu vgrajene programske opreme.

Ker je oskrbovalni računalnik za normalno obratovanje skladišč te vrste in velikosti nujen in v večini primerov ob modernizaciji ali gradnji skladišča že obstoja, ostaja investitorju le nabava krmilnega mikroračunalniškega sistema.

Cena tega sistema, ki se giblje od enega do največ deset odstotkov od vrednosti opreme takšnega skladišča, je gotovo razlog za njegovo uvajanje. Poudariti pa je potrebno, da je pri določenem obsegu informacij takšna rešitev tudi najcenejša.

AVTOMATIZACIJA MALE HIDROELEKTRARNE Z MIKRORAČUNALNIKOM

MIHA SKUMAVC, dipl. ing.

ISKRA DO AVTOMATIKA TOZD PROJEKTIRANJE IN GRADNJA SISTEMOV

UDK: 621.311.21-52

Referat podaja osnovne kriterije za avtomatizacijo hidroelektrarn, za tem obravnava problematiko hierarhije in načina vodenja. Opisane so osnovne funkcije avtomatizacije na nivoju agregata, hidroelektrarne in možnosti daljinskega centra vodenja.

Osnova sistema predstavlja aparaturna in programska oprema mikroračunalniškega sistema TI-30E, ki je kompatibilna in omogoča poljubno razširitev tudi na vsa ostala področja uporabe v energetiki, tako za vodenje, kot za procesiranje podatkov.

Automation of small hydroelectric generating station with microcomputer

Report explains the basic criteria for automation of hydroelectric generating station and deals with problematics of hiearchy an control mode.

If describes all basic functions of automation on the level of aggregate, hydroelectric generating station and possibility of remote control centre.

The system bases on hardware and software of microcomputer system TI-30E which is very compatible and enables possible enlargement to all other fields of application in electroenergetics for control as well as for processing of data.

1.0. Karakteristike male hidroelektrarne Savica

Mala hidroelektrarna Savica je del proizvodnega elektro-energetskega sistema DES - Ljubljana, TOZD Elektro Sava Kranj. Z ostalimi elektrarnami in porabniki je povezana preko 35kV daljnovoda. V primeru razpada v RTP proti Radovljici omogoča samostojno napajanje bližnje okolice z električno energijo.

Hidroelektrarno Savica karakterizirajo naslednje lastnosti:

a. Hidrodinamične:

- veliki razponi dotokov vode, od 0.15 do 150m /s,
- minimalna zajezitev z fiksno pregrado in volumnom cca 140m ,
- izredno dolg rov, preko 2km, s premerom 2m in minimalnim naklonom proti vodostanu,
- vodostan s praznilnim vodom za dušenje hidravličnih nestabilnosti. Volumen vodostana je okoli 50m ,
- jeklen (tlačni) cevovod, ki povezuje vodostan z šobami turbin. Dolžina cevovoda znaša 600m, pri premeru 0.8m, zagotavlja skoro celoten padec, 227m, vodne energije.

b. Strojna in elektro oprema:

Hidroelektrarna Savica razpolaga s dvema generatorjema na katerih oseh sta po dve turbini. Tak koncept namestitve turbin omogoča izredno velik razpon obratovanja na nivoju optimalnih izkoristkov, pri različni razpoložljivi vodni energiji. Vsak od generatorjev daje na izhodu moč 2MW, kar odgovarja skupnemu pretoku skozi obe turbini 3.2m /s. Letna proizvodnja elektrarne je do 20.000MWh.

c. Tipi obratovanja glede na dotok vode v jez:

Pri postavitvi sistema avtomatskega obratovanja male hidroelektrarne smo imeli v cilju zadostiti osnovnim aspektom:
- možnost optimiranja delitve moči skupnega sistema na posamezne komponente (iz DCV),
- optimiziranje izbire turbine (vrste obratovanja) v elektrarni,

- optimizacija delitve moči izbranih turbin v igri.

Na osnovi tega smo postavili naslednje tri tipe obratovanja hidroelektrarne Savica:
- pretočno obratovanje, ki pomeni spreminjanje pretoka skozi šobe turbin glede na spremembo dotoka, pri pogoju polne akumulacije vode v jezu,
- akumulacijsko obratovanje, elektrarna je priključena na omrežje ne oddaja pa delovne energije, turbinske igle so zaprte. V tem obratovanju akumuliramo vodno energijo za potrebo končnega obratovanja,
- končno obratovanje, zagotavlja proizvodnjo maksimalne delovne moči v zahtevanem trenutku.

d. Tipi obratovanja glede na mrežo:

- paralelno z mrežo, v tem primeru je prvenstvena naloga MHE Savica sodelovanje v pokrivanju porabe električne energije (prelaganje energije),
- otočno obratovanje, ko z električno delovno energijo napaja bližnjo okolico (regulator frekvence). Otočno obratovanje z mikroračunalnikom je predvideno le preko turbinskega regulatorja.

2.0. Funkcije sistema za vodenje

Sistem za vodenje MHE Savica, ki je razvit v Iskri Avtomatiki omogoča:
- avtomatski start/stop in obratovanje hidroelektrarne pri optimalnih izkoristkih, glede na trenuten dotok vode, brez posadke,
- enostavno ročno poseganje v vodenje preko tastature in avtomatsko koordinacijo stroj - operater,
- kombiniramo ročno/avtomatsko vodenje elektrarne,
- ročno obratovanje s kompletnim nadzorom in protokoliranjem
- prikaz vseh hidravličnih, mehanskih in električnih veličin,
- napovedovanje dogodkov (gibanja nivoja).

3.0. Struktura sistema

3.1. Uporaba MR v vodenju male hidroelektrarne

Uporaba mikroračunalnika ponuja nove alternative tudi pri avtomatizaciji elektroenergetskih objektov, tako pri avtomatiziranih sistemih za nadzor, krmiljenje in regulacijo. V Iskri Avtomatiki je uvajanje mikroračunalniške avtomatizacije v elektro energetske sisteme v fazi projektiranja in postavitve prvih tovrstnih sistemov. V članku je opisan rezultat daljšega dela na zahtevni projektno - raziskovalni nalogi, ki zajema tako celovit sistem kot je hidroelektrarna.

Koncept vodenja je izpolnjen z rezultatu obsežnih meritev in simulacij, ki smo jih konec leta 1982 izvedli na sami hidroelektrarni. Vse meritve in simulacije, ki so bile izvedene na elektrarni nam razdelil v štiri energetske celote:
- meritve hidrodinamičnega sistema,
- meritve na strani strojne opreme,
- meritve na strani elektro opreme,
- prenos (transmisija).

Rezultati analiz so, kot konstante oz. teoretična sistemska spoznanja, s pridom uporabljeni v postavitvi uporabniških modulov in osnovni filozofiji povezave uporabniških modulov v prekinitveno drevesno strukturo.

3.2. Aparaturna oprema

Osnovni aparaturni del sistema je zgrajen okoli enotnega vodila, kar omogoča poljubno aparaturno razširitev in hitrejši dostop do skupnega spomina (DMA). Osnova opisanemu sistemu, ki ga je razvila Iskra Avtomatika in se imenuje TI-30, je Motorola M 6800.

Zaradi zahteve po veliki varnosti obratovanja sistema za vodenje je izbrana podvojena konfiguracija mikroračunalniške opreme. Izhodi - komande obeh, sicer v celoti ločenih sistemov, so vodeni na modul, ki kontrolira enakost izhodov vodečega in spremljajočega mikroračunalnika. V primeru enakosti izhoda znotraj nastavljive časovne tolerance, tako v začetni kot končni fronti, so vrata za izhod komande na relejno ploščo odprta. Sicer je prožena prekinitev in blokada izhoda. Računalnik še naprej proces nadzoruje, ne posega pa več vanj. Podvojena klasična zaščita v tem primeru avtonomno izvede alarmne ukrepe na elektrarni. Aktivlranje računalnika je na zahtevo RESET.

Vse enote so na evropa karticah dvojnega formata in se nahajajo v dveh omarah elektronike. Sistem sestavljajo naslednji moduli:
- V/I moduli, vhodno/izhodni moduli,
- CPU modula, centralne procesne enote,
- RAM/ROM moduli, spomin, (1 RAM/ROM omogoča 36k, kombinirano po 2k)
- TIM, ura,
- NAK, modul za nadzor trajanja komand,
- PIK, modul za kontrolo enakosti komand.

Vhodi v sistem so razdvojeni v relejni omari in zaščiteni proti motnajam. Oba mikroračunalnika imata možnost pristopa do skupne periferije, ki obsega:
- povezavo s funkcijsko tastaturo,
- povezavo s procesom.

Oba mikroračunalniška ciklično izvajata kontrolo lastnega delovanja. Vsaka odkrita napaka na enem računalniku bo povzročila ustavitev obeh. S tem je izvedena posebna rešitev redundantnega sistema s sinhroniziranim dodatnim računalnikom, ki pa ni v stanju "vroče rezerve".

3.3. Programska oprema

3.3.1. Zajem podatkov

Sistem zajema naslednje kategorije stanj:
- analogne meritve,
- informacije (pretočni nivo),
- normalne alarme (buchholz 1),
- nujne alarme (buchholz 2).

Posebno je rešen prenos analognih meritev z jezu. Zajem informacij je izveden:
- ciklično (čas cikla je 15s),
- občasno ciklično (čas cikla je 1s, 0.1s),
- zajem spontanih informacij,
- zajem na programsko zahtevo,
- zajem na zahtevo operaterja.

Z obstoječo aparaturno opremo sistem za vodenje MHE v 10ms zajame spremembo stanj na 240 enobitnih vhodov in v času 1s izvede in prevzame 55 analognih meritev.

3.3.2. Procesiranje

Na osnovi dosedanjih spoznanj je bil razvit v Iskri Avtomatiki sistem MOSPIK (mikrooperacijski sistem protokoliranja in vodenja), ki je zgrajen na obstoječi aparaturni opremi TI-30 E. MOSPIK je sestavljen iz treh enot:
- zajema informacij,
- obdelave informacij,
- izdaja informacij (protokoliranje).

Te enote so hierarhično zgrajene. Med hierarhičnimi nivoji, kjer so različne hitrosti procesiranja, ima sistem vmesnike - krožne pomnilnike. Te vmesnike sistem uporablja za izmenjavo informacij med funkcijskimi enotami. Funkcije so razdeljene v programske module, ki jih v osnovi delimo na sistemske in uporabniške. Prvi sistem uporablja za sinhrono delovanje celote, uporabniški moduli pa vršijo konkretne naloge.

Digitalne informacije sistem obdeluje tako, da primerja staro in novo stanje byta informacij (8 vhodov). Odkritje spremembe na vhodu prožl določen aplikacijski programski modul, če dogodek zahteva krmiljenje, če ne, se izvede le protokoliranje. Meritve sistem predhodno filtrira, računanje pragov, povprečij, maksimalne in minimalne vrednosti itd.

Monitor temelji na naslednjih principih:
a) naenkrat se izvaja le en programski modul,
b) izvajanje modula je prekinjeno:
   - modul je izveden do kraja, oz. naletel je na ukaz "kliči sheduler",
   - v primeru nastopa IRQ signala; po prevzemu IRQ skočimo v sheduler, ki izbere programski modul, ki se bo izvajal. Izbira modula s shedulerja je izvedena na podlagi:
     - prioritete,
     - stanje koordinacijskega števca (število ponovitev modula),
     - stanje programskega modula (pripravljen, ustavljen),
   - nek modul se bo izvajal le, če bo:
     - v stanju pripravljen,
     - če ima najvišjo prioriteto.

Ena izmed funkcij monitorja je start sistema, oz. zagon. V tej fazi je izveden postopek:
- monitor postavi vse "startne" module v stanje "pripravljen",
- po prioriteti se izvrše zagonski deli vseh modulov.

V času delovanja sistema vsaka informacija zase povzroči klic ustreznega programskega modula. Struktura sistema je izrazito prekinitvena, s čimer dosežemo strogo periferno iniciativo.

3.3.3. Izdaja informacij in ukazov

Izhodi iz sistema za vodenje MHE so grupirani v naslednje skupine:
- komande preko modula DIZK, trajne z maksimalnim trajanjem enega cikla 2 minuti. V proces jih oddajata oba, glavni in spremljajoči mikroračunalnik.
- signali, na čelni plošči. Spontani in jih izdaja le vodeči mikroračunalnik. To so nujne signalizacije za ročno vodenje.
- izpis na pisalnik (opremljen s točnim časom)
  - cikličen,
  - spontan,
  - na zahtevo.
- prikaz na TV monitorju. Prikaz je izveden na zahtevo na funkcijski tastaturi. Tastatura je sestavljena iz šestih sekcij. V vsaki sekciji je potujoča lučka, ki sveti, ko je omogočen vnos ali zahteva (grafika). Tipke imajo dodatne lučke, ki označujejo da so bile zahteve sprejete.

3.3.4. Komunikacije

Sistem za vodenje male hidroelektrarne omogoča naslednje povezave:
- povezava jez (analogne meritve) - mikroračunalnik v HE,
- povezava med vodečim in spremljajočim mikroračunalnikom,
- povezava z oddaljenim centrom vodenja (predlog).

4.0. Bodoče naloge mikroračunalniškem vodenju malih hidroelektrarn iz DCV

Hierarhična struktura vodenja, kakršno omogoča sistem v MHE Savici zagotavlja razpoložljivost in zanesljivost.

Z njo dosežemo naslednje funkcijske nivoje:
- individualni kontrolni nivo,
- grupni kontrolni nivo,
- nadzor elektrarne.

Za to je potrebno optimizirati pretok signalov in komand med posameznimi nivoji. Podatki so hranjeni in ohranjeni čim več na vsakem nivoju.

5.0. Literatura

(1.) M. Skumavc: Poročilo o meritvah na MHE Savica, Iskra Avtomatika, Ljubljana 82

(2.) M. Skumavc: Projekt za izvedbo del - mikroračunalniški sistem za avtomatizacijo hidroelektrarne Savica, Iskra Avtomatika, Ljubljana 1983.

INFORMATICA 4/1984

ALGEBARSKA SVOJSTVA SKUPA SUPSTITUCIJA

D. MARTINOVIĆ

MATEMATIČKI INSTITUT, BEOGRAD

UDK: 512.5:681.3

U toku obavljanja logičke indukcije i dedukcije na računaru na svakom koraku se koriste supstitucije, zato dajemo neke njihove osobine koje je potrebno poznavati, da bi se sa njima radilo.

ALGEBRAIC CHARACTER OF A SET COMPOSED OF THE SUBSTITUTIONS: When doing logical induction and deduction on the computers, substitutions are used at almost every step. Therefore it is useful to know the algebraic structure of the set of a substitutions.

Supstitucije u logici igraju veliku ulogu, kada god se govori o semantičkoj strani logičkih problema, o modelima formula, interpretacijama, koriste se supstitucije, mada ne uvek eksplicitno. Interesantno je stoga istražiti kakvi odnosi postoje unutar skupa supstitucija definisanih nad azbukom koju čine simboli konstanti, promenljivih i funkcija.

Supstitucije se definišu kao skupovi uredjenih parova tipa (t_i,v_i), $i=1,\ldots,k$, kojima je prva komponenta term, a druga promenljiva. Osim toga sve promenljive v_i, $i=1,\ldots,k$ moraju biti različite medju sobom i moraju se razlikovati od njima odgovarajućeg terma t_i u paru. Domen supstitucije čine sve promenljive koje su druge komponente uredjenih parova. Supstitucijama se u logičkim izrazima vrše zamene promenljivih sa termima, na taj način što se sva pojavljivanja neke promenljive v_i u logičkom izrazu, zamene odgovarajućim termom t_i iz supstitucije, a kao rezultat se dobija slučaj zadatog izraza u odnosu na zadatu supstituciju.

Iz teoreme $\frac{F(x)}{F(t)}$ se vidi odnos izmedju logičkog izraza i njegovog slučaja, pa je jasno da se supstitucije tako mnogo koriste, jer omogućuju da se primenom odredjenih jednostavnih sintaksnih pravila, proizvode logičke posledice polaznih formula.

Def.1: Kompozicija supstitucija $\theta=\{(t_1,x_1),\ldots,(t_n,x_n)\}$ i λ u oznaci $\theta\cdot\lambda$ je skup $\theta'\cup\lambda'$, gde su u λ' samo oni parovi iz λ čije druge komponente nisu u Dom θ, a u θ' su svi parovi oblika $(t_i\lambda, x_i)$ za koje je $x_i\neq t_i\lambda$, $i=1,\ldots,n$. //

Iz definicije se vidi da je kompozicija dve supstitucije takodje supstitucija, tj. da je skup Ψ zatvoren u odnosu na operaciju kompozicije. Takodje se jednostavno pokazuje da je kompozicija supstitucija asocijativna. U skupu postoji i neutral u odnosu na operaciju kompozicije, to je prazna supstitucija ε, koja svaku promenljivu prevodi u samu sebe. Dakle $(\Psi,\cdot)$ je asocijativan grupoid sa 1.

Kako je jasno da svaka supstitucija nema sebi inverznu, pogledajmo koje uslove treba da zadovoljavaju supstitucije θ i σ, da bi važilo $\theta\cdot\sigma=\varepsilon=\sigma\cdot\theta$.

Neka je $\theta=\{(t_1,x_1),\ldots,(t_n,x_n)\}$ i $\sigma=\{(s_1,y_1),\ldots,(s_m,x_m)\}$, tada je $\theta\cdot\sigma=\theta'\cup\sigma'=\{((t_i\sigma,x_i)/(t_i,x_i)\in\theta, t_i\sigma\neq x_i\}\cup\{(s_j,y_j)/(s_j,y_j)\in\sigma, y_j\notin$ Dom $\theta\}$, $\sigma\cdot\theta=\sigma''\cup\theta''=\{(s_i\theta,y_i)/(s_i,y_i)\in\sigma, s_i\theta\neq y_i\}\cup\{(t_j,x_j)/(t_j,x_j)\in\theta, x_j\notin$ Dom $\sigma\}$.

Da bi važila navedena definicija inverznosti, skupovi θ', σ', θ'' i σ'' moraju biti prazni. Da bi to

bilo ispunjeno mora važiti:
$t_i\sigma=x_i$, $x_i \in Dom\ \sigma$, $i=1,\dots,n$; $s_j\theta=y_j, y_j \in Dom\ \theta$ $j=1,\dots,m$.

U tom slučaju su domeni ove dve supstitucije jednaki, a njihovi termi su promenljive. Tako dolazimo do zaključka da su jedine supstitucije koje zadovoljavaju navedene uslove: 1) prazna supstitucija i 2) supstitucije koje biunivoko preslikavaju skup promenljivih na samog sebe i u tom slučaju iz $(y_i,x_i)\in\theta$ sledi $(x_i,y_i)\in\theta^{-1}$. U literaturi se često koriste supstitucije koje se nazivaju invertibilnim, ali kojima bi više odgovarao naziv slabo invertibilne supstitucije.

Def.2: $\theta=\{(t_1,x_1),\dots,(t_n,x_n)\}$ je slabo invertibilna supstitucija akko su svi termi t_i medjusobno različite promenljive, u kom slučaju je $\theta^{-1}=\{(x_1,t_1),\dots,(x_n,t_n)\}$. //

Slabo invertibilnih supstitucija ima daleko više nego invertibilnih, ali kod njih, kao što ćemo videti ne važi klasična definicija inverznosti: tako je $\theta\cdot\theta^{-1}=\theta'\cup(\theta^{-1})'=\{t_i\theta^{-1},x_i)/(t_i,x_i)\in\theta, t_i\theta^{-1}\neq x_i\}\cup\{(x_i,t_i)/(x_i,t_i)\in\theta^{-1}, t_i \notin Dom\ \theta\}=\varphi\cup(\theta^{-1})'\neq\varepsilon$ sem u slučaju da je θ invertibilna u strožijem smislu, kao što je i $\theta^{-1}\cdot\theta\neq\varepsilon$ u opštem slučaju.

Primer: $\sigma_1=\{(x,y),(y,z),(z,x)\}$,
$\sigma_1^{-1}=\{(y,x),(z,y),(x,z)\}$, $\sigma\cdot\sigma^{-1}=\varepsilon=\sigma^{-1}\cdot\sigma$
$\sigma_2=\{(z,x),(x,y)\}$, $\sigma_2^{-1}=\{(x,z),(y,x)\}$,
$\sigma_2\cdot\sigma_2^{-1}=\{(x,z)\}$, $\sigma_2^{-1}\cdot\sigma_2=\{(x,y)\}$
$\sigma_3=\{(z_1,x),(z_2,y)\}$, $\sigma_3^{-1}=\{(x,z_1),(y,z_2)\}$,
$\sigma_3\cdot\sigma_3^{-1}=\sigma_3^{-1}$, $\sigma_3^{-1}\cdot\sigma_3=\sigma_3$

Ovde samo supstitucija σ_1 pretstavlja primer invertibilne supstitucije.

Skup invertibilnih supstitucija čini grupu u odnosu na operaciju kompozicije.

Ostaje još da vidimo da li kod nekih supstitucija, kompozicija ima osobinu komutativnosti.

Neka su date supstitucije $\theta=\{(t_1,x_1),\dots,(t_n,x_n)\}$ $\sigma=\{(s_1,y_1),\dots,(s_m,y_m)\}$, njihove kompozicije ćemo prikazati na sledeći način:

$\theta\cdot\sigma=\theta\cup\sigma'=\{(t_i\sigma,x_i)/(t_i,x_i)\in\theta, t_i\sigma\neq x_i,$
$x_i \notin Dom\sigma\}\cup\{(t_i\sigma,x_i)/(t_i,x_i)\in\theta, t_i\sigma\neq x_i, x_i\in Dom\sigma\}\cup$
$\cup\{(s_j,y_j)/(s_j,y_j)\in\sigma, y_j \notin Dom\ \theta\}$

$\sigma\cdot\theta=\sigma''\cup\theta''=\{(s_j\theta,y_j)/(s_j,y_j)\in\sigma, s_j\theta\neq y_j,$
$y_j \notin Dom\ \theta\}\cup\{(s_j\theta,y_j)/(s_j,y_j)\in\sigma,$
$s_j\theta\neq y_j, y_j\in Dom\ \theta\}\cup\{(t_i,x_i)/(t_i,x_i)\in\theta, x_i \notin Dom\ \sigma\}$.

Da li ova dva skupa bila jednaka, moraju važiti sledeće jednakosti:

1) za sve $x_i \notin Dom(\sigma)$, mora biti $t_i\sigma\neq x_i$ i $t_j\sigma=t_i$,
2) za sve $y_j \notin Dom(\theta)$, mora biti $s_j\theta\neq y_j$ i $s_j\theta=s_j$,
3) za svako $y_j\in Dom(\sigma)\cap Dom(\theta)$, takvo da je $s_j\theta\neq y_j$, postoji $x_i\in Dom(\sigma)\cap Dom(\theta)$ takvo da je $x_i=y_j, x_i\neq t_i\sigma$ i tada je $t_i\sigma=s_j\theta$.

Obrnuto, za svako x_i iz preseka domena, postoji njemu jednako y_j takvo da važe navedene relacije.

4) ako je $y_j\in Dom(\sigma)\cap Dom(\theta)$ i $s_j\theta=y_j$, mora biti za neko x_i iz preseka domena, $x_i=y_j$ i $x_i=t_i\sigma$.

Sumirajmo sada navedene činjenice: proizvod supstitucija sa nepraznim presekom domena je komutativan akko su zadovoljeni sledeći uslovi:
a) oni parovi u $\sigma(\theta)$ koji imaju druge komponente van $Dom(\sigma)\cap Dom(\theta)$ imaju za prve komponente terme u kojima ne učestvuju promenljive iz $Dom(\theta)$ ($Dom(\sigma)$).

b) Oni parovi u $\sigma(\theta)$ koji imaju druge komponente iz $Dom(\sigma)\cap Dom(\theta)$, i za koje je $y_j\neq s_j\theta(x_i\neq t_i\sigma)$ imaju prve komponente takve da važe jednakosti $t_i\sigma=s_j\theta$, dok parovi u $\sigma(\theta)$ kod kojih je $y_j=s_j\theta$ $(x_i=t_i\sigma, x_i=y_j)$ imaju za prve komponente promenljive i važi sledeće:

ako je $y_k\in Dom(\sigma)$, a $x_i\in Dom(\theta)$ onda iz $\{(y_k,x_i),(x_i,x_i)\}\subseteq\theta$ sledi da $\{(x_i,x_i),(x_i,y_k)\}\subseteq\sigma$

Medjutim, zbog a), moraju biti $x_i,y_k\in Dom\theta\cap Dom\ \sigma$ i to $x_i=y_k$. Tako θ i σ moraju da sadrže parove $\{(z,x_i),(x_i,z)\}$, gde $z\in Dom\ \theta\cap Dom\ \sigma$.

Ako pretpostavimo da je $\theta''=\sigma'=\emptyset$, dobijamo slučaj $Dom\ \sigma=Dom\ \theta$, pa moraju važiti zaključci iz b), a ako je presek domena prazan, važe zaključci iz a). Pod pretpostavkom da je $\theta''=\sigma''=\emptyset$, važi i $\theta''=\sigma'=\emptyset$ (i obrnuto), pa dobijamo vezu $\sigma\cdot\theta=\theta\cdot\sigma=\varepsilon$, pa su θ i σ inverzne supstitucije. Poslednja mogućnost je da su θ i σ prazne i tada svakako mogu da komutiraju.

Primer: a) $\theta=\{(f(z_1,z_2),x_1),(a,x_2),(z_3,x_3)\}$
$\sigma=\{(z_1,y_1),(g(b),y_2)\}$, $\theta\cdot\sigma=\theta\cup\sigma=\sigma\cdot\theta$

b) $\theta=\{(z,x),(a,z_1)\}$ i $\sigma=\{(a,z),(z_1,x)\}$
$\theta\cdot\sigma=\sigma\cdot\theta=\{(a,x),(a,z),(a,z_1)\}$

c) $\theta=\{(z,x_1),(x_1,z),(a,x_2)\}$,
$\sigma=\{(z,x_1),(x_1,z),(f(b),y)\}$
$\theta\cdot\sigma=\{(a,x_2),(f(b),y)\}=\sigma\cdot\theta$

Kako smo videli, invertibilne supstitucije pripadaju skupu supstitucija koje mogu da komutiraju sa svojim parom. Ali da bi za ceo skup važilo pravilo komutativnosti, ono bi moralo da se odnosi na svake dve supstitucije u skupu. Dakle, zaključujemo da su jedine podgrupe ranije konstruisane grupe supstitucija koje su Abelove, trivijalne podgrupe, koje čine samo neutralni element ϵ, ili $(\theta, \theta^{-1}, \epsilon)$.

* Evropska skupina za 5. generacijo *

Podjetje SPL International je ustanovilo evropsko skupino vodilnih industrijskih in raziskovalnih organizacij kot odgovor na japonski načrt pete računalniške generacije, po katerem naj bi se Japonci v letu 1990 pojavili na tržišču s popolnoma novimi in različnimi vrstami računalnikov. Ta skupina izpolnjuje pogoje Alveyevega poročila britanski vladi v zvezi s peto računalniško generacijo. SPL ugotavlja, da je vladna investicija v razvoj pete računalniške generacije nujna, na razpolago pa je zaenkrat za te namene 350 milijonov funtov, in sicer za industrijske raziskave, podobne japonskim. Skladno z Alveyevim poročilom je bilo konec lanskega leta ustanovljeno podjetje Imperial Software Technology; njegovi ustanovitelji so Imperial College, National Westminster Bank, Pactel in Plessey; to podjetje naj bi organiziralo industrijske in univerzitetne raziskave predvsem v smeri nove programirne in programske tehnologije.

Nadalje je šest podjetij, med njimi SPL, Pactel in Logica ustanovilo skupino za skupen razvoj programske opreme za peto računalniško generacijo. SPL je medtem organiziralo tudi prvo posvetovanje o peti generaciji izven Japonske in izdalo svoj izvedenjski sistem Sage. SPLova skupina Insight bo delala na dveh področjih: na sistemih, ki so osnovani na znanju in na povezavi s končnim uporabnikom. Delo bo vodil svetovalni kolegij osmih članov, med njimi sta tudi Alex d'Agapeyeff in Bob Kowalski, izvedenca za jezik Prolog, ki so ga Japonci izbrali za razvoj programske opreme pete generacije. Skupina Insight je omejena na 20 organizacij z letnim prispevkom 7000 funtov. S temi prispevki se bodo pokrili izdatki dvotedenskega študijskega potovanja po svetu in v Japonski, z ogledom raziskovalnih središč.

Pri nas o podobnih pobudah doslej nismo razmišljali; izjema so posamezne delovne organizacije, ki so začele oblikovati skupine izvedencev za usmeritve prihodnosti na področju računalništva in informatike. Na področju skupščinskih in vladnih institucij pa se podobne pobude doslej niso pojavile.

A.P.Železnikar

* Konec miniračunalnikov *

Več dejavnikov napoveduje konec miniračunalnikov; med njimi je tudi predsednik največje neodvisne softwarske hiše MSA International. Mikroračunalniki vse bolj spodrivajo svoje starejše predhodnike, saj njihova zmogljivost narašča, cene kabinetnih miniračunalnikov pa se morajo spričo tega zniževati. To mnenje potrjuje tudi vrsta strokovnih sestankov. Na mesto minijev bodo stopili bolj inteligentni mikroji v obliki računalniških delovnih postaj in te bodo komunicirale z velikimi kabinetnimi računalniki. Največji problem današnjih mikrojev je napor, povezan z vnosom podatkov za najbolj popularne aplikacije, kot je npr. VisiCalc. Uporabnik si danes dejansko želi komunikacijo, ker ga vnos podatkov bistveno omejuje.

V prihodnosti je potrebno razviti prevsem komunikacijo med mikroji in kabineti, tako da bo zahtevna programska oprema kabinetov dejansko dosegljiva tudi uporabnikom mikrojev in le določene podatkovne plasti naj bi se posredovale mikrojem. Med drugim naj bi to omogočilo tudi dobivanje podatkov iz kabinetov za potrebe programov v mikrojih. S tem bi odpadlo individualno zajemanje in vstavljanje podatkov. Prva resnejša povezava kabinet/mikro je predvidena za IBMov osebni računalnik (PC). Komunikacija bo tekla skozi protokolski pretvornik med kabinetom in telefonsko linijo z mikrojem. Ta pretvornik bo mogoče vgraditi tudi v PC.

A.P.Železnikar

* Rojstvo mikroračunalnika *

Razvoj novih mikroračunalnikov je vselej povezan z velikimi napori razvijalcev, še posebej tedaj, ko gre za inovacijski razvoj. V to kategorijo razvoja sodijo npr. razvoj Applove Lise, Fujitsujevega Micro 16 in tudi Deltinega Partnerja. Najbrž bi se večkrat morali vprašati, kdo so ljudje, ki so uresničili take projekte. Druga vprašanja, ki se ob tem nujno postavljajo, pa so še način organizacije projekta, kvaliteta in koncentracija razvojnih kadrov, izjemni pristopi k delu in izjemni odnosi do kadrov, posebno delovno ozračje in nujen pojav kriz v času razvoja, nesoglasja z menažersko strukturo, izoliranost inovacijskih projektov od drugih projektov v podjetju itd.

Silicijska dolina, ki leži 70 km južno od San Francisca, je zibelka vrste inovacij na področju mikroračunalnikov. To je dolina majhnih naselij in posebnih ljudi, ki kujejo mikroelektronsko prihodnost - revolucijo nad tehnološkimi revolucijami. - Posebne vrste drama v Silicijski dolini je bil tudi razvoj mikroračunalnika Lisa. Prave inovacije podjetja Apple, ki ima svoj sedež v Cupertinu. Lisa je izredno uporabniško prijazen in dolgo pričakovani mikroračunalniški (osebni) sistem, ki je zbudil največjo pozornost v tehnoloških in poslovnih okoljih strokovnjakov. V dveh letih tega razvoja so se pojavljale najrazličnejše vesti o njegovi fantastičnosti, o popolnoma novih lastnostih osebnega računalnika. V projektu Lisa se je rodil nov stroj, ki ima vsa znamenja skupinske inovativnosti in ga je spričo tega vredno natančneje spoznati.

Vodja projekta (J. Couch) je prišel v Apple za polovično prejšnjo plačo od Hewlett-Packarda v letu 1978, vendar z vznemirljivo nalogo, da oblikuje strategijo prihodnjih sistemov osebnih računalnikov. Že takrat je bilo jasno, da je treba iskati glavno smer napredka v pisarniških sistemih, tam kjer obstajajo viri za prodajo napredne tehnologije. Drugo izhodišče je bila ugotovitev, da je naslednja stopnja napredovanja tkim. programska revolucija in določena aparaturna evolucija. Ime Lisa je bilo izbrano kot ime uporabniško prijaznega sistema in temu cilju se je podredil programski in aparaturni razvoj.

Določene izkušnje so bile dobljene na osnovi neprijaznosti mikroračunalnika TRS-80, kjer je bilo za poenostavitev njegove uporabe potrebnih več in več novih programov. Na tem primeru je

POSTALO JASNO, DA BI BILO MOČ VRSTO MEHANIZMOV INTEGRIRATI V OSNOVNI SISTEM, TAKO DA BI APLIKACIJE POSTALE LAŽJE REŠLJIVE. PROJEKT LISA SE JE ZAČEL IZVAJATI S ŠIROKO PERSPEKTIVO: STROJ NAJ BI BIL PISARNIŠKO USMERJEN, ZMOGLJIV IN LAKHO UPORABEN. PRI TEM NI BILO POVSEM JASNO, KAKO NAJ BI BILI TI CILJI DOSEŽENI. KOT PRVO SE JE POSTAVILO VPRAŠANJE UPORABNIŠKE POVEZAVE, KAKO NAJ BI UPORABNIKI UPRAVLJALI S STROJEM. DO TEDAJ SO BILE ZNANE TKIM, TIPKE MEHKIH FUNKCIJ, TODA NI BILO JASNO, KATERE NOVE IN OBSTOJEČE TIPKE IZBRATI. V DECEMBRU 1979 TO VPRAŠANJE ŠE NI BILO REŠENO. IMPULZ V DOLOČENO SMER SE JE POJAVIL PO OBISKU XEROXOVEGA RAZISKOVALNEGA SREDIŠČA V PALO ALTU, KJER JE BIL RAZVIJALCEM PODJETJA APPLE PRIKAZAN SISTEM SMALLTALK. TA SISTEM JE BIL NENAVADNO PRIJAZEN Z INTENZIVNO UPORABO Z ROKO POMIČNE "MIŠKE", KI JE POMIKALA KAZALEC NA TERMINALSKEM ZASLONU. APPLE SE JE KASNEJE ODLOČIL ZA UPORABO MIŠKE OB UPORABI TASTATURE. XEROX SVOJEGA RAZVOJA NI SKRIVAL, KER NI BIL POSEBNO ZAINTERESIRAN ZA PODROČJE OSEBNIH RAČUNALNIKOV. IN TAKO SE JE GLAVNI RAZVIJALEC SISTEMA SMALLTALK (L. TESSLER) PRIDRUŽIL PROJEKTU LISA.

ODLOČITEV ZA UPORABO MIŠKE JE ZAHTEVALA TRI MESECE IN TRI MESECE PO SPREJETI ODLOČITVI JE ŽE BIL NAREJEN PROTOTIPNI STROJ. OŽIČENA VEZJA, TISKANE PLOŠČE IN ŽICE SO PREKRIVALE TLA IN MIZE, TODA SISTEM JE ŽE DELOVAL. IZBIRA PROCESORJA IN DRUGIH VEZIJ JE BILA POGOJENA Z OBSTOJEČIMI INDUSTRIJSKIMI STANDARDI. NASLEDNJA FAZA, KATERE NALOGA JE BILA OBLIKOVANJE UPORABNIŠKE POVEZAVE - TO JE BIL VIDIK, OD KATEREGA JE BIL ODVISEN USPEH ALI PADEC LISE - JE ZAHTEVALA VELIKO MERO POSKUŠANJA, EKSPERIMENTIRANJA. V TEJ FAZI JE BILA RAZVITA VRSTA IZBOLJŠAV IN SPREJETE SO BILE ODLOČITVE O OSNOVNIH FUNKCIJAH PROGRAMSKE OPREME. POJAVILE SO SE ZAMISLI VRSTEČIH SE MENUJEV, POMIČNIH PALIC IN POSEBNIH PROCEDUR ZA ZAPLETENO UPORABO KURZORJA IN MIŠKE, KI BI NADOMESTILA FUNKCIJO TASTATURE. POSEBEN PROBLEM JE BILA TUDI ODLOČITEV O ŠTEVILU TIPK NA MIŠKI (IDEALNO ENA SAMA TIPKA Z DOLOČENIM STANJEM KURZORJA PRI TRENUTNEM STANJU ZASLONA).

UPORABA MIŠKE JE TAKO POSTALA OSREDNJE VPRAŠANJE RAZVOJA NOVEGA SISTEMA. SISTEM NAJ BI TAKO POSTAL LAŽJE UPORABLJIV PREDVSEM ZA ZAČETNIKA. K TEMU JE BILA DODANA ŠE ZAHTEVA, DA NAJ BI ZAČETNIK S SISTEMSKO PODPORO OBVLADAL UPORABO SISTEMA V POL URE. NAČRTOVANJE TE NOVE POVEZAVE JE ZAHTEVALO VZORČEVANJE UPORABNIKOV, PREDVSEM USLUŽBENCEV PODJETJA APPLE, MED NJIMI DIREKTORJEV, RAČUNOVODIJ IN DRUGIH BREZ IZKUŠENJ Z RAČUNALNIKI. OPAZOVANJA IN PREIZKUSI SO BILI OPREVLJENI POD NADZORSTVOM IZKUŠENIH PSIHOLOGOV IN SHRANJENI NA VIDEOTRAKOVE. V JESENI 1980 JE BILA UPORABNIŠKA POVEZAVA V GLAVNEM DOLOČENA IN RAZVIJALCI SO SE ZAČELI DOGOVARJATI Z USLUŽBENCI IZ MARKETINGA, KATERE VRSTE UPORABE NAJ BI BILE RAZVITE DO PRVE NAJAVE NOVEGA SISTEMA. TE OSNOVNE FUNKCIJE SO BILE IZBRANE TAKOLE:

PROCESIRANJE TEKSTA
GRAFIKA
FINANČNO MODELIRANJE

V TREH MESECIH JE PODJETJE PRIDOBILO 20 NOVIH PROGRAMERJEV S POVPREČNO DESETLETNIMI IZKUŠNJAMI: PRIŠLI SO Z VSEH STRANI ZDA IN V TEM RAZDOBJU SO SE PRIVAJALI SKUPINSKEMU DELU IN ZAČENJALI Z DELOM NA PROJEKTIH.

PRVA, GROBA VERZIJA APLIKACIJSKIH PAKETOV JE BILA KONČANA POLETI 1981. TA VERZIJA JE IMELA ŠE VRSTO NAPAK, VENDAR JE OMOGOČALA INTERAKTIVNO DELO NA SISTEMU. NAJTEŽJA NALOGA TE FAZE JE BILA INTEGRACIJA APLIKACIJSKIH PAKETOV S PREMETAVANJEM PODATKOV NAPREJ IN NAZAJ, S PRIKROJEVANJEM TEKSTOV IN USKLAJEVANJEM OPERACIJ. CILJ TE FAZE JE BILO ZASLONSKO KRMILJENJE FUNKCIJ Z UPORABO TKIM, NAMIZNEGA UPRAVLJAVNIKA. PRI TEM JE ŠTEVILO NALOG TAKO NARASLO, DA JE BILO POTREBNO UVESTI PERT METODO PLANIRANJA OB UPOŠTEVANJU VSEH VIDIKOV PROJEKTA. DIAGRAMI PERT METODE SO SE DOPOLNJEVALI IN SPREMINJALI VSAK TEDEN. TAKO JE BILA DOSEŽENA VISOKA MOTIVACIJA IN VZBURJENOST SLEHERNEGA RAZVIJALCA PRI IZDELAVI NOVIH PROGRAMSKIH PAKETOV. NA SAMEM ZAČETKU SE JE POJAVILO TUDI TIPIČNO (LAIČNO) NASPROTOVANJE MARKETINGA (ZNAČILNO ZA INOVATIVNE RAZVOJNE TIPE), KI JE PRENEHALO ŠELE Z OSEBNIM PREPRIČANJEM, DA SO NOVI PAKETI RESNIČNO UPORABLJIVI IN BOLJŠI OD DOSEDANJIH PRISTOPOV NA DRUGIH RAČUNALNIKIH TIPA APPLE. TAKO JE PROJEKT LISA POSTAL REGULARNA PRODUKTNA LINIJA IN KASNEJŠE MARKETINŠKE ŠTUDIJE SO LE POTRDILE NJEGOVO TRŽNO PRIMERNOST IN PROIZVODNO MASOVNOST.

ROJSTVO LISE SE JE ZAKASNILO ZA DVE LETI V PRIMERJAVI S PRVOTNO PLANIRANIM ROJSTVOM IN DOLOČENA UVIDEVNOST PODJETJA JE BILA PRAV GOTOVO POTREBNA V OBLOKI POLNE PODPORE MENAŽMENTA. INOVACIJ NAMREČ NI MOGOČE V CELOTI VNAPREJ OBVLADATI. KONFLIKTI MED MENAŽMENTOM IN NOVIM PROJEKTOM SO VSELEJ OSTALI NA NIVOJU OŽJEGA VODSTVA PROJEKTA IN NISO OBREMENJEVALI RAZVIJALCEV. ŠTEVILNE IZBOLJŠEVALNE ITERACIJE RAZVOJA SO ZNANTNO ZAKASNJEVALE DOKONČANJE RAZVOJA IN S TEM KONČNO OBLIKO NOVEGA PRODUKTA. NAJBOLJ KRITIČNO OBDOBJE JE NASTOPILO PRI INTEGRACIJI SISTEMA, TJ. PRI INTEGRACIJI APLIKACIJ V OKVIRU TKIM, NAMIZNEGA UPRAVLJALNIKA. TA POVEZOVALNI MEHANIZEM NAJ BI OMOGOČAL APLIKACIJAM DODELJEVANJE DOLOČENEGA PROGRAMSKEGA KODA V DOVOLJ KRATKIH ČASOVNIH REZINAH. V JULIJU 1982 SO SE APLIKACIJE KONČNO LAHKO IZVAJALE SIMULTANO V OKVIRU NAMIZNEGA UPRAVLJALNIKA IN UPORABNIK SE JE KONČNO LAHKO BREZ TEŽAV POMIKAL OD ENE APLIKACIJE K DRUGI. KLJUB TEMU USPEHU IN PROSLAVLJANJU NOVEGA PRODUKTA SE JE SKUPINA INŽENIRJEV ODLOČILA, DA SE VRNE NA DELO IN DA VSTAVI MED APLIKACIJE ŠE DOLOČENE IZBOLJŠEVALNE SEGMENTE. TA POSTOPEK JE ZAHTEVAL ŠE NEKAJ DODATNIH TEDNOV DELA, KO SO BILE ODSTRANJENE ŠE NEKATERE NAPAKE IN STRANSKI UČINKI.

KO JE BIL PROJEKT LISA V POLNEM TEKU, JE PODJETJE APPLE V JUNIJU 1980 UVEDLO NA TRŽIŠČE NOV PROIZVOD APPLE III, KI NI BIL PLOD DOZORELEGA DELA. PRITISK NA PROJEKT LISA SE JE ZARADI TEGA ŠE POVEČAL. V ZAČETKU LETA 1981 JE BILO NAREJENIH ŽE 50 UPORABNIH SISTEMOV LISA ZA RAZVOJ PROGRAMSKE OPREME. ISTOČASNO JE BILA NA TEH SISTEMIH OPRAVLJENA TUDI ANALIZA OBRATOVANJA V NAJKRITIČNEJŠIH POGOJIH (ČASOVNE OBLIKE, TEMPERATURA, LOGIKA, NALAGANJE ITD.) IN USTREZNE SPREMEMBE V NAČRTIH. TAKO SO SE IZBOLJŠEVALE LASTNOSTI ZANESLJIVOSTI IN PROIZVODNOSTI, KO JE BIL V FEBRUARJU 1982 IZDELEK PRIPRAVLJEN ZA PROIZVODNJO, TUDI PREIZKUŠANJE PROGRAMSKE OPREME JE BILO PODVRŽENO OSTRIM ZAHTEVAM. ODSTRANJENIH IN UGOTOVLJENIH JE BILO VEČ KOT 1000 NAPAK.

FORMAT NAMIZNEGA UPRAVLJALNIKA JE BIL NATANKO DOLOČEN TAKO, DA OMOGOČA UPORABNIKU MANIPULACIJO KURZORJA IN IZBIRO MED APLIKACIJAMI. NA SAMEM ZAČETKU JE BILO NEKAJ TEŽAV, VENDAR JE PODJETJE XEROX V JUNIJU 1981 POKAZALO SVOJO DELOVNO POSTAJO STAR, KI JE BILA TUDI UPORABNIŠKO PRIJAZNA, Z MOČNIM VPLIVOM SISTEMA SMALLTALK. XEROXOV MIKRORAČUNALNIK JE BIL MOČNO PODPRT Z GRAFIČNIMI SIMBOLI - IKONAMI - ZA ORGANIZACIJO SVOJIH FUNKCIJ. TAKO SO SE TUDI PRI APPLU ODLOČILI ZA IKONE, KI SO JIH PRIDRUŽILI V SISTEM ŽE SKORAJDA NA KONCU PROJEKTA. IKONE SO MAJHNE SLIKE SIMBOLIČNIH PREDMETOV IN SO NPR. ZBRANE V STOLPCU NA DESNEM ROBU ZASLONA. OBLIKA IKON JE IZREDNEGA POMENA, MORAJO BITI SKLADNE IN DOVOLJ ELEGANTNE IN TUDI LAHKO RAZUMLJIVE. MIŠKA IN IKONE SO BILE DOKAJ PODOBNE ZAMISLI XEOROXOVEGA STARA. SODELOVANJE MED APPLOM IN XEROXOM JE BILO TEMELJNEGA POMENA. PRODUKT LISA JE TAKO NA DOLOČEN NAČIN PODPIRAL PRODUKT STAR NA TRŽIŠČU, SAJ JE BILA MED NJIMA DOLOČENA PODOBNOST. TU PA JE BIL SKUPEN INTERES OBEH PODJETIJ.

PROJEKT LISA JE BIL V MARSIČEM POSEBEN. ŽE NA SAMEM ZAČETKU JE BIL V CELOTI LOČEN OD OSTALEGA DELA PODJETJA APPLE. TAKO SE JE PROJEKT IZVAJAL V POSEBNI, LOČENI STAVBI, DALEČ STRAN OD DRUGIH OBRATOV. PROJEKT SO SMATRALI KOT REVOLUCIONAREN IN GA ZATO NISO OMEJEVALI Z ŽE OSVOJENO (STAREJŠO) TEHNOLOGIJO. NOVOPRIDOBLJENI PROGRAMERJI SO BILI LOČENI IN DELEŽNI POSEBNIH PREDNOSTI V PODJETJU. V PODJETJU SE JE POJAVLJAL SKEPTICIZEM IN USLUŽBENCI SO SE VPRAŠEVALI, KDAJ SE BODO LAHKO POJAVILI PLODOVI TEGA NOVEGA PROJEKTA. VENDAR JE LISA POČASI POSTALA DEL APPLOVE KULTURE IN DELO NA NJEJ SE JE VZDRŽEVALO V TREH LETIH S PRODAJO SISTEMOV APPLE II IN III. SEVEDA PA ZDAJ OD PRODAJE LISE PRIČAKUJEJO PODPORO NOVIM RAZISKAVAM IN RAZVOJU.

LISA JE BIL MASIVEN PROJEKT, KI JE PORABIL VEČ KOT 200 ČLOVEK-LET ZA RAZVOJ; APPLE II IN III STA PORABILA LE 25 ČLOVEK-LET. NAJVEČ DELA JE BILO VLOŽENEGA V NAPOREN RAZVOJ PROGRAMSKE OPREME; TA VIŠINA JE DOSEGLA NPR. NAPOR, KI JE BIL POTREBEN PRI RAZVOJU PROGRAMSKE OPREME ZA MINIRAČUNALNIK VAX (PRODAJNI RAZRED 100000 $). VELIKO RAZVIJALCEV V LETIH IZVAJANJA PROJEKTA NI IZKORISTILO LETNIH DOPUSTOV. TAKO JE BIL TA PROJEKT PODOBEN ONEMU, OPISANEM V USPEŠNICI "SOUL OF A NEW MACHINE", KI OPISUJE RAZVOJ MINIRAČUNALNIKA (AVON BOOKS, 1982). V TEJ KNJIGI JE OPISANO "NORO IN NAPORNO USTVARJALNO DELO PRI OBLIKOVANJU VIZIJE, PROJEKTIRANJU IN RAZVOJU NOVEGA RAČUNALNIKA PODJETJA DATA GENERAL. PROJEKT LISA PREDSTAVLJA NENAVADNO OBLIKO SKUPINSKEGA NAPORA, KI JE UPOŠTEVAL LE TOLIKO STRUKTURE, KOT JE BILO NUJNO POTREBNO. VODJE PROJEKTA NISO IMELI NOBENIH POSEBNIH NAZIVOV. DOKUMENTACIJA PROJEKTA JE BILA ZELO PODROBNA, OD 50 DO 100 STRANI. NA KONCU PROJEKTA JE BILA TA DOKUMENTACIJA KRATKA IN NEFORMALNA; TERMINALSKI EMULATOR JE BIL OPISAN LE NA DVEH STRANEH. VENDAR JE BILO NA ZAČETKU PROJEKTA JASNO, DA JE LISA VZNEMIRLJIV PISARNIŠKI PRODUKT Z GRAFIKO IN LAHKO UPORABLJIV. PO ENEM LETU JE BILO VSEM JASNO, KAJ LISA POMENI - NASLEDNJO STOPNJO DOMISELNOSTI.

PROJEKT NI IMEL VODILNEGA GENIJA, OBSTAJALA PA JE VODILNA VIZIJA, SKUPNO DELJENA PODOBA PRODUKTA. VSE JE BILO PODOBNO USPEŠNEMU FILMU, V KATEREM SO BILE VLOGE NATANKO OPREDELJENE. V JESENI 1982 SE JE ŽE PRIPRAVLJALA PREDSTAVITEV LISE IN V KONFERENČNI DVORANI PODJETJA SO BILI POSTAVLJENI TRIJE SISTEMI LISA. ODZIVI OB PREDSTAVITVI SO BILI ZELO POZITIVNI IN TO JE BILA TUDI NAJVEČJA NAGRADA RAZVOJNI SKUPINI.

V ISKRI DELTI SMO DOŽIVELI TAK FILM V MALEM, V NAŠIH SLOVENSKIH RAZMERJIH PRI RAZVOJU NAMIZNEGA SISTEMA PARTNER. TUDI MI SMO IMELI PERT DIGRAME, KRATKE ROKE, PRITISKE MARKETINGA, NERAZUMLJIVE NABAVNE PROBLEME, PORODNE KRČE PRI PRENOSU IZDELKA V PROIZVODNJO, KONCENTRACIJO SPOSOBNIH KADROV, DOLOČENO IZOLIRANOST V SVOJEM OKOLJU ITD., PREDEN JE BIL PARTNER DOKONČNO ROJEN IN POSTAVLJEN V TRŽNI PROSTOR. O TEH DOŽIVETJIH IN IZKUŠNJAH PA BI VELJALO NAPISATI POSEBEN PRISPEVEK.

A.P.ŽELEZNIKAR

Canonov mikroračunalnik

Japonsko podjetje Canon, znano kot proizvajalec filmskih kamer, kopirnikov in kalkulatorjev, je najavilo proizvodnjo 8/16-bitnega mikroračunalnika. Ta računalnik posnema težnje novih proizvodov, ki sledijo konceptu osebnega računalnika PC IBM. Canonov računalnik AS100 bo uporabljal dva operacijska sistema, in sicer CP/M-86 in MS-DOS. Ločljivost grafike bo 640 x 400 točk za barvno ali monokromatično sliko. Sedem barv od 27 bo lahko prikazanih. AS100 uporablja procesor 8088 pri taktni frekvenci 4MHz. Standardni pomnilnik je 128k, razširljiv na 512k zlogov. Uporabnik lahko izbere magnetni medij med 5-colskim, 8-colskim upogljivim diskom in 5-colskim vinčestrskim diskom. Pomnilni obseg pri tej izbiri znaša od 1,28 do 8,14M zlogov (formatirano). Osem V/I integriranih vezij krmili vrata, ki se lahko uporabljajo v lokalnoh mrežah. Opcije Canonovega sistema pa so tele: akumulatorsko napajana ura realnega časa, razširitvena plošča za trdni disk z RS232C kanalom in paralelno podatkovno povezavo, pomnilniška razširitev 256/384k zlogov, posebni serijski, centronski in sinhroni V/I kanali. Cene začenjajo pri $3500 za črnobelo in končujejo pri $6000 za dva diska in barvno grafiko. Cena tiskalnika je $900.

A.P.Železnikar

Milijon računalnikov za dom

Angleško podjetje Sinclair Research je prodalo že milijon računalnikov za dom. Njegov glavni tekmec je podjetje Commodore, ki se je tudi približalo ali pa celo preseglo to število. Sinclair izdela nov računalnik vsake štiri sekunde na liniji podjetja Timex v škotskem Dundee. Doslej je Sinclair dobavil 130000 ZX80, 750000 ZX81 in 200000 ZX Spectrum. Commodore dobavlja množično sisteme VIC-20. K tem številom je treba dodati še 600000 sistemov, ki jih je Timex prodal na severnoameriškem tržišču. Tako ima danes Britanija več računalnikov na glavo kot katerakoli druga dežela. Cena teh sistemov je 40 funtov in njihova uporaba je dejansko množična.

A.P.Železnikar

Spremenjena vloga programerjev

Nova računalniška pismenost se danes začenja že v osnovni šoli in bo brzkone vplivala na prihodnjo vlogo poklicnih programerjev. Vse več je ljudi, ki spoznavajo pomembnost računalnikov v vsakdanjem življenju in se zato priučujejo njihovi uporabi in programiranju. Programiranje in način mišljenja, ki je za programiranje potreben, postajata del vsakdanjosti, tako kot sta npr. osebni prevoz in televizija. To pa bo odločilno vplivalo na prihodnje perspektive poklicnih programerjev. Če bo praktično vsakdo obvladal programiranje, zakaj bodo potem potrebni še poklicni programerji? Imeli bodo sicer še določeno vlogo (kot npr. poklicni vozniki), toda kakšen nov tip programerja bo še potreben?

Običajno se programiranje opravlja v visokih programirnih jezikih (Cobol, Fortan, PL/I). Ta vrsta programiranja je še vedno draga in rezultati ne dosegajo ciljev. Takšne so tudi izkušnje DOD (US Department of Defense), ki je za 9 projektov v vrednosti $ 6,8M lahko uporabilo le programsko opremo vredno $ 119k. Nadaljnih $ 198k je bilo lahko uporabljenih po spremembah, $ 6,5M vrednosti pa ni bilo nikoli dobavljenih ali pa nikdar uporabljenih. Poskus s projektom Ada naj bi izboljšal probleme z neuporabno programsko opremo. Podoben poskus je bil narejen že z jezikom Cobol, ki pa se ni uspešno iztekel. Pri Adi so se pojavile tudi zahteve za izdelavo avtomatiziranih programirnih pripomočkov. Komercialne težnje kažejo v smer tkim. produktivnostnih pripomočkov za programerje in za vodstvo programiranja, vključujoč operaterje

in direktorje. Unix je npr. stopnica navzgor k uporabniški prijaznosti, ni pa primeren za končne uporabnike. Unix je odličen sistem za programerja, z obilno podporo in primernimi lastnostmi. Podobne pripombe veljajo tudi za program Ditto (IBM DOS), ki bi lahko v rokah končnega uporabnika povzročil nepredvideno zmedo. V takem kontekstu programiranje ni umirajoča umetnost in zato lahko naraščajo zahteve po poklicnem programiranju.

Programiranje bo bržkone postalo nov način komuniciranja ob uporabi novih produktivnostnih pripomočkov z novimi programirnimi jeziki in drugimi avtomatičnimi mehanizmi za pisanje programov. Široka uporaba in pripomočki bodo oblikovali splošno kulturo računalniške pismenosti, pri čemer se bodo prav gotovo pojavili tudi avtorji in pasivni bralci napisanih programov, ki ne bodo potrebovali razumevanja določenih programirnih in računalniških podrobnosti.

Naloga prihodnjih programerjev bo predvsem v tem, da olajšajo to nalogo kolikor je le mogoče. Programeji bodo morali tako predvsem reševati probleme končnih uporabnikov in manj delovanja sistema. To bo zahtevalo, da postane delo sistemskih analitikov, višjih programerjev in industrijskih psihologov pomembnejše kot je danes. Kritika tkim. programerske tehnokracije napoveduje približevanje konca tkim. poklicnih programerjev. Namesto kodirajočega programerja, ki presedi tedne in mesece pri enoličnem programiranju, se bo pojavola potreba po uporabniškem izvedencu. Kakšna bo tedaj prihodnost novega programerja?

Potrebni bosta dve vrsti programerjev. Prvi tip programerjev bo uporabljal programirne generatorje za visoke jezike; to bodo posebni generirni programski paketi za oblikovanje aplikacij in za njih ne bo potrebno posebno znanje o tehnloogiji oziroma metodologiji. Pri tem ne bo več potrebno znanje o notranjem ustroju operacijskega sistema toda razumeti bo potrebno konceptualni okvir določenega generativnega paketa. Druga vrsta programerjev bo tehnološko in sistemsko usmerjena in njihova naloga bodo storitve in pomoč novim aplikativnim programerjem ali tkim. aplikacijska sinteza. Takšna povezava je potrebna že danes npr. med cobolskimi programerji in 'starimi' sistemskimi programerji. Obe skupini programerjev bosta izdatno uporabljali programirne generatorje, preizkuševalne pripomočke in nove sistemske programirne jezike.

Sistemski programeji bodo potrebni zlasti na področju novih mikroračunalniških sistemov, pri gradnji mrež in uporabi nove periferije, pri velikih bazah podatkov, elektronski pošti itd. Vloga programiranja in programerja bo tako bistveno spremenjena, vendar bo še vedno privlačna in vnemirjajoča - kot je danes.

A.P.Železnikar

****
* ČSSR RAM v Avstriji *
****

Vodstvo koncerna Siemens se je 11. julija 1983 odločilo, da vloži nadaljnih 665 milijonov ASch v beljaško tovarno integriranih vezij. S temi sredstvi bo realiziran projekt beljak II. Koroška tovarna polprevodnikov bo tako postala najsodobnejše središče za mikroelektroniko nemškega Siemensa. V kratkem bodo v tej tovarni začeli proizvajati 256k-bitne dinamične pomnilnike. S tem se bo Siemens pridružil vodilnim proizvajalcem polprevodniških pomnilnikov. Z dograditvijo tovarne bo na razpolago tudi 500 novih delovnih mest, tako da bo posodobljena tovarna zaposlovala skupno 1300 delavcev.

A.P.Železnikar

****
* Evropsko zaostajanje *
****

Evropska elektronska industrija čedalje bolj zaostaja. Polprevodniška evropska podjetja pokrivajo trenutno le 11% svetovnega tržišča, na področju integriranih vezij pa le 8%. Še bolj neugodna so razmerja za MOS integrirana vezja (5%) in za pomnilna vezja (2%). Zaradi tega sta bili pred kratkim sklicani dve konferenci direktorjev vodilnih podjetij, in sicer v Monte Carlu (Dataquest UK) in v Londonu (Financial Times). Oba sestanka sta potekala v ozračju kriznega stanja.

V analizi vzrokov evropskega mrtvila je bila poudarjena razpršenost evropskega tržišča, ki ga sestavljajo premajhni segmenti. V Evropi še ni tkim. elektronskih zgoščevalnin središč, kot je npr. Silicijska dolina pri San Franciscu, manjkajo pa tudi dajalci rizičnega kapitala. Nadaljna slabost je pomanjkanje zmogljivih potrošniških tržišč, kot je tržišče velike oboroževalne industrije v ZDA in tržišče zabavne elektronike na Japonskem. Razen tega imajo tako Američani kot Japonci bogate dodatne finančne vire: ministrstvo za mednarodno trgovino (MITI) na Japonskem ter obrambno ministrstvo in združenje kapitalistov v ZDA.

Diagnoza je tako tale: stanje polprevodniške evropske industrije je resno, toda ne brez upanja na izboljšanje. Referenti podjetij udeleženk (SGS-Ates, Motorola, Philips, Olivetti) so nakazali nekatere možnosti za izboljšanje tega stanja. Prva možnost je združevanje evropske polprevodniške industrije, kjer doslej ni bilo vidnih premikov. Glavno potrošniško tržišče naj bi postale telekomunikacije, toda pri tem bi bilo potrebno sprostiti protekcionistične omejitve znotraj posameznih evropskih držav in sprejeti skupne telekomunikacijske standarde. Razen tega naj bi vlade držav ustanovile institucijo, podobno japonskemu MITI in seveda uskladiti socialne in politične interese. Pri tem bi lahko uporabili svoje inovačije, kot so npr. integrirana vezja za digitalno televizijo (ITT-Intermetall) in polja logičnih vrat (Ferranti).

H.P.Železnikar

****
* Splošno o kritiki na področju *
* računalništva in informatike *
****

S to številko odpiramo v časopisu Informatica rubriko o kritiki dejavnosti na področju računakništva in informatike (kratko RiI). RiI sta danes v središču pozornosti tehnološkega, proizvodnega, izobraževalnega in splošno obveščevalnega razvoja. V časopisu Ekonomska politika (1620, letnik 32, 18.4.1983, str. 22, prvi stolpec, 4.odstavek) lahko preberemo tole:

"... Intenzivni tokovi razvoja informatike
i promene koje taj razvoj unosi
u privredu i društvo
ostavljaju nas po svemu sudeći
s a s v i m r a v n o d u š n i m .
Postaje drastično uočljiv
kontrast
opsednutosti sveta kvalitetom razvoja
i s p o k o j s t v a u našoj zemlji
koja kao da postaje
jedinstvena o a z a b e z b r i g e
u tehnološki zahuktalom svetu . ..."

Ta tekst se brez dvoma dotika tudi same kritike na področju RiI pri nas, ki je preprosto nismo razvili, nismo jo sistematično spodbujali in ščitili kot del življenskega, kulturnega in predvsem tudi tehnološkega napredka. Da bi to zadušljivo stanje lahko ohranjali, smo razvili dve protikritiški, značilni reakciji:

- ogorčenost kot posledico vsakršnje kritike (zbeganost, prepadenost, osuplost, strah, vzvišenost, nesposobnost, manipulativnost) in
- brezbrižnost (pomanjkanje občutljivosti, dojemljivosti, odgovornosti, željo po miru).

Kritiko naj bi razumeli predvsem kot spodbudo za boljše delo, kot pripomoček za odpravljanje pomanjkljivosti, kot normativno kulturno navado, kot nasprotje spokojne tihote in nesposobnosti in tistega brezbrižja, ki hromi misel in z njo povezano možno tehnološko in organizacijsko napredovanje na področju RiI.

Spoštovani bralci! Pripravite se na kritiko! Ko boste začeli, boste ugotovili, koliko ustvarjalnega napora, odgovornosti in strokovnosti je za kritičen prispevek potrebno. Izurite se v kritiki, negujte jo: tudi skozi njeno kakovostno jedro drži pot naprej - iz nesposobnosti, malomarnosti, površnosti, lagodnosti, lahkomiselnosti. Kritiki! Dobrodošli in pozdravljeni!

A.P.Železnikar

****
* Posvetovanje o mikroračunalnikih *
****

Naslovljeno: Organizacijski odbor posvetovanja
Informatica 83, Parmova 41, 61000
Ljubljana

V dneh 7. do 9. 6. 1983 ste organizirali posvetovanje o mikroračunalniški tehnologiji in uporabi. Organizacija posvetovanja in potek razstave nista bila na ustreznem nivoju. Predvsem pa me moti, da so napovedana predavanja drug za drugim odpadala oziroma niso bila izvedena. Vse skupaj vzeto, je bilo posvetovanje dokaj slaba reklama za informacijske sisteme, ki naj bi jih začeli uvajati (z vašo pomočjo?) v gospodarstvo.

Mislim, da smo udeleženci pošteno plačali kotizacijo za udeležbo in za pripravljene referate. Glede na to bi bilo pošteno, predvsem pa načelno pravilno, da bi udeležencem vrnili del plačane kotizacije. Razmislite!

Reklame za obvladovanje informacijskih sistemov pa si s tem posvetovanjem prav gotovo niste naredili.

Lep pozdrav!

Emil Sekne
Iskra
Industrija merilno-regulacijske in
stikalne tehnike Kranj, n.sol.o.,
Delovna skupnost administrativno
tehničnih služb Kranj,
Savska loka 4, 64001 Kranj
10. 6. 1983

****
* Odgovor na pismo tov. E. Sekneta *
****

Podpisani sem globoko zainteresiran za kritiko na področju RiI, za njen kakovostni nastanek in razvoj. Pismo tov. Emila Sekneta pozdravljam in z njegovimi ugotovitvami v zvezi s posvetovanjem se strinjam. Na teh osnovah naj mi bo dovoljeno, da pismo tov. Sekneta analiziram in da predmete kritike razširim in jih osvetlim še iz drugih vidikov.

Posvetovanje in razstava Informatica 83 prav gotovo nista bila na mogoči ustrezni višini: k tej višini prispevajo različni dejavniki, kot so organizatorji, razstavljavci, predavatelji in seveda udeleženci. Kakovost posvetovanja in razstave ne more biti odvisna samo od dobre volje organizatorjev, v marsičem je odvisna tudi od njihovih zmogljivosti, od vloženih naporov vrste sodelavcev, zlasti prostovoljcev. Vobče velja ugotovitev, da doslej nismo uspeli vzbuditi tisti prostovoljni interes strokovnjaka, ki je nujno potreben za uspešno organizacijo in izvedbo posvetovanja in razstave RiI. Pasivnost je postala manira, za pičlo aktivnost pa so odgovorni redki posamezniki, ki jim ostane prevelika količina dela in odgovornosti.

Posledica določene pasivnosti so tudi tisti primeri, ko se predavatelju ne zdi vredno, da bi svoje najavljeno predavanje predstavil. To seveda ni značilnost posvetovanj tipa Informatica, je pogost pojav na računalniških posvetovanjih pri nas. Tov. Sekne bi dejansko prispeval k zdravljenju takih pojavov, če bi poimensko navedel to zaporedje neopravljenih predavanj, tako da bi bila mogoča evidenca brezbrižnih predavateljev. Organizator posvetovanja naj bi v takih primerih zahteval opravičilo (dopis delovne organizacije, zdravniško potrdilo), saj malomarnost te vrste posredno škoduje ugledu strokovne prireditve.

Med akutne primere posvetovanj ne sodijo le malomarni predavatelji temveč tudi neresni, strokovno nezainteresirani udeleženci. Večkrat se pripeti, da predsednik sekcije in predava-

telj zaman čakata poslušalce, pa tudi predsedniki sekcij se velikokrat ne pojavijo, tako da skupina predavanj nima predvidenega vodenja razprav. Roko na srce! Razvili smo natanko to, kar smo imenovali in še imenujemo turizem posvetovanj v negativnem besednem pomenu. Taka je torej zavest udeležencev, znana in neizrečena.

Pred leti sem se odločno uprl in se še upiram posvetovanjem v turističnih središčih, kjer so bila posvetovanja tudi sredstvo za plačano razvedrilo, oddih ali celo za odsotnost na samem posvetovanju. Mednarodne letne šole NATO na področju RiI (v juliju, avgustu) pa so bile intenzivno delovne (predavanja, razprave, strokovni pogovori v prostem času) in poživljajoče, osamljene v gorskih naseljih, brez turistične infrastrukture, vendar izredno kvalitetne, seveda z izbiro udeležencev.

Slovensko društvo Informatika prireja posvetovanja zaradi potrebe po strokovni izmenjavi mnenj, dosežkov, zaradi prepotrebne vzgoje na zahtevnem tehnološkem področju. SDI nima reklamnega programa, potrebuje pa prostovoljne sodelavce pri izvajanju svojih akcij. Očitek v tej smeri je brez osnov (Sekne) in SDI nima realne moči, da bi na osnovi strokovnih argumentov vplivalo na uvajanje določenih informacijskih sistemov v gospodarstvo: to je lahko kvečjemu možnost daljne prihodnosti.

A.P.Železnikar

ABECEDNO AVTORSKO KAZALO ČLANKOV LETNIKA INFORMATICA 7 (1985)

****
*
* V A B I L O K S O D E L O V A N J U *
*
* V Č A S O P I S U I N F O R M A T I C A *
*
****

Časopis Informatica lahko redno izhaja le, če ima pravočasno na razpolago zadostno količino kakovostnih prispevkov. Kljub kriznim razmeram izhaja časopis Informatica redno in poskuša ohranjati tehnološko raven računalništva in informatike (RiI).

A v t o r j i vseh rubrik časopisa Informatica so vabljeni k sodelovanju - še posebej novi, mladi avtorji, ki lahko prispevajo s svojo strokovno in delovno zagnanostjo k kakovostnemu dvigu časopisa Informatica. Rubrike časopisa, ki so prejkoslej deficitarne so

-- pregledni članki
-- strokovni članki (razvoj, raziskave, proizvodnja)
-- kritični in polemični prispevki RiI
-- uporabni programi v visokih in zbirnih jezikih
-- novice in zanimivosti, ki so pomembne za naš nadaljni razvoj na področju RiI
-- vzgoja in izobraževanje na področju RiI
-- nova strokovna literatura (knjige, časopisi, tudi članki)

Časopis Informatika je de facto tudi jugoslovanski časopis in število prispevkov v srbohrvaškem jeziku je odvisno le od števila prispevkov, ki so uredniku na razpolago v tem jeziku. Zato pozivam predvsem mlajše avtorje s tega jezikovnega področja, da začno posiljati več svojih prispevkov na naslov uredništva.

Naš skupen, jugoslovanski, strokovni interes je, da razvijamo področje RiI, da na tem področju producirama materialno, vzgojno in publicistično, skratka da pozenemo in zaganjamo skozi RiI naše celotno proizvajalno delo - zlasti na tehnološko zahtevnih področjih.

Mladi avtorji! Pokažite se iz anonimnosti, dokazujte svojo strokovno sposobnost skozi prispevke v časopisu Informatica. Tako boste pomagali tudi svoji delovni organizaciji, svojemu ožjemu in širšemu strokovnemu okolju.

A. P. Železnikar

PARTNER

PARTNER — SODELAVEC V SODOBNEM POSLOVANJU

ISKRA DELTA uvaja na jugoslovansko in tuje tržišče mikroračunalnik PARTNER, rezultat lastnega razvoja. Ne nudimo vam samo strojno opremo, ampak sodobno, integrirano rešitev vašega finančnega, skladiščnega in materialnega poslovanja. Pri vsem tem pa je uporaba tako enostavna, da jo obvladate v enem dnevu.

PARTNER

Ažurnost in natančnost informacij sta pogoj za uspešno poslovanje vaše delovne organizacije. Sproten vpogled v finančno, skladiščno in materialno poslovanje omogoča ekonomičnost, konkurenčnost na trgu in vašo pot v izvoz. In s PARTNERJEM boste to dosegli! Postal bo orodje v rokah delavcev v komerciali, računovodstvu in neposredni proizvodnji. Primeren je za obdelavo v industriji, kmetijstvu, trgovini, prometu, turizmu, gostinstvu, obrti in ostalih dejavnostih.
Uvajanje PARTNERJA pomeni nov kvaliteten premik v gospodarjenju z družbenimi sredstvi. Samo na področju finančnega poslovanja in optimizacije zalog se vam ob uporabi PARTNERJA investicija povrne že v šestih mesecih.

STROJNA OPREMA

Mikroračunalnik PARTNER temelji na preizkušenem mikroprocesorju Z80 in ima pomnilnik s kapaciteto 128 KB. Kot zunanji pomnilnik služi diskovna enota tipa Winchester, z zavidljivo zmogljivostjo 10 MB. To so kapacitete, ki so jih pred leti imeli veliki računalniški sistemi, mi pa smo jih integrirali v naš namizni mikroračunalnik PARTNER. Za zaščito podatkov (back-up) in enostavnejši prenos, smo v konfiguracijo vključili še disketno enoto z gibkimi diski kapacitete 1 MB. Izpise rezultatov vaših obdelav na papir omogoča matrični tiskalnik, ki je prav tako vključen v standardno konfiguracijo.
Pri načrtovanju PARTNERJA smo upoštevali tudi ergonomska spoznanja. Zaslon je zelene barve in nebleščeč, zato je prijeten za oči. Premakljiva tastatura vam omogoča optimalni delovni položaj.

PARTNER - STANDARDNA KONFIGURACIJA

Centralna procesna enota

Pomnilnik 128 KB

Disketna enota 1 MB

Diskovna enota 10 MB

Tiskalnik TRS 835

Uporabniški programi — MIPOS

Dokumentacija

PROGRAMI

MIPOS je ime uporabniških programov za integrirano ali posamično obdelavo poslovnih podatkov. Integrirana obdelava podatkov občutno zmanjša obseg administrativnega dela, zmanjša možnosti napak ter omogoča, da so enkrat vnešeni podatki sočasno obdelani v vseh programih MIPOSA. Uporaba je tako enostavna, da jo lahko brez znanja računalništva in programiranja obvladate v enem dnevu. Tak način dela ima nešteto prednosti pred obdelavami na dosedanjih mehanografskih strojih in zaostalih velikih računalniških sistemih. MIPOS vključuje štiri programe:

— glavna knjiga
— saldakonti kupcev in dobaviteljev
— skladiščno poslovanje
— fakturiranje.

Ti programi se lahko uporabljajo tudi posamično, če to zahteva organizacija poslovanja.

PARTNER bo vaš pravi sodelavec pri delu. Preko zaslona boste z njim v stalnem dialogu, saj vas bo vodil skozi delo po principu vprašanj in odgovorov. Z njim boste z lahkoto povečali produktivnost.

TEHNIČNI PODATKI

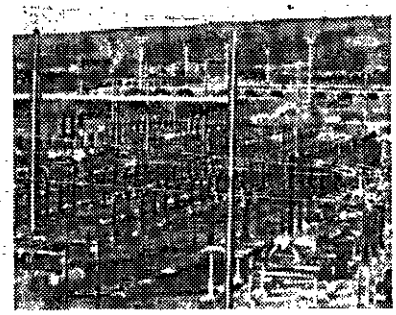
Procesor	Z80
Pomnilnik	128 KB RAM
Zunanji pomnilnik	Diskovna enota WINCHESTER, 5.25", kapaciteta 10 MB, prenosna hitrost 5 Mbit/s Disketna enota, 5.25", kapaciteta 1 MB, prenosna hitrost 250 Kbit/s
Zaslon	12", fosfor P 31 zelen, nebleščeč, 24 vrstic×80 znakov, točkovna matrika 5×7
Tastatura	premakljiva, dolžina kabla 0,7 m
Priključek za tiskalnik	RS-232C/CCITT V. 24, asinhroni, prenosna hitrost 175 do 9600 Baudov
Opcija 01	2 dodatna serijska asinhrona priključka RS-232C/CCITT V.24
Opcija 02	2 dodatna 8-bitna paralelna kanala
Delovna temperatura	16 °C—32 °C
Vlažnost zraka	20—80 %
Priključek v mrežo	220 V/50 Hz, 150 W
Širina	522 mm
Globina	655 mm s tastaturo
Višina	344 mm
Teža	22 kg

Pridržujemo si pravico do tehničnih sprememb.
PARTNER je zaščitni znak ISKRE DELTE.
©Tržno komuniciranje ISKRA DELTA (PARTNER, slov. V. 1—1983)

Iskra Delta
proizvodnja računalniških sistemov in inženiring, p. o.
61000 Ljubljana, Parmova 41
telefon: (061) 312-988
telex: 31366 YU DELTA

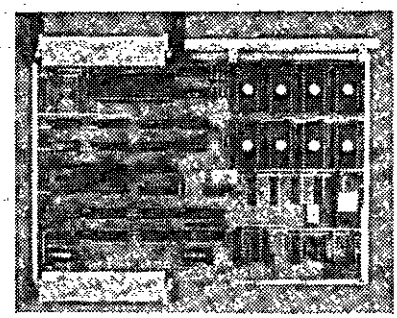
SISTEMI ZA ENERGETIKO

Ljubljana, Tržaška c. 2



DALJINSKO IN LOKALNO PROCESNO VODENJE Z RAČUNALNIKI
MINIRAČUNALNIKI IN MIKRORAČUNALNIKI V NAŠIH DOMAČIH
SISTEMIH DIPS-11 IN DIPS-85

RAZISKAVE, RAZVOJ, PROIZVODNJA, INSTALACIJA,VZDRŽEVANJE
SPECIALISTIČNO ŠOLANJE KUPČEVIH STROKOVNJAKOV
ELEKTROENERGETIKA, PLINOVODI, NAFTOVODI, VODOVODI,
INDUSTRIJA



SODOBNA TEHNOLOGIJA - NAŠ TEMELJ PRI RAZVOJNEM DELU
RAČUNALNIKI - NAŠI SOPOTNIKI NA POTI NAPREDKA
OBIŠČITE NAS IN SE PREPRIČAJTE

Že veliko let se ukvarjamo z raziskavami, razvojem in proizvodnjo sistemov za daljinsko in lokalno procesno vodenje. Temeljno vodilo našega delovanja na tem področju je slediti napredku v svetu in ga presajati na naša domača tla. Vedno smo zavračali nosilno licenčno povezovanje s tujimi firmami povsod tam, kjer smo jasno videli, da vodi v dolgoročno odvisnost in tehnično nazadovanje. Verjeli pa smo v moč lastnega marljivega dela in v ustvarjalnost naših delavcev ter z vstrajnim delom dosegli uspehe, katere nam lahko zavidajo neprimerno večji in bogatejši tekmeci.

Prav zaradi lastne poti in lastnega znanja smo s svojim razvojnim delom ves čas uspeli slediti najnovejšim tehnološkim dosežkom v svetu. V praktično življenje (računalniški nadzor v elektroenergetiki) smo vpeljali najsodobnejše mikroračunalnike.

Tako smo od prvih računalniških korakov pred več kot petnajstimi leti dospeli do sedanjih kompleksnih sistemov za procesno vodenje.