YU ISSN 0350-5596

informatica

JOURNAL OF COMPUTING AND INFORMATICS

YU ISSN 0350–5596

VOLUME 9, 1985 – No. 1

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

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

PRINTED BY: Tiskarna Kresija, Ljubljana

DESIGN: Rasto Kirn

CONTENTS

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 9, 1985 – št. 1

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

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

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

Pri financiranju časopisa sodeluje 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

V S E B I N A

INFORMATICA 1/85

REDUCIRANJE SIMPLEKSNE TABELE SPLOŠNE METODE SIMPLEKSOV

TALIB DAMIJ
JANEZ GRAD

UDK: 519.852.61

ISKRA DELTA, LJUBLJANA,
UNIVERZA EDVARDA KARDELJA V LJUBLJANI,
EKONOMSKA FAKULTETA BORISA KIDRIČA,
LJUBLJANA

Metoda simpleksov je iteracijski postopek, ki se odlikuje po enostavnosti in hitrosti, vendar pa je velik porabnik pomnilnika. V našem članku opišemo, kako je možno postopek simpleksov modificirati ter s tem zmanjšati zahteve po pomnilniku samo na prostor, ki ga potrebuje vhodna matrika neenačb.

REDUCTION OF THE SIMPLEX TABLEAU WITHIN THE GENERAL SIMPLEX METHOD. The general simplex method is known as a simple and fast iterative process but it as also known as a big computer storage consumer. In our paper we describe the necessary modifications of this method which enable us to reduce the size of simplex tableau to the size of the input matrix of the set of linear unequations.

1. UVOD

Metoda simpleksov je najbolj splošna metoda za numerično reševanje problemov linearnega programiranja; z njo lahko rešimo sleherni linearni program, kar je izrednega pomena za njeno uporabo pri različnih problemih optimizacije v gospodarstvu.

Metoda simpleksov je v bistvu iteracijski postopek, pri katerem v okviru vsakega iteracijskega koraka iz že znane bazne možne rešitve iščemo novo optimalnejšo možno rešitev. Odlikuje se po tem, da je zelo primerna za računalniško obdelavo, vendar pa je pri velikih sistemih tudi velik porabnik pomnilnika. Za metodo je namreč značilno, da temelji na tako imenovani simpleksni tabeli, v katero morajo biti razvrščene vse spremenljivke ter da je potrebno na vsakem koraku izračunati in shraniti v pomnilniku celotno tabelo. Tak način dela pa zahteva precej računalniškega pomnilnika, kar predstavlja osnovni problem uporabe te metode. To je tudi pomemben razlog, zaradi katerega se uporabniki raje odločajo za uporabo drugih metod, kot na primer revidirane metode simpleksov, ki so sicer bolj zapletene, vendar pa so manjši porabniki pomnilnika.

V tem prispevku opišemo, kako je možno postopek simpleksov modificirati ter s tem zmanjšati zahteve po pomnilniku samo na prostor, ki ga potrebuje vhodna matrika neenačb. Pri tem podajamo metodo simpleksov le toliko, kolikor je potrebno za spremljanje razvoja postopka reduciranja simpleksne tabele, podroben opis teoretičnih osnov te metode pa je opisan na primer v [2] in [3].

2. SPLOŠNA METODA SIMPLEKSOV

2.1. Definicija linearnega programa

Splošni problem linearnega programiranja za minimum namenske funkcije definiramo v matematični obliki takole: določiti je treba vrednosti spremenljivk $x_1,\ldots,x_s$, ki zadoščajo pogojem nenegativnosti

$$x_i \geq 0 \qquad (i=1,\ldots,s)$$

in linearnim pogojem

$$a_{11}x_1 + \ldots + a_{1s}x_s \geq b_1$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$a_{m1}x_1 + \ldots + a_{ms}x_s \geq b_m$$

tako, da ima namenska funkcija

$$c_1x_1 + \ldots + c_sx_s$$

minimum. Vsi koeficienti so realna števila.

Zgornji zapis linearnega programa ni uporaben za numerično reševanje, ker nastopajo v obliki neenačb. Take pogoje spremenimo v enačbe s privzemom dopolnilnih spremenljivk. Če odštejemo spremenljivke $x_{s+1},\ldots,x_{s+m}$ zaporedoma od levih strani neenačb, preidejo te neenačbe v enačbe. Dopolnilne spremenljivke upoštevamo tudi v namenski funkciji, kjer je $c_{s+1} = \ldots = c_{s+m} = 0$.

Da bi omogočili začetek numeričnega reševanja, vrinemo v vsako pogojno enačbo še po eno umetno spremenljivko, torej k levi strani enačb prištejemo zaporedoma spremenljivke $x_{s+m+1},\ldots,x_{s+2m}$. Umetne spremenljivke upoštevamo tudi v namenski funkciji, kjer je $c_{s+m+1} = \ldots = c_{s+2m} = M$.

Tako dopolnjen linearni program lahko zapišemo v matrični obliki takole:

potrebno je izračunati vektor x, ki zadošča neenačbi

$x \geq 0$

in enačbi

$Ax = b$

tako, da ima namenska funkcija

cx

minimum. Pri tem so:

$$x = \begin{vmatrix} x_1 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{vmatrix} \qquad A = \begin{vmatrix} a_{11} & \cdots & a_{1n} \\ \cdot & & \cdot \\ \cdot & & \cdot \\ a_{m1} & & a_{mn} \end{vmatrix} \qquad b = \begin{vmatrix} b_1 \\ \cdot \\ \cdot \\ b_m \end{vmatrix}$$

$$c = \begin{bmatrix} c_1 & \cdots & c_n \end{bmatrix}$$

kjer je $n = s+2m$.

Problem linearnega programiranja zapišemo lahko v vektorski obliki tudi takole:

potrebno je izračunati vektor x, ki zadošča neenačbi

$x \geq 0$

in enačbi

$x_1 p^1 + \ldots + x_n p^n = p^0$

tako, da ima namenska funkcija

cx

minimum. Pri tem so:

$$p^0 = \begin{vmatrix} b_1 \\ \cdot \\ \cdot \\ b_m \end{vmatrix} \qquad p^1 = \begin{vmatrix} a_{11} \\ \cdot \\ \cdot \\ a_{m1} \end{vmatrix} \quad \ldots \quad p^n = \begin{vmatrix} a_{1n} \\ \cdot \\ \cdot \\ a_{mn} \end{vmatrix}$$

Vektorji p^0, p^1, ..., p^s, p^{s+1}, ..., p^{s+m}, p^{s+m+1}, ... p^n so vektorji m-dimenzionalnega vektorskega prostora V^m. Med njimi sestavlja zadnjih m vektorjev sistem m linearno neodvisnih vektorskih enot. To so vektorji

$p^{s+m+1} = (1, 0, \ldots 0)$, ..., $p^n = (0, 0, \ldots 1)$

2.2. Določitev začetne baze

Dogovorimo se, da začetno bazo vektorskega prostora V^m določimo iz vektorjev umetnih spremenljivk ter vektorjev dopolnilnih spremenljivk, ki so vektorske enote. Ti vektorji torej sestavljajo bazno matriko B, ostali vektorji dopolnilnih spremenljivk pa naj sestavljajo matriko, ki jo označimo z D.

Potem lahko matriko A napišemo kot:

$$A = [F, D, E] \qquad (1)$$

Struktura vektorjev bazne matrike B je znana, saj je sestavljena iz vektorskih enot. Koeficienti baznih vektorjev v namenski funkciji so ravno tako znani in sicer so enaki M (kjer je M primerno velika vrednost) za umetne spremenljivke in 0 za dopolnilne spremenljivke.

Isto velja za matriko D, saj ima vsak vektor te matrike samo en element različen od nič in sicer (-1). Koeficienti vektorjev matrike D v namenski funkciji pa so enaki 0.

Iz tega sledi, da je struktura vektorjev, ki so v matrikah D in B ter njihovi koeficienti v namenski funkciji že vnaprej določeni s predpisanimi oblikami pogojnih enačb in neenačb linearnega programa.

Če najdemo način, s katerim lahko na začetku reševanja linearnega programa ugotovimo, kateri vektorji pridejo v sestavo začetne baze ter v ustreznem trenutku reševanja, kateri vektor matrike D je potrebno transformirati, potem lahko opustimo formiranje matrik D in B ter omejimo reševanje linearnega problema samo na matriko dimenzije m x s. Z računalniškega vidika pomeni tako zmanjšanje dimenzije simpleksne tabele zadostno zmanjšanje računalniškega pomnilnika, potrebnega za izvajanje programov, ki rešujejo probleme linearnega programiranja z uporabo splošne metode simpleksov.

V ta namen naj bo q vektor reda m, ki ima za svoje elemente simbole relacij v pogojnih enačbah in neenačbah obravnavanega linearnega programa ($\geq$, $=$, $\leq$). Ta vektor ima odločilno vlogo v modificiranem postopku, kajti s pomočjo tega vektorja lahko določimo vektorje matrik D in B ter njihove koeficiente v namenski funkciji.

Za določitev začetne baze je potrebno ugotoviti indekse baznih vektorjev ter njihove koeficiente v namenski funkciji.

Zato definirajno naslednje vrednosti:

g^1 naj bo število, ki nam pove, koliko je pogojnih neenačb v linearnem programu ($\geq$, $\leq$);

gq naj bo število pogojnih neenačb ali enačb oblike ($\geq$) ali ($=$);

d naj bo število vhodnih in dopolnilnih spremenljivk v sestavi matrike A; to vrednost dobimo kot

$d = s + gl$

n je število vseh spremenljivk matrike A; to vrednost dobimo kot

$n = s + gl + gq$

Z njimi moremo določiti indekse baznih vektorjev ter njihove koeficiente v namenski funkciji brez oblikovanja matrik D in B.

Nadalje definirajmo še naslednja vektorja reda m:

c^B, ki vsebuje koeficiente baznih vektorjev v namenski funkciji in

v^B, ki ima za elemente indekse baznih vektorjev.

Postopek kreiranja začetne baze je takle:

A) Izračunajo se vrednosti gl, gq, d in n.

B) Glede na vrednost elementa q_i, $i = 1,\ldots,m$, se izračuna indeks i-tega baznega vektorja in njegov koeficient v namenski funkciji z naslednjim algoritmom:

1) $i = 1$

2) q_i ima vrednost ($\geq$); potem je

$gq = gq-1$

$gl = gl-1$

$v_i^B = n-gq$

$c_i^B = M$

3) q_i ima vrednost ($\leq$); potem je

$gl = gl-1$

$v_i^B = d-gl$

$c_i^B = 0$

4) q_i ima vrednost ($*$); potem je

$gq = gq-1$

$v_i^B = n-gq$

$c_i^B = M$

5) $i = i+1$

6) Če je $i \leq m$, se postopek nadaljuje na točki 2).

S takim postopkom kreiranja začetne baze eliminiramo oblikovanje matrike B, kajti vse značilnosti te matrike so dosegljive z uporabo vektorjev c^B in v^B.

2.3. Izboljšanje možne rešitve

V izrazu (1) je A matrika dimenzije m x n. Vendar zaradi zgoraj prikazanega postopka eliminacije matrike B ter zaradi postopka eliminacije matrike D, katerega opis je podan v naslednjem, velja, da je A matrika dimenzije m x s, kar označimo z $A_{m,s}$. Kljub temu pa mora matrika A ohraniti vse značilnosti celotne simpleksne tabele, potrebne za reševanje linearnega programa.

V ta namen definirajmo naslednja vektorja:

c^A naj bo vektor reda s, ki ima za svoje elemente koeficiente vektorjev matrike $A_{m,s}$ v namenski funkciji,

v^A pa naj bo vektor reda s, ki ima za svoje elemente indekse vektorjev matrike $A_{m,s}$.

Iz prvotne baze dobimo novo bazo tako, da kak njen vektor zamenjamo s kakim drugim vektorjem, ki ga še ni v bazi.

Vzemimo, da odstranimo iz baze vektor p^r in ga zamenjamo z vektorjem p^k.

Za vsako možno rešitev ima namenska funkcija določeno vrednost. Pri prehodu iz prvotne v novo možno rešitev se vrednost namenske funkcije izboljša.

Definirajmo sledeče količine:

$$z_j = c_1^B a_{1j} + \ldots + c_m^B a_{mj}, \quad j = 1,\ldots,s$$

kjer z_j pomeni vrednost vektorja p^j, izraženega z baznimi vektorji, in

$$z_0 = c_1^B p_1^0 + \ldots + c_m^B p_m^0$$

kjer z_0 pomeni vrednost namenske funkcije, ki ustreza prvotni bazi.

V matriko A vpeljemo (m+1) vrstico, katere koeficienti $a_{m+1,j}$ so definirani z

$$a_{m+1,j} = z_j - c_j^A, \quad j = 1,\ldots,s$$

Razlika med prvotno in novo vrednostjo namenske funkcije je tem večja, čim večja je diferenca $z_k - c_k^A$. Ker želimo to razliko povečati, da dobimo čim boljšo novo možno rešitev, uvedemo v novo bazo tisti vektor p^j, ki mu ustreza največja pozitivna diferenca $z_j - c_j^A$. Vzemimo, da ustreza vektorju p^k največja diferenca

$$z_k - c_k = \max_j (z_j - c_j^A) \qquad (2)$$

kjer je $z_j - c_j^A > 0$ in $j = 1,\ldots,s$.

Če nobena od diferenc ni pozitivna, obstoječe rešitve ni mogoče več izboljšati, zato je to že optimalna rešitev.

2.4. Zamenjava baze

V bazo uvedemo tisti vektor p^k, pri katerem ima izraz (2) največjo vrednost, iz baze pa odstranimo vektor p^r, ki ga določimo tako, da vsako komponento p^0 delimo s homologno komponento vektorja p^k, pri tem pa upoštevamo samo pozitivne komponente vektorja p^k, najmanjši od teh ulomkov določa vektor p^r:

$$\theta = \min_i \frac{x_i}{a_{ik}} = \frac{x_r}{a_{rk}}$$

kjer $a_{ik} > 0$, za $1 \leq i \leq m$.

Če noben koeficient a_{ik} ni večji od 0, je rešitev neomejena.

Vrstico (m+1) vpeljemo tudi za vektor p^0, katerega koeficient je definiran s

$$p_{m+1}^0 = z_0$$

Vse koeficiente v simpleksni tabeli (koeficienti vektorja p^0 in matrike $A_{m+1,s}$) transformiramo po naslednjem transformacijskem zakonu:

a) $$x_r' = \frac{x_r}{a_{rk}}$$

$$a_{rj}' = \frac{a_{rj}}{a_{rk}}, \text{ kjer } j = 1,\ldots,s \qquad (3)$$

b) $$x_i' = x_i - \frac{x_r}{a_{rk}} \cdot a_{ik}$$

$$a_{ij}' = a_{ij} - \frac{a_{rj}}{a_{rk}} \cdot a_{ik}$$

kjer je $i = 1,\ldots,m+1$, $i \neq r$ in $j = 1,\ldots,s$.

Uvedba vektorja p^k v bazo in odstranitev vektorja p^r iz baze se izvrši tako, da se v v_r^B vnese vrednost iz v_k^A (indeks vektorja p^k), v c_r^B pa se vnese vrednost iz c_k^A (koeficient vektorja p^k v namenski funkciji). Prvotni vrednosti komponent v_r^B in c_r^B se predhodno shranita v postavkah, ki ju označimo z vbr in cbr. Torej je:

$vbr = v_r^B$

$cbr = c_r^B$

$$v_r^B = v_k^A$$

$$c_r^B = c_k^A$$

2.5. Zamenjava vektorjev matrike A

Matrika A ima na začetku numeričnega reševanja isto strukturo kot matrika F, torej predstavlja matriko vektorjev vhodnih spremenljivk. Prav tako pa sta na začetku numeričnega reševanja v^A in c^A vektorja indeksov vhodnih spremenljivk oziroma njihovih koeficientov v namenski funkciji. V matriki A in v vektorjih v^A ter c^A se v dani iteraciji vrši zamenjava vektorjev, njihovih indeksov ter koeficientov v namenski funkciji.

Smisel zamenjave vektorjev matrike A je v naslednjem:

1) izkoriščanje prostora vektorja p^k v matriki A, saj postane vektor p^k po uvedbi v bazo vektorska enota;
2) ohranitev značilnosti vektorjev dopolnilnih spremenljivk v simpleksni tabeli tako, da ob določenih pogojih dane iteracije uvedemo ustrezni dopolnilni vektor namesto vektorja p^k v matriko A;
3) ohranitev značilnosti vektorja p^r, ki ga odstranimo iz baze z uvedbo tega vektorja v matriko A na mesto vektorja p^k zaradi njegove možne vrnitve v bazo.

Izvršitev zamenjave vektorjev matrike A je odvisna od vektorjev q in c^B. S pomočjo teh vektorjev lahko sklepamo, če je potrebno izvršiti zamenjavo, poleg tega pa, kateri vektor je treba v dani iteraciji uvesti v matriko A.

Definirajmo vrednost ql, ki naj pove, koliko pogojnih enačb je v prvih r enačbah ali neenačbah linearnega programa. To vrednost potrebujemo za ugotavljanje indeksa ustreznega dopolnilnega vektorja, ki ga moramo uvesti v matriko A.

Preden realiziramo postopek zamenjave vektorjev matrike A, moramo zaradi degeneracije in zamenjave vektorjev shraniti značilnosti vektorjev matrike A, ki imajo po odstranitvi iz baze možnost ponovne vrnitve v bazo. To so vektorji vhodnih in dopolnilnih spremenljivk.

V ta namen vpeljemo vektor v reda d, ki ima za svoje elemente značilnosti teh vektorjev. Vrednost elementa v_j (j = 1, ..., d) mora zagotavljati hitro ugotavljanje mesta in strukture vektorja p^j v dani iteraciji.

Postopek oblikovanja vektorja v:

A) Za j = 1, ..., s velja

$$v_j = j$$

Gre za vektorje vhodnih spremenljivk, ki sestavljajo matriko A na začetku numeričnega reševanja.

p^j je vektor j-te vhodne spremenljivke, njegove komponente pa so shranjene v j-tem stolpcu matrike A.

B) Za j = s + 1, ..., d

1. j = s , r = 0
2. j = j + 1
3. r = r + 1
4. q_r ima vrednost ($\leq$); potem je

$$v_j = d + r$$

taka vrednost elementa v_j pomeni, da je p^j vektorska enota, katere r-ti element je enak 1 in se trenutno nahaja na r-tem mestu v bazi.

5. q_r ima vrednost ($\geq$); potem je

$$v_j = -(s + r)$$

negativna vrednost elementa v_j pomeni, da je p^j nebazni vektor dopolnilne spremenljivke x_j, njegov r-ti element pa je enak -1.

6. q_r ima vrednost (=); če r < m, se postopek nadaljuje na točki 3, sicer pa se konča.
7. če je j < d, se postopek nadaljuje na točki 2.
8. konec postopka.

Postopek zamenjave vektorjev matrike A:

A) Vrednost q_r je ($\geq$) in vrednost cbr (vrednost c_r^B pred zamenjavo) je enaka M.

Iz tega sledi, da je p^r umetni vektor, zato ga po odstranitvi iz baze zanemarimo, ker nima nobene možnosti vrnitve v bazo, namesto transformiranega vektorja p^k uvedemo ustrezni dopolnilni vektor p^d.

Vsi elementi vektorja p^d so enaki nič, razen r-tega elementa, ki je enak (-1). Koeficient p_{m+1}^d pa je definiran s

$p_{m+1}^d = - c_r^B$ (vrednost pred zamenjavo).

Za uvedbo p^d v matriko A je potrebno

- shraniti vrednost vektorja p^k; za ta namen definirajmo vektor p reda (m+1), kjer

$$p_i = a_{ik}, \quad \text{pri } i = 1, \ldots, m+1$$

- uvesti vektor p^d v k-ti stolpec matrike A, kjer

$$a_{ik} = 0 \quad , \quad \text{za } i = 1, \ldots, m, \; i \neq r$$

$$a_{rk} = -1$$

$$a_{m+1,k} = - \text{cbr}$$

- za r izračunati ql
- ugotoviti indeks vektorja p^d in njegov koeficient v namenski funkciji ter ažurirati vektor v s tem, da

$$j = v_k^A$$

$$u = s + r - ql$$

$$c_k^A = 0$$

$$v_k^A = u$$

$$v_u = k$$

$$v_j = d+r$$

B) Vrednost q_r je (=) in vrednost cbr (vrednost c_r^B pred zamenjavo) je enaka M.

V tem primeru ne izvršimo nobene zamenjave, ker je p^r umetni vektor, poleg tega pa ne obstoji v tej situaciji kak ustrezni dopolnilni vektor.

Za uvedbo vektorja p^k v bazo je treba ažurirati vektor v s tem, da

$$u = v_k^A$$

$$v_u = d+r$$

C) Za vse druge možnosti moramo ugotoviti naslednje:

1. vektor p^r je v sestavi matrike A; potem se zadovoljimo z ažuriranjem vektorja v s tem, da

   $v_{(vbr)} = vbr$

2. izvršiti moramo zamenjavo vektorjev, tako da uvedemo namesto vektorja p^k v matriko A vektor p^r zaradi možnosti njegove ponovne vrnitve v bazo.

   Vektor p^r kot bazni vektor ima strukturo vektorske enote, katere r-ta komponenta je enaka 1, koeficient p^r_{m+1} pa je enak nič.

   Za uvedbo p^r v matriko A je potrebno:

   - shraniti vrednost vektorja p^k, kjer

     $p_i = a_{ik}$, pri $i = 1, \ldots, m+1$

   - uvesti vektor p^r v matriko A s tem, da postavimo

     $a_{ik} = 0$, pri $i = 1, \ldots, m, i \neq r$

     $a_{rk} = 1$

     $a_{m+1,k} = o$

   - ugotoviti indeks vektorja p^r in njegov koeficient v namenski funkciji ter ažurirati vektor v s tem, da

     $j = v^A_k$

     $v^A_k = vbr$

     $c^A_k = cbr$

     $v_{(vbr)} = k$

     $v_j = d+r$

2.6. Degeneracija

Število θ oziroma p^r določimo z najmanjšim izmed ulomkov x_i/a_{ik}, $(a_{ik} > 0)$. Če je teh najmanjših ulomkov več, dobimo novo možno rešitev, ki ima manj kot m pozitivnih komponent in je zato degenerirana. V primeru degeneracije ni enolično določen vektor p^r, ki ga odstranimo iz baze.

Denimo, da pridemo v linearnem programu do degeneracije in sicer zato, ker sta vsaj dva od navedenih ulomkov enaka. Vzemimo, da sta enaka ulomka x_r/a_{rk} in x_s/a_{sk}. Zato primerjamo druga dva ulomka a_{r1}/a_{rk} in a_{s1}/a_{sk}; med njima izberemo tistega, ki je najmanjši. Če sta pa tudi ta dva ulomka enaka, še ne pridemo do odločitve in naprej primerjamo še naslednja dva ulomka a_{r2}/a_{rk} in a_{s2}/a_{sk} ter izberemo tistega, ki je manjši. Če še ne pridemo do odločitve, nadaljujemo primerjanje, dokler ne pridemo do nje.

Denimo, da ugotovimo s takim primerjanjem, da je ulomek y_r/a_{rk} manjši od ulomka y_s/a_{sk}. Število r določa zato enolično vektor p^r, ki ga odstranimo iz baze |2|.

Zaradi postopka zamenjave vektorjev, je potrebno v primeru degeneracije uporabiti vektor v, kjer element v_j $(j = 1,\ldots,d)$ rabi za ugotovitev značilnosti (indeks, struktura in mesto) vektorja p^j:

- indeks elementa v_j predstavlja indeks vektorja p^j,
- vrednost elementa v_j pa omogoča ugotoviti strukturo in pozicijo vektorja p^j s tem, da

a) če $v_j > d$

pomeni, da se vektor p^j nahaja v bazi, njegov r-ti element je enak 1 in vrednost indeksa r dobimo z

$r = v_j - d$

b) če $v_j \leq s$ in $v_j \neq k$

pomeni, da je vektor p^j v sestavi matrike A, njegova struktura pa daje v_j-ti stolpec matrike A

c) če $v_j < 0$

pomeni, da je p^j vektor dopolnilne spremenljivke x_j, njegov r-ti element enak -1 in vrednost indeksa r dobimo z

$r = -(v_j + s)$

3. POSTOPEK REŠEVANJA Z METODO SIMPLEKSOV

Postopek reševanja linearnega programa z metodo simpleksov obsega naslednje korake:

1) - Določitev začetne baze vektorskega prostora V^m z vključitvijo postopka kreiranja začetne baze (podpoglavje 2.2.)
   - Vstavitev začetnih vrednosti vektorjev c^A in v^A

2) Vključitev postopka oblikovanja vektorja v (podpoglavje 2.5.)

3) Izračun naslednjih vrednosti:

$z_j = c^B_1 a_{1j} + \ldots + c^B_m a_{mj}$, za $j = 1,\ldots,s$

$a_{m+1,j} = z_j - c^A_j$

$z_0 = x_{m+1} = c^B_1 x_1 + \ldots + c^B_m x_m$

$= c^B_1 b_1 + \ldots + c^B_m b_m = p^0_{m+1}$

4) Določitev vektorja p^k oziroma indeksa k:

$z_k - c^A_k = \max_{1 \leq j \leq s} (z_j - c^A_j)$, za $z_j - c^A_j > 0$

V primeru, da je $z_k - c^A_k \leq \varepsilon$, kjer je ε primerno majhno vnaprej predpisano pozitivno število, je obstoječa možna rešitev optimalna in iteracijski postopek se konča.

5) Če so vse vrednosti $a_{ik} < 0$, $(i = 1,\ldots,m)$, se iteracijski postopek konča in rešitev je neomejena.

6) Določitev vektorja p^r oziroma indeksa r:

$0 < \theta = \frac{x_r}{a_{rk}} = \min_{1 \leq i \leq m} (x_i/a_{ik})$, za $a_{ik} > 0$

Če doseže vrednost (x_i/a_{ik}) minimum pri več vrednostih indeksa i, je rešitev degenerirana.

7) Uvedba vektorja p^k v bazo (podpoglavje 2.4.)

8) Zamenjava vektorjev matrike A z uporabo postopka, opisanega v podpoglavju 2.5.

9) Transformacija tabele:

$$x_r' = \frac{x_r}{p_r}$$

$$a_{rj}' = \frac{a_{rj}}{p_r}, \quad \text{za } j = 1, \ldots, s$$

$$x_i' = x_i - \frac{x_r}{p_r} \cdot p_i, \quad \text{za } i = 1,\ldots,m+1, \; i \neq r$$

$$a_{ij}' = a_{ij} - \frac{a_{rj}}{p_r} \cdot p_i$$

10) Vrnitev k točki (3).

4. ZAKLJUČEK

Glavna značilnost prikazanega načina uporabe splošne metode simpleksov je, da v dani iteraciji vključuje v matriko A samo tiste vektorje, ki določajo vektor p^k v sledeči iteraciji.

Posledica tega je zmanjšanje računalniškega pomnilnika, potrebnega za hranjenje matrike A ter hitrejša transformacija simpleksne tabele v dani iteraciji. To pa omogoča ekonomičnejše in hitrejše obdelave problemov linearnega programiranja.

Na koncu naj omenimo primer obdelave linearnega modela s 100 spremenljivkami in 100 pogojnimi neenačbami. Obdelava tega modela s formiranjem celotne simpleksne tabele zahteva (128.512 bytov), medtem pa obdelava istega modela z uporabo reducirane simpleksne tabele zahteva (48.128 bytov) računalniškega pomnilnika.

5. LITERATURA

1) Grad, J., Strukturno programiranje linearnega programa, Ekonomska revija, Ljubljana, 28 (1977), str. 289-300.

2) Vadnal, A., Rešeni problemi linearnega programiranja, Knjižnica Sigma, Mladinska knjiga, Ljubljana, 1977.

3) Vadnal, A., Primjena matematičnih metoda u ekonomiji, Informator, Zagreb, 1980.

INFORMATICA 1/85

IBMOVSKI OSEBNI RAČUNALNIK PETRA II

ANTON P. ŽELEZNIKAR

ISKRA DELTA, LJUBLJANA

UDK: 681.3.06 Petra II

V tem nadaljevanju članka so opisani še preostali deli ibmovskega osebnega računalnika Petra, in sicer krmilno vezje za diskovne enote z upogljivimi diski, časovniško vezje za generiranje DMA zahtev, vezja serijskih V-I kanalov in paralelni krmilnik za V-I in za konfiguriranje računalnika.

Petra - An IBM-like Personal Computer II

In this part of the article several remaining elements of the IBM-like Personal Computer Petra are described: these are the floppy disk control circuit, the timer circuit for DMA request generation, circuits for serial channel I-O and the parallel controller for I-O and for computer configuring.

8. Logična shema lista 5

Logično vezje na listu 5 predstavlja krmilnik za diskovne enote z upogljivimi diski, ko imamo diskovno bralno in zapisovalno logiko in še vezje za oblikovanje DMA signala. Na sliki 6 je prikazana blokovna shema vezja na listu 5. Iz blokovne sheme so razvidna posamezna vezja, in sicer taktno vezje z različnimi frekvenčnimi in oblikovnimi odvodi, ki se oblikujejo iz osnovne frekvence 8MHz (za enojno in dvojno bralno-zapisno gostoto), diskovno bralno-oblikovalno vezje z napetostno krmiljenim oscilatorjem (VCO), enostavna vrata, diskovno zapisovalno-oblikovalno vezje, vezje za oblikovanje DMA signala, krmilno vezje med diskovnimi enotami in krmilnikom za te enote in seveda sam integrirani krmilnik (8272A).

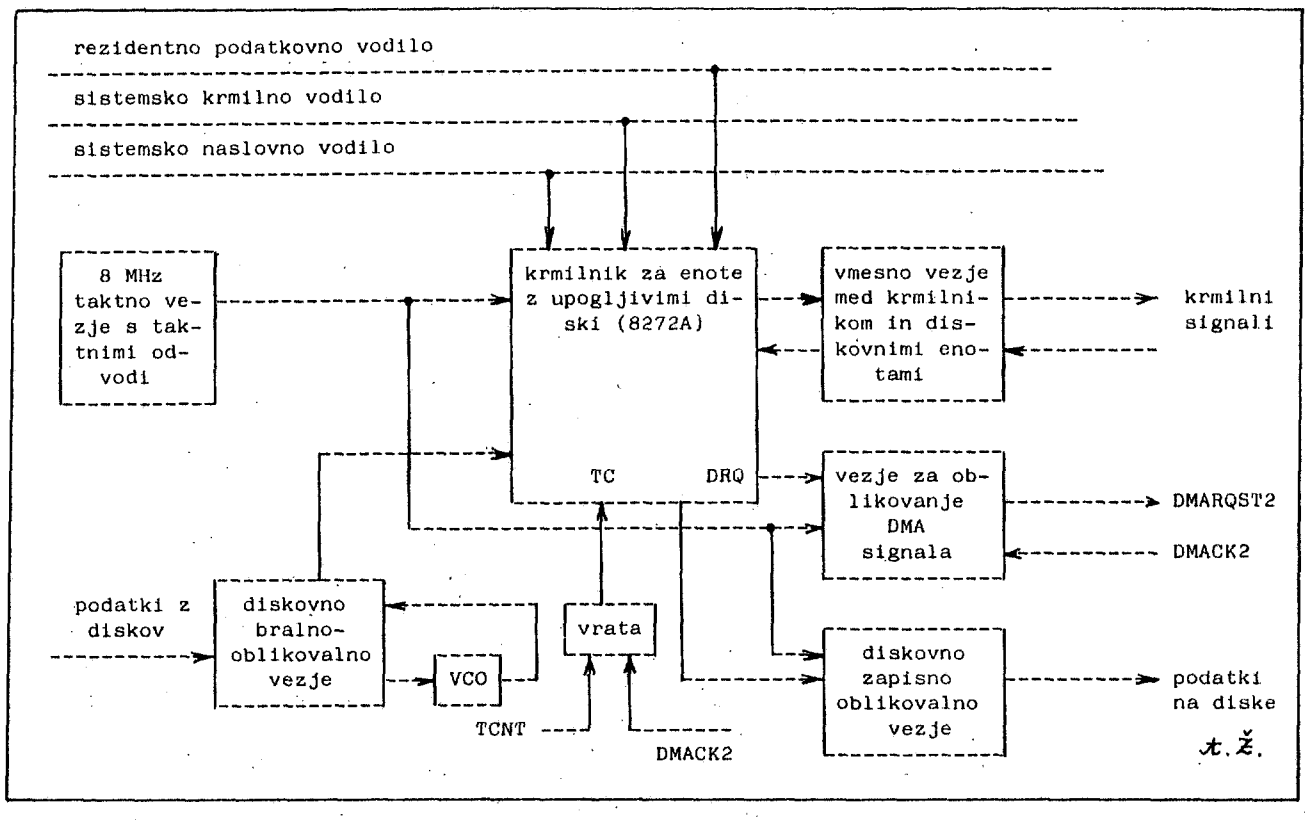
Krmilnik 8272A podpira uporabo do največ 4 diskovnih enot za upogljive diske premera 5,25 col (priključnica P11 s 34 nožicami) ali premera 8 col (priključnica P12 s 50 nožicami); priključnici za prvi in drugi primer sta prikazani na listu 6 (nadaljevanje lista 5). Tu imamo še 3 signale za vključevanje in izključevanje treh diskovnih motorjev (list 6) in 4 signale za aktiviranje bralnih-zapisnih glav (list 6).

Zapis na disk z enojno ali dvojno gostoto je opredeljiv programsko. Normalni diskovni format je združljiv s standardom IBM 3740 (za diskete premera 8") ali s standardom za IBM PC (za diskete premera 5,25", tj. s formatom IBM 5150), programsko pa ga lahko poljubno spreminjamo. Pri zapisu z enojno gostoto se uporablja metoda FM (frekvenčna modulacija), pri zapisu z dvojno gostoto pa metoda MFM (modificirana frekvenčna modulacija).

Osrednji element vezja na listu 5 je integrirano vezje 8272A, tj. krmilnik za upogljive diske (I21). Ta krmilnik je združljiv s krmilnikom uPD765 (NEC). Ta krmilnika podpirata 15 programskih ukazov, generiranje procesorske prekinitve, DMA podatkovni prenos in generiranje več krmilnih signalov, ki zmanjšujejo obseg dodatne logike. Krmilnik 8272A in DMA krmilnik 8237A-5 (I48) oblikujeta učinkovit diskovnopovezovalni podsistem.

Na listu 5 imamo vezja za oblikovanje taktnih signalov (8 in 4 MHz), za diskovno branje in zapisovanje, vmesno vezje in vezje za oblikovanje DMA krmilnega signala. Na listu 6 se nahaja nadaljevanje povezovalnega (vmesniškega) vezja in logika za vključevanje in izključevanje motorjev diskovnih enot.

Krmilnik 8272A potrebuje dva zunanja taktna signala, in sicer pravokotni obliki s frekvencama 4 ali 8 MHz in takt za podatkovni zapis z dolžino impulza 250 ns. Vhodni pravokotni taktni signal na nožici 19 krmilnika I21 se dobi kot odvod signala taktnega oscilatorja I10 s frekvenco 8 MHz. Pri uporabi disket premera 8"



Slika 6. Ta slika prikazuje v blokovni obliki logično vezje na listu 5. To vezje sestavljajo tale podvezja: taktno vezje z osnovno frekvenco 8 MHz, ki generira vse potrebne frekvenčne in oblikovne odvode, diskovni krmilnik (8272A), vmesna vezja k diskovnim enotam (bralno, zapisno, krmilno) in vezje za oblikovanje DMA signala.

---

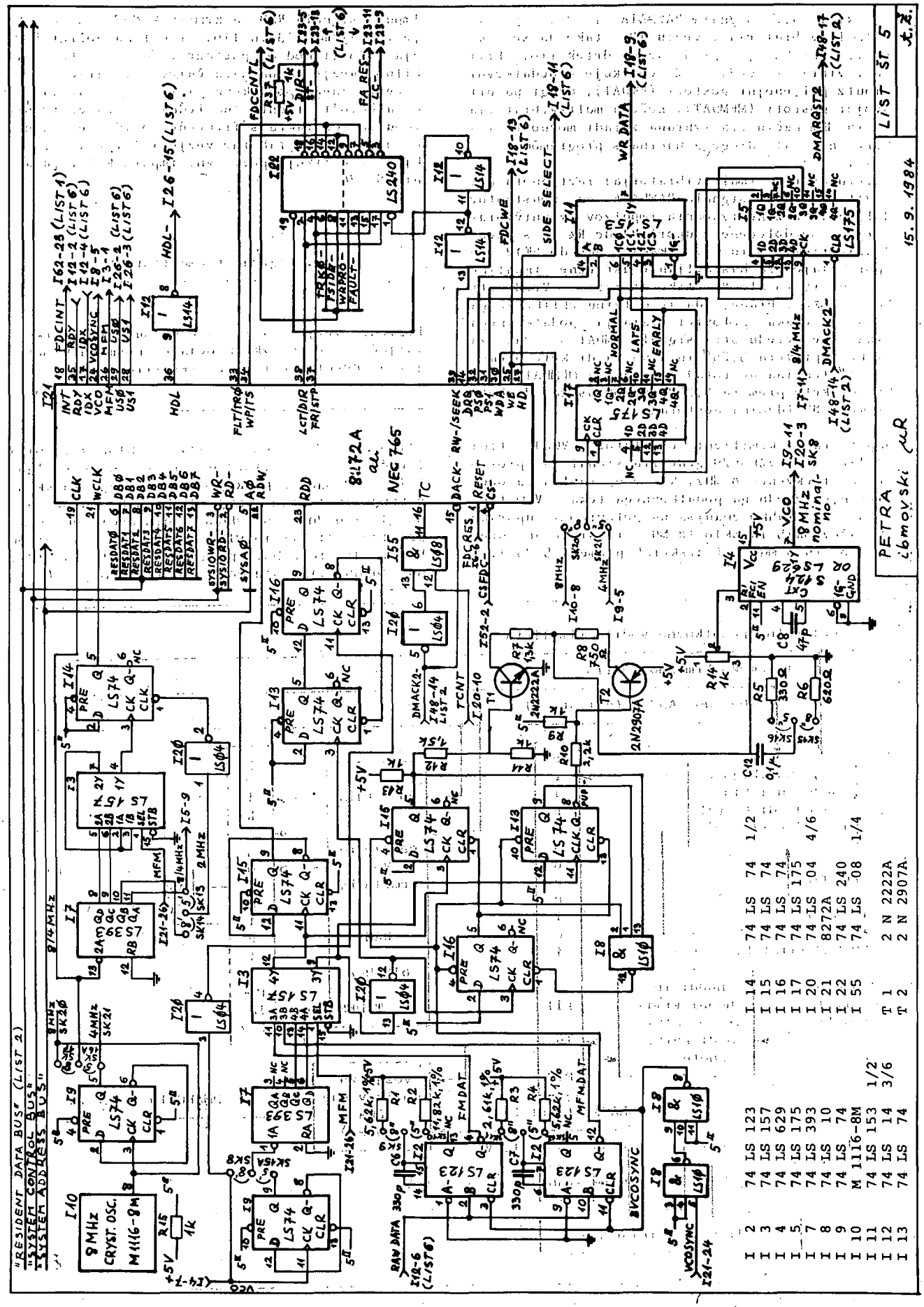
je zataknjen skakač SK16 in iztaknjen skakač SK17. Pri tem položaju obeh skakačev je taktni signal 8 MHz usmerjen neposredno na nožico 19. Pri uporabi disket premera 5,25" pa mora biti zataknjen skakač SK27 in iztaknjen skakač SK16, tako da prispe na nožico 19 taktni signal s frekvenco 4 MHz.

Ponavljalna frakvenca taktnih impulzov za podatkovno zapisovanje dolžine 250 ns znaša 1 MHz, 500 kHz ali 250 kHz in je odvisna od tipa diskovne enote in diskovnega formata. Multiplekser 13 izbere ustrezno frekvenco za zahtevano zapisovalno gostoto. Če je MFM signal, ki prihaja iz krmilnika 8272A v logično nizkem stanju, se izberejo frekvence za enojno zapisno gostoto sicer pa za dvojno zapisno gostoto.

Podatkovno zapisno logiko sestavljata četverno bistabilno vezje tipa D 74LS175 (117) in dekodirnik 4-v-1 74LS153 (111). Vezje 117 je pomikalni register, ki se pomika s taktno frekvenco zapisa z enojno ali dvojno gostoto in zagotavlja tkim. prekompenzacijo, ki je potrebna pri zapisu z dvojno gostoto. Dejanska vrednost (250 ali 125 ns) je odvisna od uporabljene diskovne enote in se izbere s skakačema SK20 in SK21.

Podatkovna bralna logika je dokaj zapletena še posebej pri metodi MFM. Pri tem se uporablja PLL vezje (fazno krmiljena zanka) za dekodiranje signalov z dvojno gostoto. Diskovni krmilnik 8272A (121) potrebuje dva vhodna signala, in sicer RDD in RDW na nožicah 23 in 22, ki nastajata iz surovo prebranega signala z diska (raw data) prek vmesnika 112 (list 6). RDD signal je sestavljen iz pozitivnega impulza za vsako negativno signalno fronto magnetnega pretoka in označuje ali taktni ali podatkovni bit. RDW signal posreduje krmilniku 8272A stanje tkim. podatkovnega okna (tj. časovni interval, v katerem se impulz lahko pojavi ali ne pojavi), ki ga krmilnik 8272A uporablja za določevanje ali podatkovnega ali taktnega bita.

Krmilnik 8272A daje dva izhodna signala, in sicer VCOSYNC (nožica 24) in MFM (nožica 26), ki poenostavljata realizacijo PLL vezja za obnavljanje signalov. Signal VCOSYNC je visok, če se je z diska prebral veljaven signal in se uporablja za aktiviranje PLL logike (vezje 18, nožica 5). V praznem območju (mesto na disketi, kjer podatki niso zapisani, npr. pri diskovni identifikaciji in diskovnih podatkovnih poljih) je signal VCOSYNC nizek in PLL vezje ni aktivirano. Razen tega je signal VCOSYNC lahko visok le, če je bila bralno-pisalna glava pritegnjena in se je čas pritegnitve iztekel. Signal MFM iz krmilnika 8272A pa kaže v visokem stanju, da je bil krmilnik 8272A programiran za delovanje z dvojno gostoto (bralno in pisalno). Ko je signal MFM nizek, imamo enojno bralno-pisalno gostoto. Ta signal omogoča skupaj z oblikovalno (obnovitveno) logiko programsko izbiro enojne ali dvojne gostote.


"RESIDENT DATA BUS" (LIST 2)
"SYSTEM CONTROL BUS"
"SYSTEM ADDRESS BUS"
8:172A
ALT. NEC 765
9 MHz
CRYST. OSC.
M 1116-9M
LS 123
LS 157
LS 629
LS 175
LS 393
LS 10
LS 74
M 1116-8M
LS 153
LS 14
LS 74
74 LS 123
74 LS 157
74 LS 629
74 LS 175
74 LS 393
74 LS 10
74 LS 74
M 1116-8M
74 LS 153
74 LS 14
74 LS 74
I 14 74 LS 74 1/2
I 15 74 LS 74
I 16 74 LS 74
I 17 74 LS 175
I 20 74 LS 04 4/6
I 21 8272A
I 22 74 LS 240
I 55 74 LS 08 1/4
T 1 2 N 2222A
T 2 2 N 2907A
PETRA
Lbmovski
15. 9. 1984
LIST ST 5
J. Z.

Visoki impulzi signala RAWDATA (12-2-10) prožijo multivibratorja v vezju 12, tako da se impulzi oblikujejo pred fazno detekcijo. Prvi multivibrator vezja 12 oblikuje podatkovni impulz pri enojni gostoti (FMDAT), drugi pa pri dvojni gostoti (MFMDAT). Ločena multivibratorja FM in MFM način sta izbrana zaradi možnosti izbire enega ali drugega načina s programom.

Univibratorja (multivibratorja) raztegneta ali zožita impulz signala RAWDATA na konstantno dolžino. Dolžine izhodnih impulzov iz univibratorjev so določene z upori R1 do R4 in s kondenzatorjema C6 in C7. Skakači SK9 do SK12 se uporabljajo za nastavljanje ustrezne impulzne dolžine (5,25 ali 8 colske diskete). Vrednosti RC so tako izbrane, da je dolžina oblikovanega impulza enaka polovici trajanja podatkovnega okna. Te vrednosti signala FMDAT so 2us za diskete premera 5,25" in 1 us za diskete premera 8" (enojna gostota) in signala MFMDAT 1 us za diskete premera 5,25" in 500 ns za diskete premera 8" (dvojna gostota).

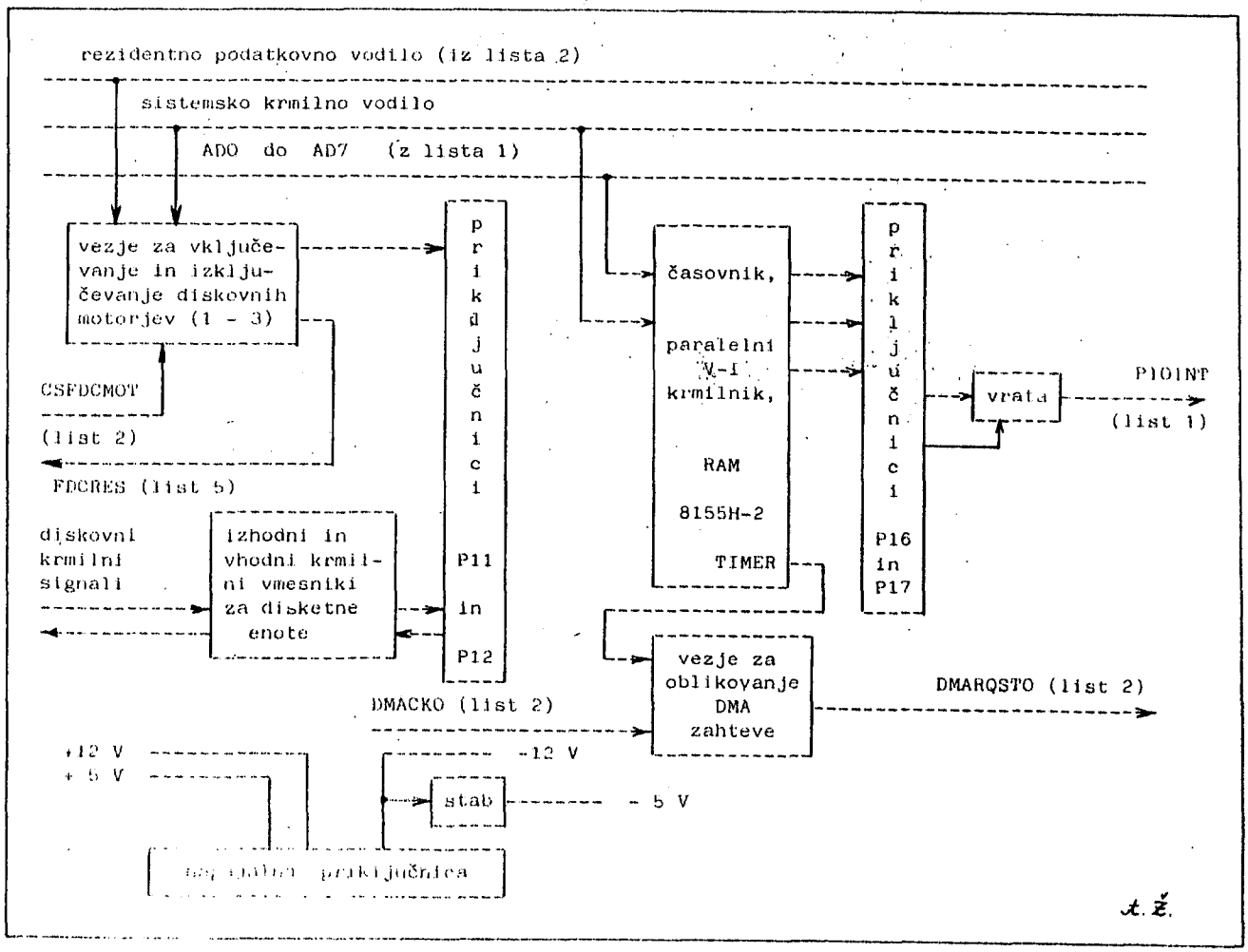
Napetostno krmiljeni oscilator (VCO) tipa 74S124 ali 74LS629 generira signal s prosto tekočo frekvenco 8 MHz, ki se uporablja za ocenjevanje vhodnega podatkovnega toka. VCO frekvenca se lahko zmanjša na polovico z uporabo skakačev SK15 in SK16 (8 MHz pri disketah premera 8" in 4 MHz pri disketah premera 5,25").

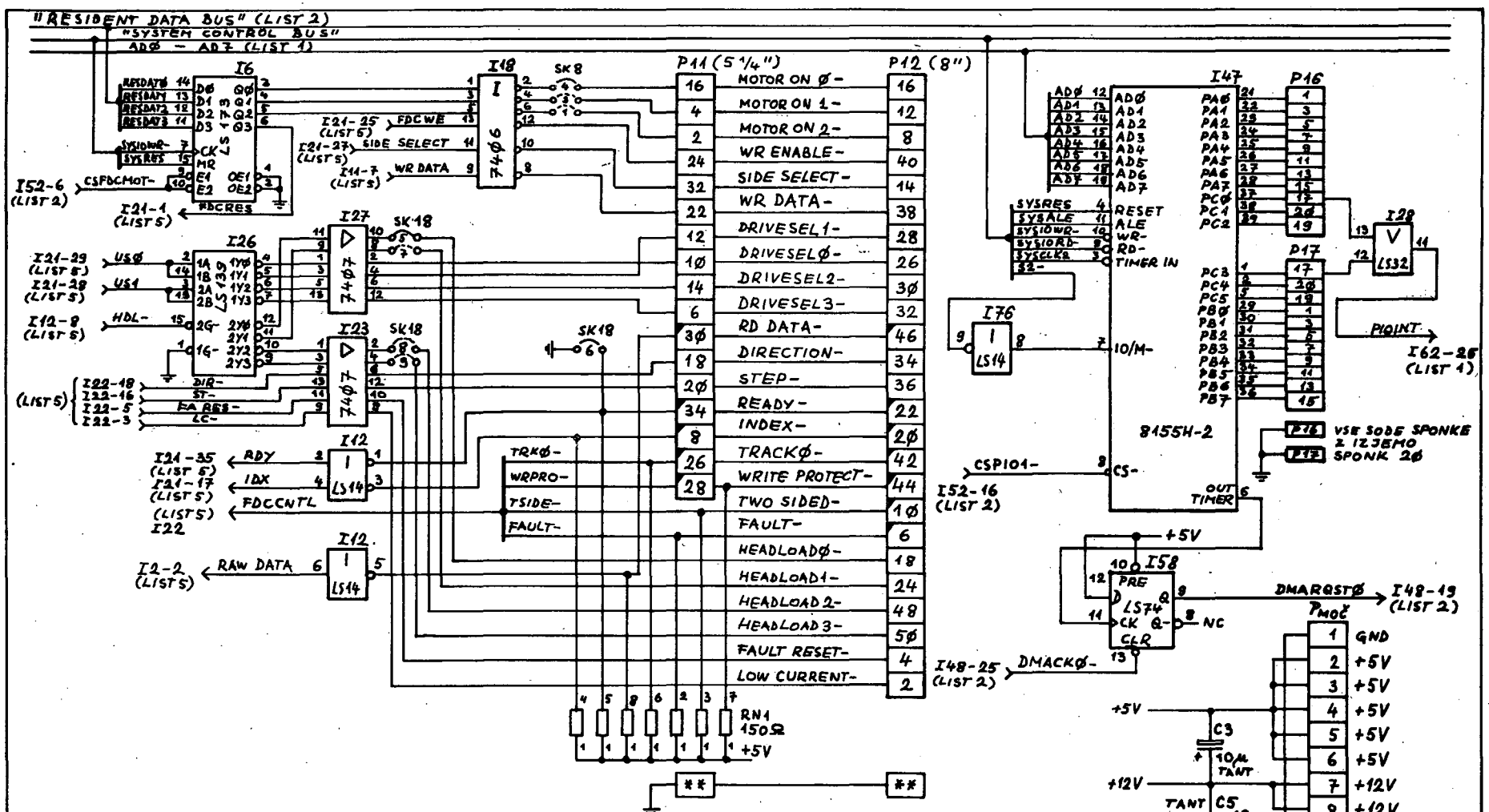
Impulz signala RDD za krmilnik 8272A se oblikuje z vezjema 113 in 116. Signal na nožici 5 vezja 113 (izhod Q) postane visok, ko to bistabilno vezje zazna naraščajočo signalno fronto invertiranega podatkovnega impulza z diska. Pri naraščajoči fronti naslednjega invertiranega impulza VCO takta s frekvenco 8 MHz se izhod Q prenese v bistabilno vezje 116 (prek 120-3), kjer se oblikuje pozitivni impulz RDD za krmilnik 8272A.

9. Logična shema lista 6

Logično vezje na listu 6 vsebuje krmilnik za vključevanje in izključevanje enosmernih motorjev diskovnih enot in za resetiranje krmilnika

Slika 7. Ta slika prikazuje v blokovni obliki logično vezje na listu 6. To vezje sestavljajo tale podvezja in priključnice: vezje za vključevanje in izključevanje diskovnih motorjev (1 do 3), izhodni in vhodni diskovni vmesniki, priključnici P11 in P12, krmilnik s časovnikom, V-I vezjem in RAMom, priključnici P16 in P17 in napajalna priključnica s stabilizatorjem (-5 V)





I 6	74 LS 173	
I 12	74 LS 14	2/6
I 18	7406	
I 23	7407	
I 26	74 LS 139	
I 27	7407	
I 28	74 LS 32	1/4
I 47	8155H-2	
I 58	74 LS 74	1/2
I 76	74 LS 14	1/6
ST 1	7905	

vhodna sponka nn

PETRA ibmovski µR 13.10.1984

LIST ŠT. 6 št. ž.

8272A prek dela rezidentnega podatkovnega vodila (4 biti), krmilnik za izbiro diskovnih, smeri, koraka in ostalih signalov, ki prihajajo in odhajajo iz diskovnega krmilnika 8272A, nadalje vezje za oblikovanje DMA zahteve in kombinirani krmilnik, časovnik in RAM pomnilnik 8155H-2. Na tem listu se nahajajo tudi sheme priključnic P11, P12, P16, P17 in napajalne priključnice. Na sliki 7 je prikazana blokovna shema logičnega vezja na listu 6.

Diskovno vmesno vezje vsebuje tri krmilne linije za vključevanje in izključevanje motorjev, in sicer MOTOR ON 0-, MOTOR ON 1- in MOTOR ON 2- (priključnica P11, sponke 14, 4 in 2 na listu 6). Ti signali se oblikujejo v registrskem vezju tipa D (16, 74 LS 173). Štirje registri v tem vezju se naslovijo kot V-I naprava z heksadecimalnim naslovom 0A0H.

Izhod Q0 vezja 16 krmili linijo MOTOR ON 0. Za vključitev motorja se v Q0 vpiše logična enica, za izključitev pa logična ničla. Podobno krmilita izhoda Q1 in Q2 vezja 16 liniji MOTOR ON 1 in MOTOR ON 2.

Linija MOTOR ON 0- končuje na sponki 16 priključnic P11 in P12. Uporaba tega signala za vključevanje in izključevanje motorja prek te sponke je standardna. Ostali dve krmilni liniji pa nista standardni in omogočata dodatno krmiljenje, če se pojavi potreba. Običajna uporaba teh linij je npr. linija MOTOR ON 0- za diskovno enoto A, linija MOTOR ON 1- za diskovno enoto B in linija MOTOR ON 2- za diskovni enoti C in D. Za vse tri linije obstajajo skakači, ki omogočajo aktiviranje teh linij po potrebi.

Vmesniško krmilno vezje vsebuje vso vhodno in izhodno logiko z izjemo opisanih signalov za vključevanje in izključevanje motorjev; vsi ti signali prihajajo iz ali odhajajo v diskovni krmilnik 8272A. Vsi izhodni signali se ojačujejo v močnostnih invertorjih tipa 7406 in v ojačevalnikih tipa 7407 z odprtimi kolektorji. Vse vhodne signalne linije so zaključene z upori vrednosti 150 Ohm na napajalno napetost +5 V in oblikovani z vezji 74 LS 14 (Schmittovi prožilniki).

Linija RW- SEEK krmilnika 8272A na listu 5 se uporablja za multipleksiranje osmih vmesnih signalov na štiri sponke krmilnika 8272A. Ko je krmilnik 8272A v iskalnem načinu (signal RW-SEEK je nizek), ima nožica 19 vezja 122 na listu 5 nizko vrednost. To povzroči, da signala TRACK0- in TWOSIDED- dosežeta nožici 33 in 34 krmilnika 8272A na listu 5 in signala DIRECTION- in STEP- z nožic 38 in 37 se oddata v diskovne enote.

Ko je krmilnik 8272A v načinu branja-pisanja (signal RW- SEEK je visok), ima nožica 1 obrnitvenega vmesnika 122 na listu 5 nizko vrednost. To omogoči signaloma WRITE PROTECT- in FAULT-, da dosežeta nožici 33 in 34 krmilnika 8272A in signaloma FAULT RESET- in LOW CURRENT- z nožic 37 in 38 krmilnika 8272A, da se oddata v diskovne enote. Poseben upor R37 na liniji ST- (list 5) zagotavlja, da se ne pojavi napačen ukaz STEP- na diskovnih enotah.

Krmilnik 8272A daje dva krmilna signala za izbiro 4 diskovnih enot; to sta signala US0 in US1 na nožicah 29 in 28. Ta dva signala se pošiljata na demultipleksor 126 (74 LS 139, list 6), ki je tipa 2-v-4. Vezje 126 izbere ustrezno diskovno enoto z nizkim signalom na liniji DRIVESEL n-. Signala US0 in US1 pa prihajata še v drugi del demultipleksorja in povzročata ustrezen nastanek signalov HEADLOAD n- (priključnica P12 na listu 6 za 8 colske diskovne enote).

Izhod HD (izbira glave) krmilnika 8272A na nožici 27 se uporablja pri dvostranskih diskovnih enotah. Ta signal se lahko uporabi za izbiro ene izmed obeh glav (pri dvostranskih enotah). Pri ibmovskih računalnikih so v uporabi tako eno- kot dvostranske diskovne enote kot tudi dve gostoti stez, in sicer 48 in 96 tpi. Za načine zapisa pi velja v vseh primerih dvojna gostota zapisa.

Vhodna signala READY- in INDEX- se oblikujeta v vezju 74 LS 14 in prihajata na nožici 35 in 17 krmilnika 8272A. Linija READY- se lahko s skakačem SK18-6 ozemlji, če uporabljena diskovna enota nima tega signala. Indeksni impulz pa se pojavi pri vsakem zavrtljaju diskete.

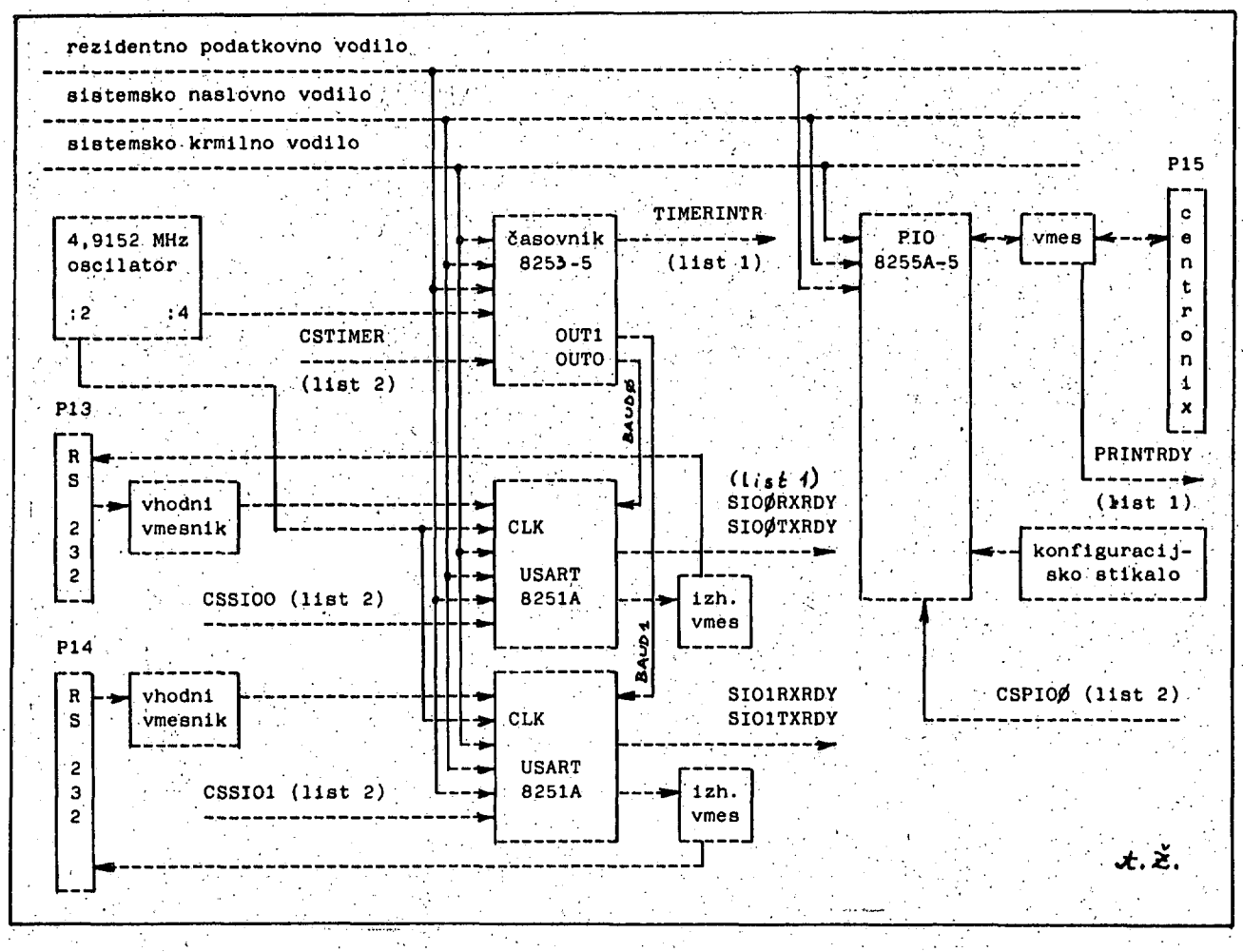
Vezje za oblikovanje posebne (osveževalne) DMA zahteve je prikazano na listu 6. Ta DMA kanal (z indeksom nič) se uporablja za generiranje zahteve DMARQST0. Časovnik 8155H-2 je programsko tako inicializiran, da v dovolj kratkih časovnih presledkih (vsakih 15,1 us) generira signal TIMER OUT na nožici 6 vezja 147 (list 6). Ta taktni impulz sproži nastanek signala DMARQST0, ki se posreduje DMA krmilniku (8237A-5, 148, nožica 19, list 2).

Vezje 8155H-2 na listu 6 vsebuje dvoje V-I vrat, 14-bitni števnik-časovnik in 256 zlogov RAM pomnilnika; ta pomnilnik se seveda lahko uporabi za posebne namene, npr. s posebnim uporabniškim programom. Dvoje 11-bitnih vrat (priključnici P16 in P17, list 6) se lahko po potrebi uporabi za komuniciranje z okolico. Uporaba teh vrat je odvisna od programske inicializacije vezja 8155H-2.

Končno imamo na listu 6 še napajalno priključnico in napetostni stabilizator 7905. Pri tem velja opozorilo, da smemo imeti na kontaktno nožico največjo tokovno obremenitev 2,5 A. Tako smemo imeti v našem primeru obremenitev 12,5 A za napetost +5 V.

10. Logična shema lista 7

Slika 8 prikazuje v blokovni obliki logično vezje na listu 7. To vezje je sestavljeno iz osnovnega oscilatorja s frekvenco 4,1952 MHz, ki generira dve derivaciji: ena se uporablja kot vhodni takt za časovnik 8253-5, druga pa kot vhodni takt za vezji tipa USART (8251A); nadalje imamo dve vezji USART z vhodnimi in izhodnimi vmesniki na priključnici tipa RS-232 (P13 in P14), časovnik 8253-5 in V-I paralelni krmilnik 8255A-5 z vmesnikom na priključnico tipa Centronix in s konfiguracijskim stikalom.



Slika 8. Ta slika prikazuje v blokovni obliki logično vezje na listu 7. To vezje sestavljajo tale podvezja in priključnice: oscilator z osnovno frekvenco 4,9152 MHz in odvodoma :2 in :4, časovnik 8253-5 za generiranje taktov (dva USARTa in prekinitev), dva USARTa z vmesniki in PIO 8255A-5 s konfiguracijskim stikalom (za nastavitev konfiguracije) in dve priključnici tipa RS-232 in priključnica tipa Centronix.

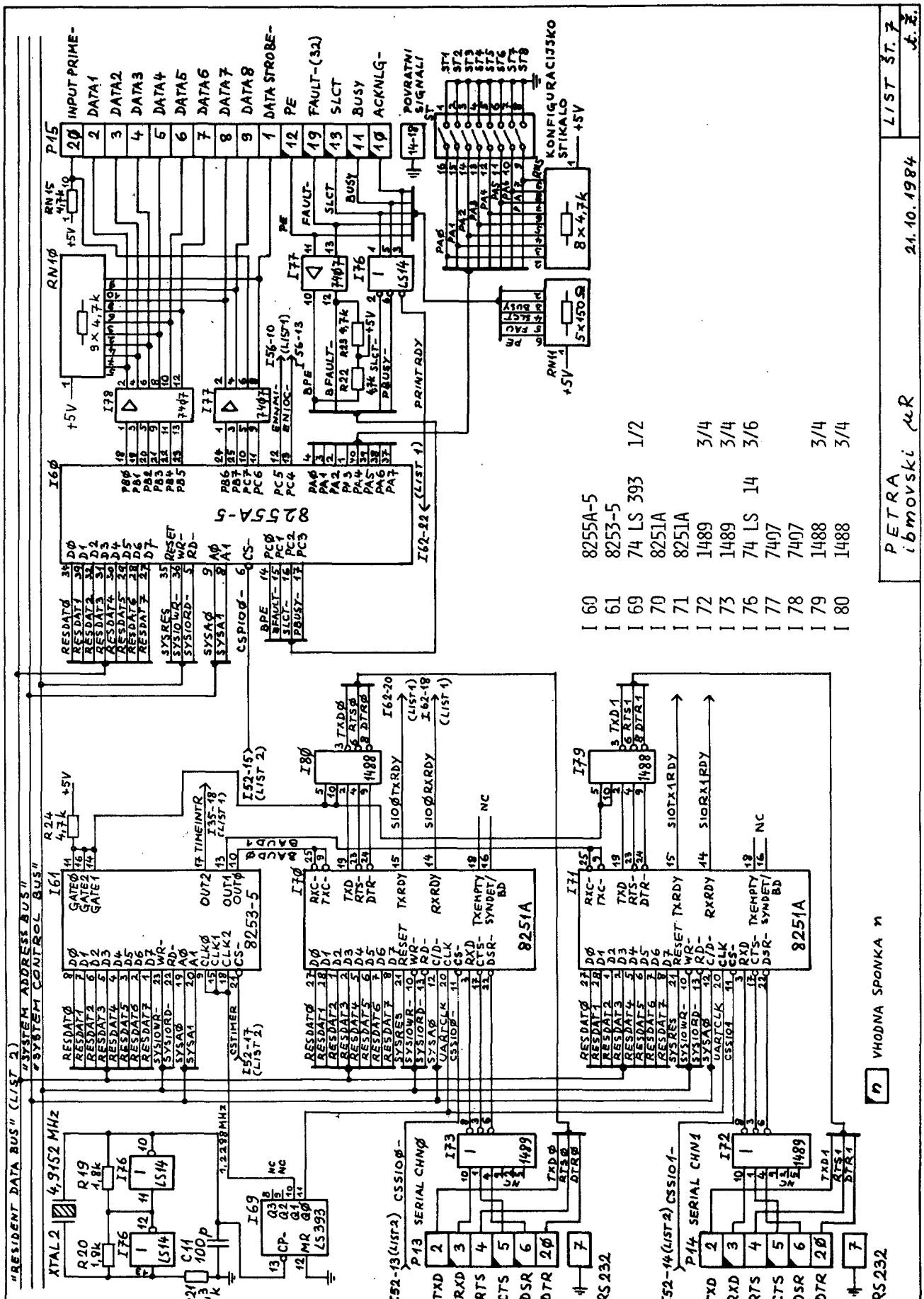
Računalnik Petra ima dva neodvisna asinhrona serijska V-1 kanala (standard RS-232C). Ta kanala omogočata med drugim uporabo prikazovalnih terminalov. Eden izmed kanalov (kanal 0) se lahko opredeli kot konzolna V-1 vrata za operacijski sistem CP M-86. Drugi kanal (kanal 1) je namenjen prosti uporabniški izbiri.

Serijska kanala uporabljata vezje USART (8251A), ki lahko istočasno oddaja in sprejema signala z različnima taktnima razmerjema. Vezje na listu 7 predvideva uporabo enakega takta sa sprejemni in oddajni kanal istega vezja USART. Pri tem se generirata dva prekinitvena signala za vsak USART, in sicer SIOxRXRDY in SIOxTXRDY (x = 0, 1). Ti štirje signali se vodijo k prekinitvenemu procesorju 8259A (162, list 1) na vhodne sponke IR0, IR1, IR2 in IR3 (nožice 18, 19, 20, 21 vezja 162, ki je pomočnik mojstra-krmilnika 135). Prekinitvi kanala 0 imata prednost pred prekinitvama kanala 1, in sprejemni kanal ima prednost pred oddajnim.

Signali za prekinitveno zahtevo so v aktivnem stanju visoki. Zahteva SIOxRXRDY javlja glavnemu procesorju, da je bil sprejet zunanji znak in da je bil pretvorjen iz serijske v paralelno obliko. Podobno se z zahtevo SIOxTXRDY javlja glavnemu procesorju, da se lahko pošlje v vezje USART znak za oddajo v okolico.

Programirljivi števnik-časovnik 8253-5 ima tri kanale s 16-bitnimi vezji. Ti števniki se lahko uporabijo na 6 različnih načinov: kot generatorji za prekinitveni signal po štetju, kot programirani generator z enim samim izhodnim impulzom, kot taktni generator, kot generator pravokotne oblike, kot s programom prožen impulzni generator in kot z zunanjimi logičnimi vezji prožen impulzni generator.

Vsi trije časovniški-števniški kanali so uporabljeni in vezje 161 se programsko inicializira ob vklopu računalnika s posebnim programom, tako da delujejo v načinu 3 (generatorji pravokotnih impulzov). Oscilatorsko vezje 176 generira signal z osnovno frekvenco 4,9152 MHz. Z delitvijo te frekvence se dobita dva signala, in sicer 1,2288 MHz za dva USARTa in 2,4576 MHz za vezje 8253-5 (161).


"RESIDENT DATA BUS" (LIST 2)
"SYSTEM ADDRESS BUS"
"SYSTEM CONTROL BUS"
P15 INPUT PRIME-
DATA1
DATA2
DATA3
DATA4
DATA5
DATA6
DATA7
DATA8
DATA STROBE-
PE
FAULT-(32)
SLCT
BUSY
ACKNLG-
POVRATNI SIGNALI
KONFIGURACIJSKO STIKALO
8255A-5
8253-5
74 LS 393 1/2
8251A
8251A
1489 3/4
1489 3/4
74 LS 14 3/6
7407
7407
1488 3/4
1488 3/4
I 60
I 61
I 69
I 70
I 71
I 72
I 73
I 76
I 77
I 78
I 79
I 80
PETRA
ibmovski uR
21.10.1984
LIST ŠT. 7
J. Ž.
VHODNA SPONKA n
4,9152 MHz
1,2288 MHz
P13 SERIAL CHN0 CSSIO0-
P14 SERIAL CHN1 CSSIO1-
TXD
RXD
RTS
CTS
DSR
DTR
RS 232

Prvi števnik časovnika 8253-5 je programirani taktni generator, ki proizvaja signal BAUD0. Podobno proizvede drugo števniško vezje taktni signal BAUD1. Pri vključitvi računalnika je taktna frekvenca obeh signalov enaka 9600 bns (bitov na sekundo ali Baudov) pomnoženo s 16 (delitev s faktorjem 16 se opravi v USARTih). Računalniški program inicializira taktno frekvenco serijskega kanala 0, ko uporabnik vtipka znak RETURN. Pri drugačni vhodni taktni frekvenci znaka RETURN se časovnik 8253-5 sprogramira 9prilagodi) na novo taktno frekvenco.

Tretji časovnik-števnik vezja 8253-5 se uporablja kot ura realnega časa ali za zakasnilne funkcije. Ta ura ime po inicializaciji periodo 10 ms (100 Hz). Ta signal se vodi na sponko IR0 mojstrskega prekinitvenega krmilnika 8259A (135, list 1) in ima najvišjo prednost v okviru maskirnega prekinitvenega sistema. Ta lastnost lahko postane zelo koristna pri aplikacijah, ki so odvisne od prekinitev in v procesih realnega časa.

Vezje 8255A-5 (160, list 7) ima dvoje V-I vrat. Ena vrata se uporabljajo med inicializacijo za branje nastavitev konfiguracijskega stikala ST (1 - 8). Osem stikal določa signale za vhodna vrata A vezja 8255A-5. Položaj stikal določa trenutno konfiguracijo sistema pri heksadecimalnem naslovu vrat 01A0H. Druga paralelna vrata so tipa Centronix za uporabo paralelnega tiskalnika. Ta druga vrata se lahko uporabijo tudi kot splošna 15-bitna vrata z 10 izhodnimi in 5 vhodnimi linijami. Vseh 15 linij je ločenih od priključnice P15 z ojačevalniki tipa 7407. Pet vhodnih linij je ločenih z ojačevalniki tipa 74 LS 14 (176, 177), vhodi pa so karakteristično zaključeni z upori 150 Ohm na napetost +5 V.

11. Sklep k drugemu delu

V prvih dveh nadaljevanjih članka smo skoraj v celoti opisali materialno zgradbo procesorja (matične plošče) ibmovskega računalnika Petra. S tem smo opravili šele lažji del naloge. Glavni poudarek v nadaljevanjih bo namenjen oživljanju sistema in uporabi operacijskega sistema; tu pa nas čaka še obilo eksperimentalnega in programirnega dela.

Slovstvo

((11)) A.P.Železnikar: Ibmovski osebni računalnik Petra I. Informatica 8 (1984), št. 4, str. 55 - 67.

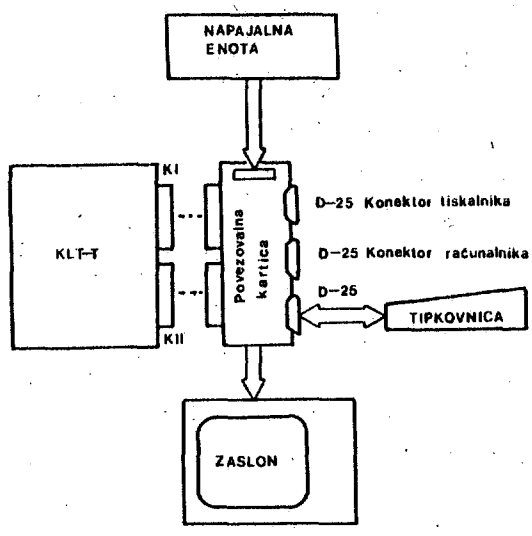
Dodatek

V tem dodatku so zbrani naslovi V_I vrat v perifernih krmilnikih (integriranih vezjih) listov 1 do 7. Ti naslovi so nam potrebni pri programiranju začetnega monitorja, s katerim bomo začeli oživljati mikroračunalnik Petra. Spočetka bo ta monitor enostaven, postopoma pa bo postajal vse bolj in bolj zapleten. Ko bomo končno ugotovili, da delujejo vsa vezja in podsistemi Petre ustrezno in zanesljivo, bomo monitor izločili in bomo začeli preizkušati posamezne segmente BIOSa, ki nam bo potreben za dani operacijski sistem (npr. za PC-DOS ali za CP_M-86).

Imamo tole razpredelnico naslovov in njihovih pomenov:

Naslovi V_I vrat I52 (74 LS 154)	
PROGRAMMABLE DMA CONTROLLER	
CSDMA-	I52-1(12) na I48-11(12) (8237A-5)
000H	channel 0: base and current address
001H	channel 0: base and current word count
002H	channel 1: base and current address
003H	channel 1: base and current word count
004H	channel 2: base and current address
005H	channel 2: base and current word count
006H	channel 3: base and current address
007H	channel 3: base and current word count
FLOPPY DISK CONTROLLER	
CSFDC-	I52-2(12) na I21-4(15) (8272A)
020H	read main status register
021H	read-write from-into data register
DMA PAGE 0_1 LATCH	
CSDMAPG0_1-	I52-3(12) na I42-10(12)(74LS173)
040H	DMA page 0 and page 1 select
DMA PAGE 2 LATCH	
CSDMAPG2-	I52-4(12) na I41-10(12) (74LS173)
060H	DMA page 2 select

ločenim mikroprocesorjem Intel 8035. Razpored tipk ustreza VT 100 standardu.



Slika 1. Zgradba terminala PAKA 3000.

2. ZGRADBA MATERIALNE OPREME

Poenostavljena zgradba materialne opreme logične kartice KLT-T je narisana na sliki 2. Sistemsko vodilo mikroprocesorja INTEL 8085 združuje vse osnovne module značilne za mikroračunalniški sistem, ki emulira zaslonski terminal. Signali sistemskega vodila so povezani na zunanji konektor KI in kartico lahko (od zunaj) razširimo s programskim pomnilnikom (EPROM) in delovnim pomnilnikom (RAM), kakor tudi z vhodno/izhodnimi napravami. CPE 8085 je zelo ugoden za našo aplikacijo zaradi primerne prekinitvene zgradbe. Razen tega ima še posebno serijsko vhodno (SID) in izhodno (SOD) linijo. Na ti dve liniji lahko preko zelo enostavnega vmesnika priključimo serijsko tipkovnico.
Prikazovalni podsistem smo zgradili na osnovi dveh najnovejših integriranih vezij z zelo visoko stopnjo integracije (AVDC in CMAC) ameriške firme Signetics. AVDC (Advanced Video Display Controller) generira osnovne časovne signale potrebne za naslavljanje video pomnilnika in znakovnega generatorja (osveževanje slike).
CMAC (Color Monchrome Attribute Controller) sestavi iz atributnih, znakovnih in sinhronizacijskih signalov ter kazalca, video-signal za krmiljenje zaslonske elektronike.

Kot komunikacijski krmilnik smo uporabili znano vezje Z80 SIO, ki omogoča tako sinhrono kot asinhrono komunikacijo preko dveh serijskih kanalov. Komunikacijski fizični vmesniki za povezavo z računalnikom so ali RS 232 C ali tokovna zanka ali posebni polduplexni vmesnik, za povezavo s tiskalnikom pa je RS 232 C vmesnik. S posebnim programabilnim časovnikom lahko programsko spreminjamo komunikacijsko uro SIO krmilnika.
Vključili smo še nepozabljajoči pomnilnik z naključnim dostopom, v katerega shranimo nekatere parametre terminala, ki jih lahko spreminjamo, vendar jih želimo ohraniti ob izklopu terminala.
Na kartici je tudi register svetlobnega peresa. Preko posebne logike za odčitavanje lahko na KLT-T priključimo svetlobo pero, ki se uporablja v nekaterih IBM-ovih terminalih in nekaterih industrijskih aplikacijah.
Paralelni vmesnik je dvosmerni in osembitni; uporabimo pa ga lahko za priključitev zunanjega grafičnega procesorja, ki je samostojna naprava, zgrajena tako, da komunicira z računalnikom posredno preko KLT-T in omenjenega paralelnega vmesnika. Video izhoda znakovnega dela in grafičnega dela terminala sta povezana v en sestavljeni video izhod, ki krmili skupni zaslon. Tako sestavljena naprava predstavlja grafični terminal.

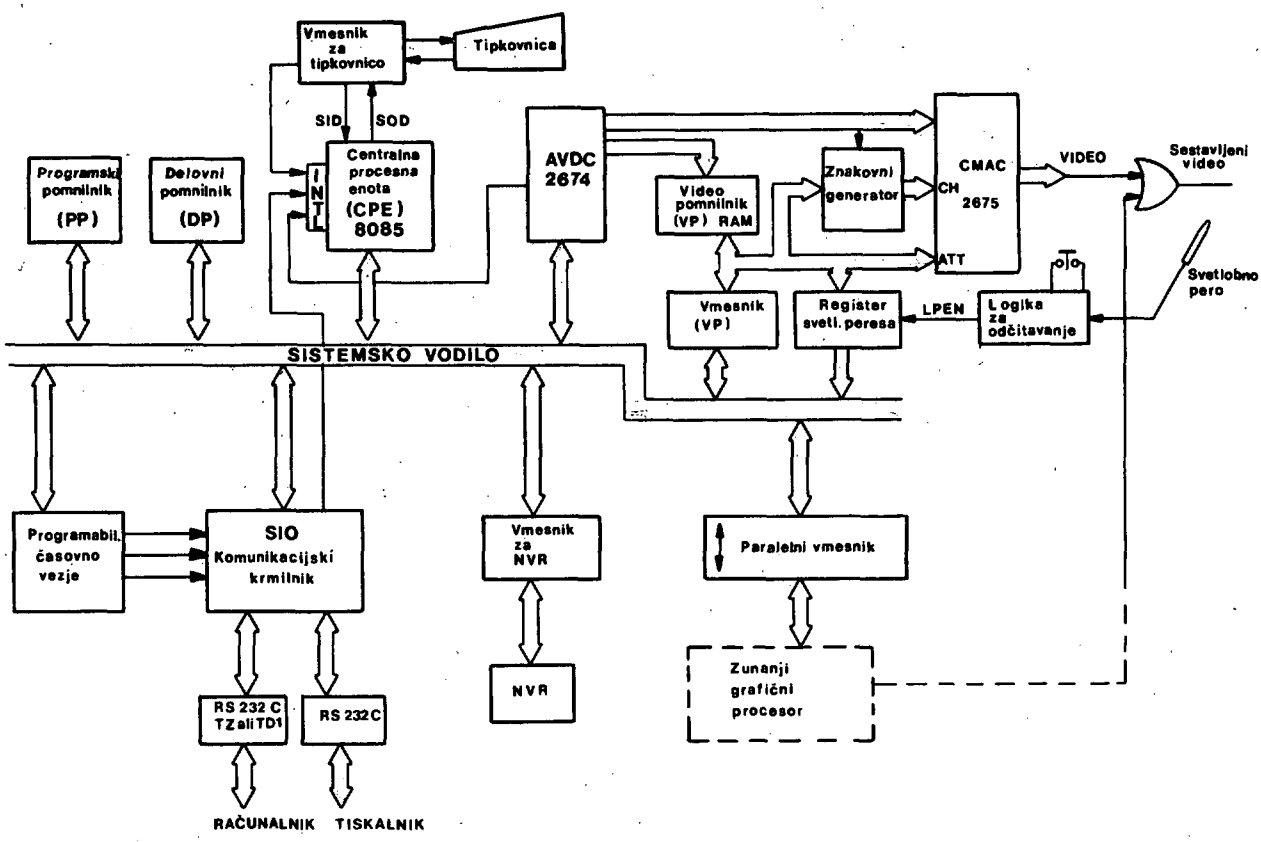
2.1. PROCESORSKI DEL KLT-T

Ta del kartice sestavljajo:
- Intelov mikroprocesor 8085
- 8 do 32 k programskega EPROM pomnilnika
- 2 do 6 k delovnega RAM pomnilnika in
- vmesnik za tipkovnico.

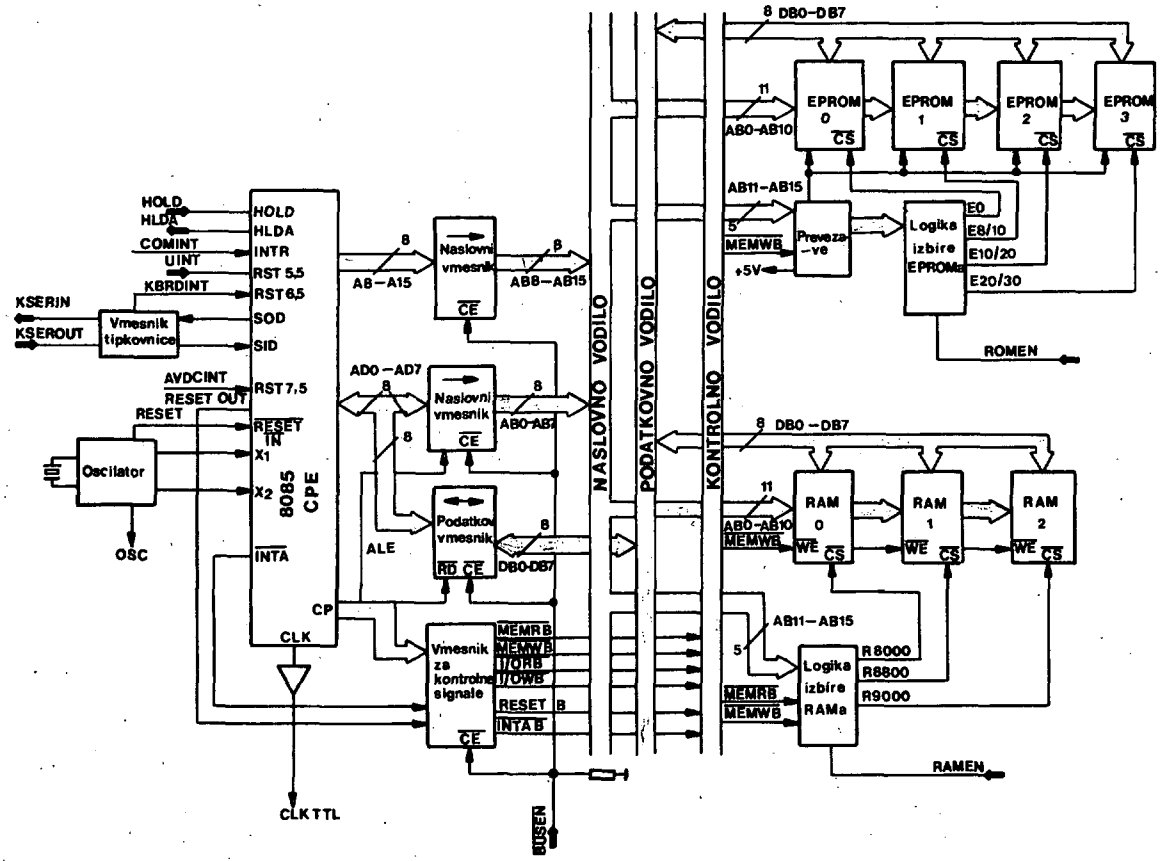
Na sliki tri je narisana bločna zgradba, na listu 1 v dodatku pa logična shema procesorskega dela KLT-T.

Procesorskega dela ne bomo natančno opisovali, saj predstavlja zelo običajno mikroračunalniško vezje.
Opozorimo naj samo na nekatere značilnosti.

Na serijski vhod in serijski izhod procesorja 8085 (IC 5) smo priključili vhodni (KSEROUT) in izhodni signal KSERIN tipkovnice. Izhodni signal vsebuje posebne sinhronizacijske impulze, ki jih izločimo s pomočjo zapaha (IC2). Tako izločeni impulzi predstavljajo prekinitveni signal (KBDINT), ki je povezan na prekinitveni vhod RST 6,5 mikroprocesorja.



Slika 2. Zgradba materialne opreme KLT-T.



Slika: 3. Zgradba procesorskega dela logične kartice

Vsak znak iz tipkovnice sestavlja osem serijsko kodiranih bitov. Tipkovnica jih pošlje v KLT-T tako, da se vsak bit začne s sinhronizacijskim impulzom. Začetek znaka določa začetni, konec znaka pa končni impulz. Programski pomnilnik smo realizirali s štirimi EPROM-i 0,1,2 in 3 (IC11, IC12, IC13 in IC14). Uporabimo lahko naslednje tipe EPROM-ov: 2716, 2732, 2764 in 27128. Prevezave P1 in P2 moramo povezati uporabljenemu EPROM-u ustrezno. S signaloma ROMEN in RAMEN lahko programsko izklopimo programski oziroma delovni pomnilnik na KLT-T in ga seveda nadomestimo z zunanjim pomnilnikom.
Sistemsko vodilo procesorja je povezano na konektor K1. Vsi signali so ojačani preko vmesnikov.

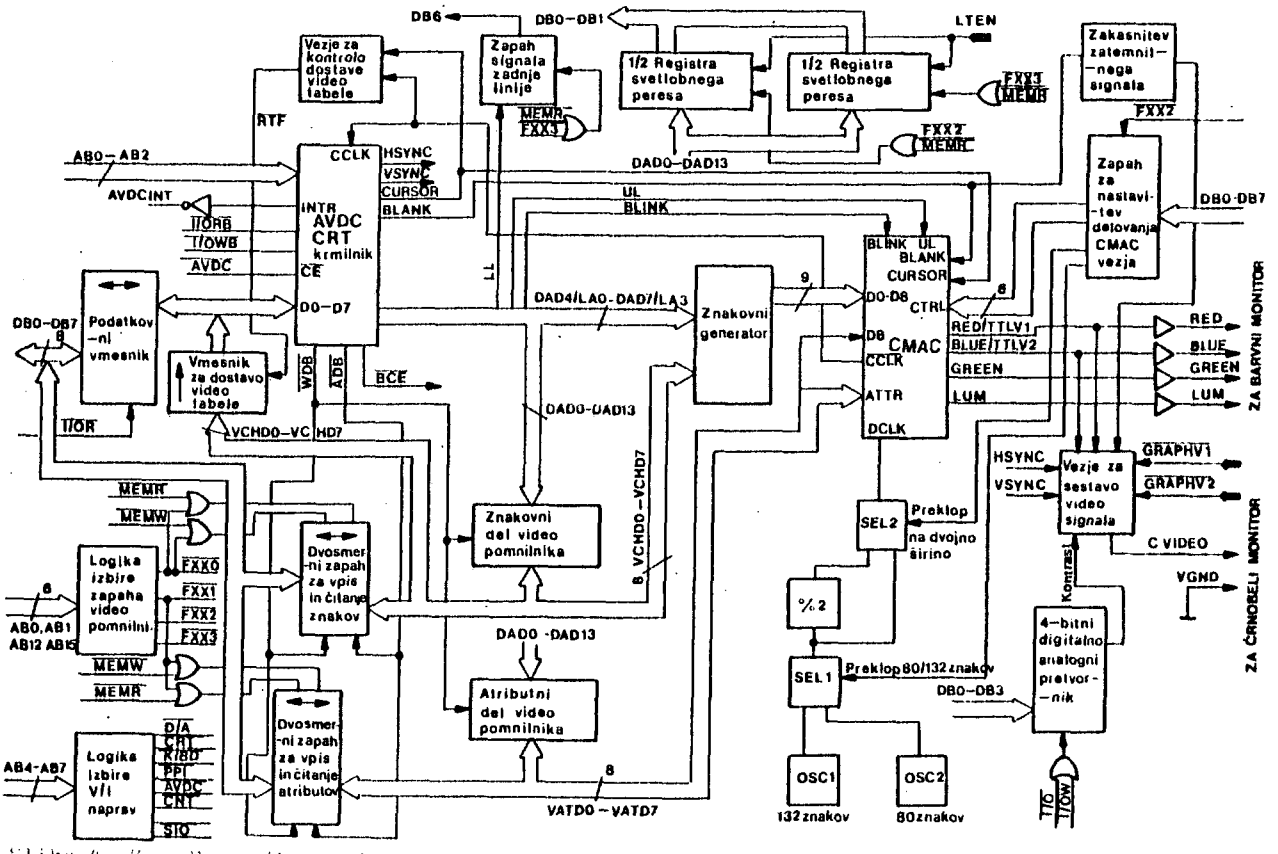
2.2. PRIKAZOVALNI DEL KLT-T

Ta del logične kartice določa prikazovalne značilnosti terminala, ki jih moramo pazljivo izbrati, saj pogojujejo (poleg komunikacijskih značilnosti) združljivost terminala z računalnikom, na katerega ga priključimo.

Realizirali smo naslednje prikazovalne značilnosti, ki ustrezajo VT 100 standardu:

- format zaslona je 24 vrstic x 80 stolpcev ali 24 vrstic x 132 stolpcev;
- prikazovalni atributi so: obratni video, utripajoč, povečana svetlost, podčrtani in kombinacija vseh naštetih;
- pomik vsebine zaslona je lahko: navzgor, navzdol, delni, skokovit (vrstica za vrstico) in gladek (linija za linijo)
- vrstice z dvojno širino ali višino znakov;
- nabori posebnih znakov (linijske grafike, semigrafike in nacionalnih abeced).
- znaki za zaslon so sestavljeni iz točk tako, da je matrika velika 7 x 9 točk, znotraj bloka velikega 9 x 12; semigrafični znaki zapolnjujejo celotni blok,
- znaki so poudarjeni.

Zadnji dve značilnosti sta predvsem pomembni za uporabnikovo udobnost in povečata čitljivost znakov.
Z naštetimi in rekoliko prirejenimi prikazovalnimi značilnostmi lahko emuliramo tudi večino drugih tipov terminalov.
Zgradba prikazovalnega dela je narisana na sliki 4, logična shema pa na listu 2 v dodatku.
AVDC (IC27), programabilni CRT krmilnik, generira vertikalne in horizontalne časovne signale za prikazovanje podatkov na zaslonu



Slika 4. Zgradba prikazovalnega dela logične kartice.

monitorja. Zaporedno naslavlja video pomnilnik in nadzoruje njegovo doseganje s strani procesorja. Programsko lahko izberemo tri različne načine delovanja, format zaslona in oblike časovnih signalov za osveževanje.

Znaki, ki se prikazujejo na zaslonu, so v ASCII kodi zapisani v znakovnem delu video pomnilnika (IC 32 in IC 33) atributi pa v atributnem delu (IC 34 in IC 35). V času osveževanja zaslona AVDC krmilnik zaporedno naslavlja video pomnilnik z naslovnimi linijami DAD0 do DAD13. Znake dostavlja po VCHD0 - VCHD7 linijah na vhod znakovnega generatorja, atribute pa po linijah VATD0 - VADD7 na vhod atributnega krmilnika CMAC (IC 47).
Procesor programsko dosega AVDC krmilnik preko kontrolnih (/ORB/,I/OWB/ in AVDC/), treh naslovnih (AB0-AB2) in podatkovnih (DB0-DB7) linij. S signalom AVDCINT zahteva krmilnik servisiranje v realnem času.
AVDC krmilnik deluje v t.i. neodvisnem načinu, kjer je videopomnilnik ločen od sistemskega. Procesor ga lahko dosega samo posredno preko AVDC. Hkrati smo izkoristili še sposobnost krmilnika, da si sam dostavlja v notranje registre začetne naslove vrstic iz video pomnilnika s pomočjo vezja za kontrolo dostave video tabele (IC 20 in IC 54) in signala RTF/ preko vmesnika za dostavo tabele (IC 22).
Procesor dosega video pomnilnik preko dvosmernega zapaha za vpis oziroma čitanje znakov in atributov (IC 28, IC 29, IC 30 in IC 31). Zapahi so postavljeni na pomnilniške naslove FXX0 in FXX1 (IC 23).
Postopek vpisa enega znaka in hkrati atributa (vpišeta se paralelno) je naslednji:
- v zapah vpišemo znak in trenutno vrednost atributa (ukaz SHLD FXX0);
- v AVDC krmilnik vpišemo ukaz: "ZAPIŠI NA NASLOV KAZALCA";
- AVDC izvrši ukaz v času, ko ne osvežuje zaslona, s pomočjo signalov WDB/ in ADB/.

Znaki oziroma vsebina videopomnilnika se v času osveževanja zaporedno dostavljajo na vhod znakovnega generatorja (IC 45), kjer se prekodirajo v obliko, enenkrat še paralelno, primerno za prikaz na zaslonu monitorja.
Izhod iz znakovnega generatorja se paralelno vpiše v serijski pomnilni register CMAC krmilnika. Na njegov vhod se zaporedno dostavljajo še signali atributov, ki se v CMAC krmilniku prištejejo izhodu serijskega pomikalnega registra in tvorijo dva TTL video izhoda (štirje nivoji video signala) TTLV1 in TTLV2. Na vhod tega krmilnika so vezani še signali BLINK (utripanje), UL (podčrtani), CURSOR (kazalec) in BLANK (zatemnitev), ki se tudi dodajo video izhodu.
Nekatere značilnosti CMAC vezja lahko programsko izbiramo s pomočjo vezja za nastavitev (IC 48):
- faktor deljenja taktnega signala DOTCLK (vhoda C0 in C1),
- črnobeli ali barvni način delovanja (vhod M/C,
- oblika kazalca v barvnem načinu (CMODE) in
- oblikovanje točk, iz katerih so sestavljeni znaki (DOTM in DOTS).

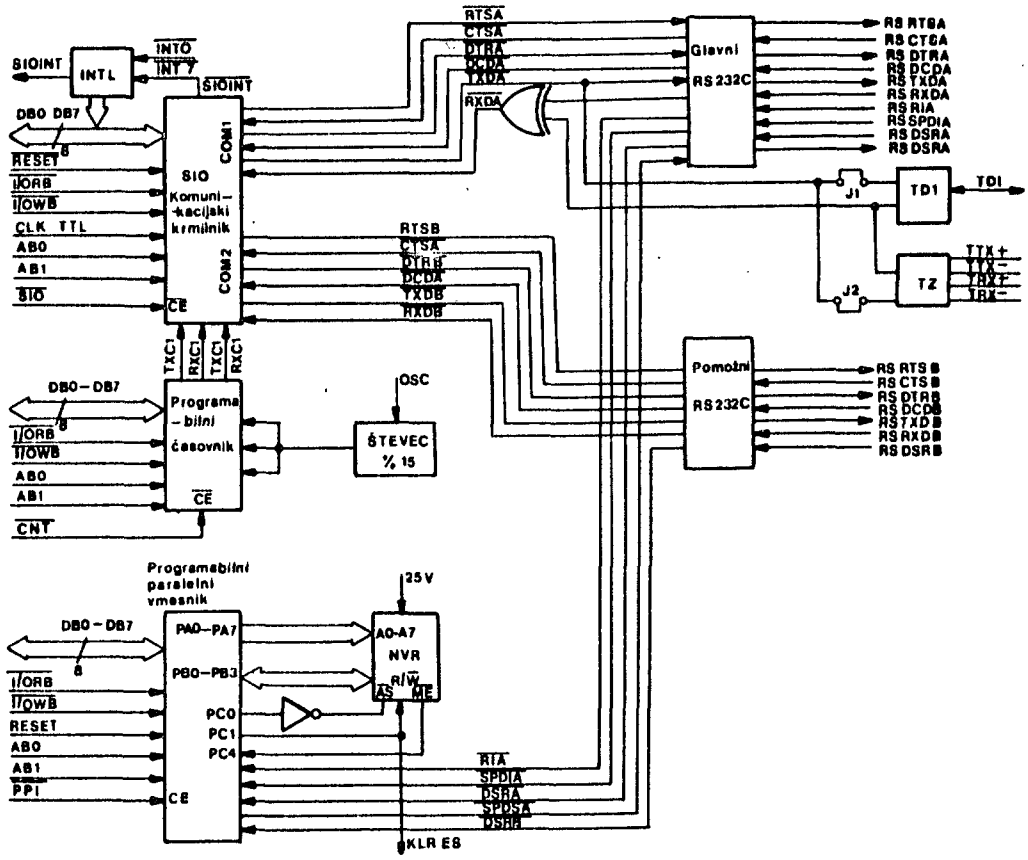
S pomočjo istega zapaha lahko tudi programsko izbiramo med izhodoma dveh oscilatorjev OSC1 in OSC2 za 80 ali 132 znakov v vrstici.
Njuna izhoda predstavljata osnovno frekvenco točke na zaslonu. Ta signal se znotraj CMAC krmilnika podeli s faktorjem, ki je določen s stanji na vhodih C0 in C1. Izhod delilnika predstavlja znakovno uro, ki je taktni signal AVDC krmilnika.
V vezja za sestavo video signala (IC 56, IC59, IC 58, IC 53, T11) tvorimo iz signalov TTLV1, TTLV2 in sinhronizacijskih impulzov HSYNC in VSYNC sestavljeni (COMPOSITE) video, ki ustreza RS 170 standardu. Dodamo lahko zunanji TTL (GRAPHV1 in 2) ali sestavljeni (CVGRPH) video signal, ki se prišteje video signalu KLT-T. Tako dobimo sestavljeno sliko iz znakovnega dela in grafičnega dela (grafični terminal).
Iz zgradbe prikazovalnega dela KLT-T (slika 4) vidimo, da imamo tudi signale RED, BLUE, GREEN in LUM za krmiljenje barvnega monitorja. Dobili smo barvni znakovni terminal.
V signalu CVIDEO za črnobeli monitor lahko s pomočjo 4-bitnega D/A pretvornika (IC 63 in IC 64) programsko spreminjamo kontrast slike.

2.3. VHODNO/IZHODNI DEL KLT-T

Vhodno/izhodni del KLT-T sestavljajo:
- serijski komunikacijski krmilnik SIO, (IC 65),
- generator komunikacijske ure (IC 70 in IC 76); osnovni oscilator je na procesorskem delu (IC 4),
- glavni in pomožni RS 232 C vmesnik (IC 73, IC 74, IC 75, IC 67 in IC 77)
- tokovna zanka (OP 1, OP 2, T 8, T 9 in T 10),
- posebni polduplekxni vmesnik, ki zagotavlja prenos po eni žici (T4, T5, T6, in T7) in

- nepozabljajoči pomilnik z vmesnikom (IC 71 in IC 72).

Zgradba tega dela KLT-T je zelo jasna, zato ga ne bomo natančneje opisali.

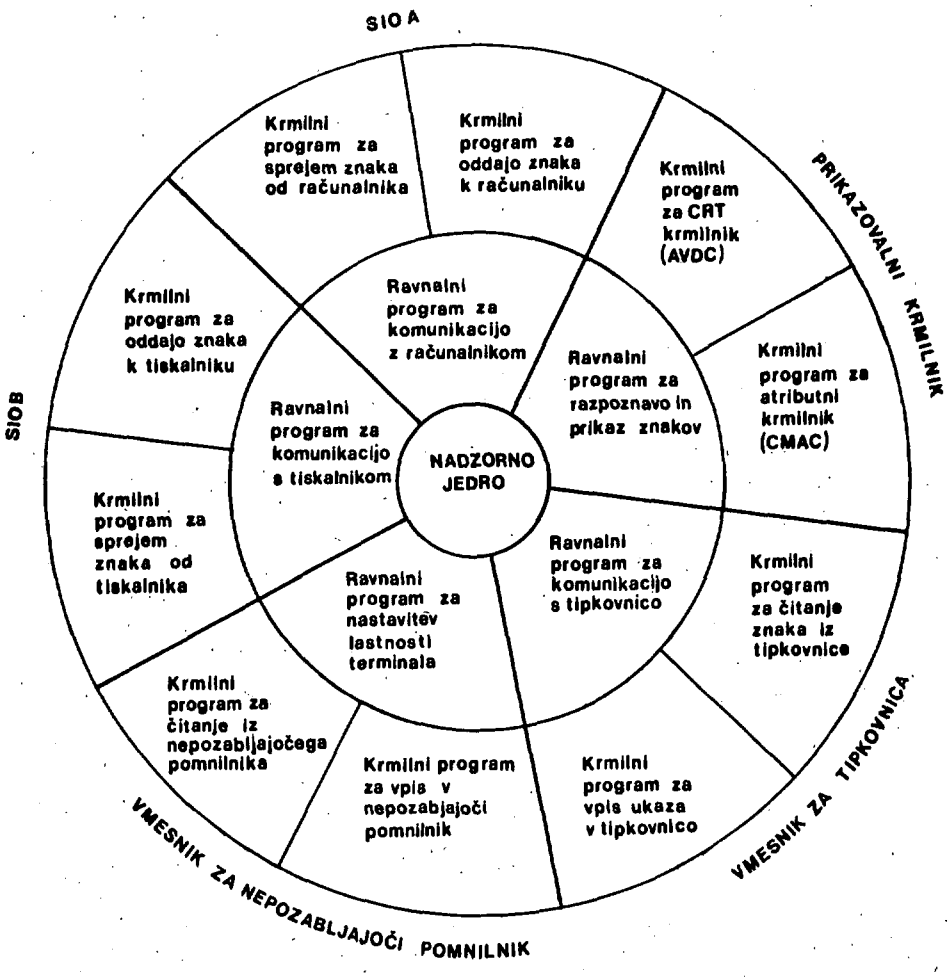


Slika 5. Zgradba vhodno/izhodnega dela KLT-T

5. ZGRADBA PROGRAMSKE OPREME

Zgradbo programske opreme bomo opisali ob primeru emulacije DEC VT 100 terminala. Shematsko jo ponazarja slika 6. Pri načrtovanju programske opreme smo predvsem težili k njeni čim dosledenjši modularnosti, saj nam takšna zgradba omogoča enostavno spreminjanje lastnosti terminala (prilagajanje tipu, ki ga emuliramo). Nadzorno jedro na osnovi vrednosti sistemskih spremenljivk dodeljuje procesorske zmogljivosti posameznim ravnalnim programom.

Ravnalni programi so medsebojno neodvisni; komunicirajo preko sistemskih spremenljivk in vmesnih pomnilnikov. Pri upravljanju s posameznimi moduli aparaturne opreme kličejo ravnalni programi odgovarjajoče krmilne programe. Nekateri krmilni programi lahko preko prekinitvene strukture nemudoma zmedejo procesorske zmogljivosti in po potrebi s spreminjanjem sistemskih spremenljivk zahtevajo ukrepanje ravnalnih programov.



Slika 6. Zgradba programske opreme

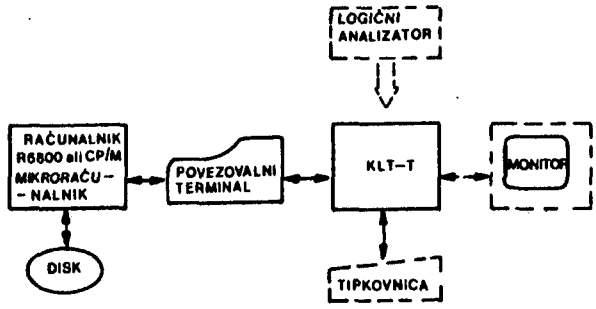
Naloge posameznih programskih modulov so naslednje:

Ravnalni program za komunikacijo s tipkovnico nadzira delovanje inteligentne tipkovnice in vtipkane znake dostavlja glede na stanje terminala (lokalno/ on-line, dupleks/polidupleks) v sprejemni vmesni pomnilnik in/ali v oddajni vmesni pomnilnik. Nadzorno jedro v prvem primeru aktivira ravnalni program za razpoznavo in prikaz znakov, v drugem primeru pa za komunikadijo z računalnikom. Ravnalni program za razpoznavo in prikaz zapiše znak v video pomnilnik, vpliva na oblikovanje zaslona ali spremeni sistemske spremenljivke tako, da nadzorno jedro aktivira ravnalni program za komunikacijo s tiskalnikom ali za nastavitev lastnosti terminala. Ravnalni program za komunikacijo z računalnikom oddaja znake iz oddajnega vmesnega pomnilnika k računalniku. Kadar sprejme znak od računalnika, ga zapiše v sprejemni vmesni pomnilnik. Nadaljno obdelavo pa, kot v primeru vtipkanega znaka, prevzame ravnalni program za razpoznavo in prikaz znakov. Ravnalni program za komunikacijo s tiskalnikom nadzira prenos znakov iz sprejemnega vmesnega pomnilnika ali iz video pomnilnika k tiskalniku. Ravnalni program za nastavitev lastnosti terminala omogoča izbiranje in spreminjanje komunikacijskih in prikazovalnih lastnosti terminala ter lastnosti tipkovnice. Spreminja tudi sistemske spremenljivke in s tem vpliva na delovanje ostalih ravnalnih programov.

V primeru emulacije drugih tipov terminalov ostanejo krmilni programi nespremenjeni. Spremeniti moramo ravnalni program za razpoznavo in prikaz znakov, redkeje še za komunikacijo z računalnikom ali s tiskalnikom, zelo redko pa tudi za komunikacijo s tipkovnico ali za nastavitev lastnosti terminala.

4. Opis razvojnega orodja

Običajno orodje za razvoj mikroračunalniškega sistema (kot je n.pr. KLT-T) je razvojni sistem z emulatorjem za mikroprocesor, na katerem je zasnovan. V našem primeru smo uporabili svojski sistem, katerega zgradba je narisana na sliki 6.



Slika 6. Zgradba sistema za razvoj programske in aparaturne opreme KLT-T.

V KLT-T smo vgradili monitorski program, ki opravlja funkcije potrebne za testiranje programske in tudi aparaturne opreme mikroračunalniškega sistema (KLT-T): nastavitev uporabnikovih registrov (mikroprocesorja 8085), pregledovanje in spreminjanje pomnilniških lokacij, poganjanje programov do prekinitvene točke in koračno, obratni zbirnik 8080, ... Logični analizator uporabimo za testiranje aparaturne opreme in realnočasovnih delov programske opreme. Vhodno izhodna enota monitorskega programa je povezovalni terminal, ki je tudi terminal računalnika R6800 ali CP/M mikroračunalnika. Na tem računalniku z diskovnim operacijskim sistemom kreiramo izvorne preoograme in jih prevajamo v objektno kodo, ki jo s pomočjo povezovalnega terminala prenašamo v KLT-T. Posebni prečni nalagalnik monitorskega programa KLT-T naloži objektni program na v naprej določeni del pomnilnika. Naložen program lahko sedaj testiramo.

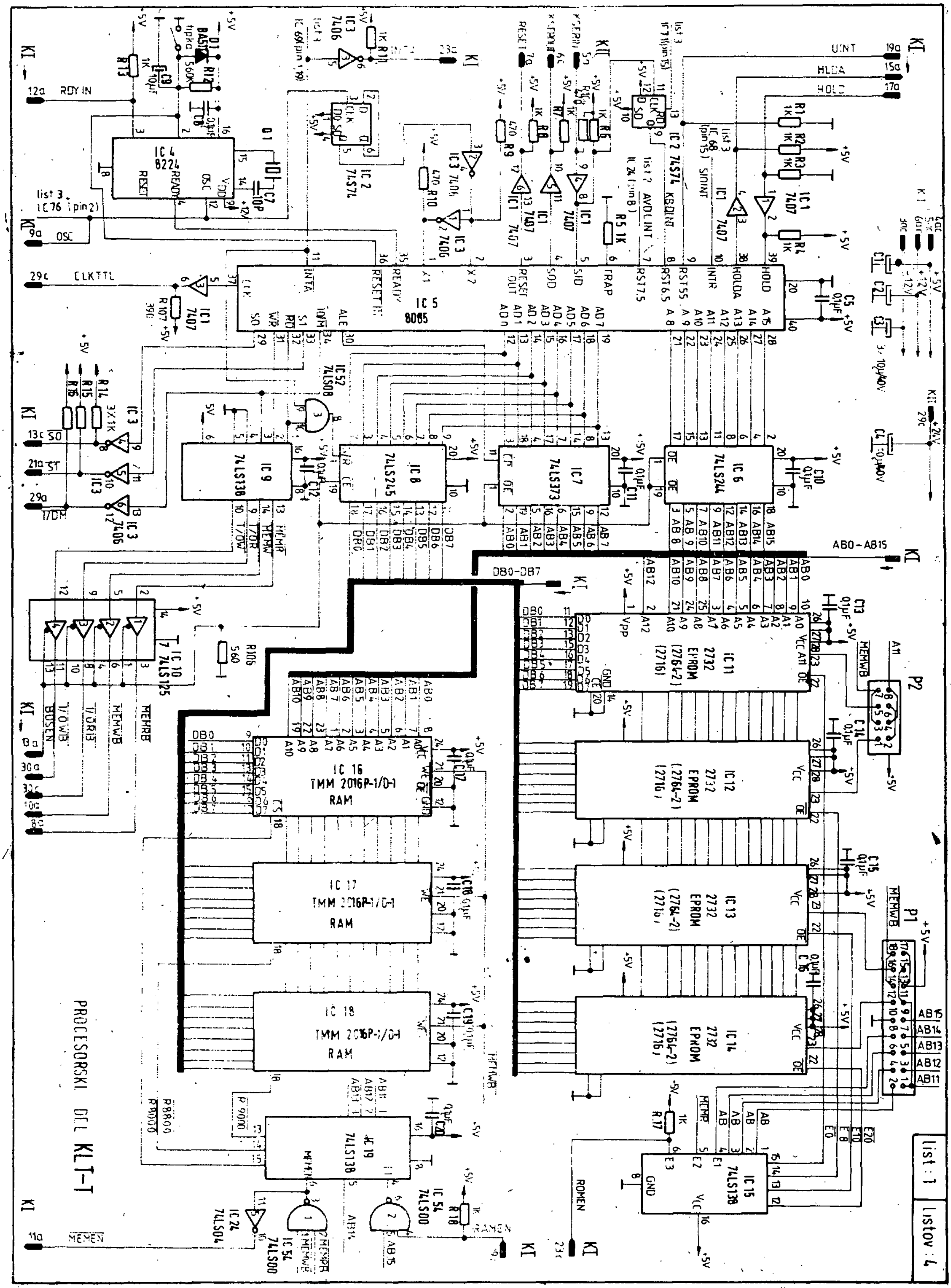
Računalnik R6800 je večuporabniški in lahko nanj preko večjega števila povezovalnih terminalov priključimo več KLT-T. Uporabljati pa moramo prečno programsko opremo (prečni zbirnik 8080).
CP/M mikroračunalnik je enouporabniški vendar na njem ni potrebno uporabljati prečne programske opreme. V primeru, da ima mikroračunalnik vključen prikazovalni krmilnik, ne potrebujemo povezovalnega terminala, pač pa se na KLT-T priključim neposredno preko pomožnega RS 232 C vmesnika.

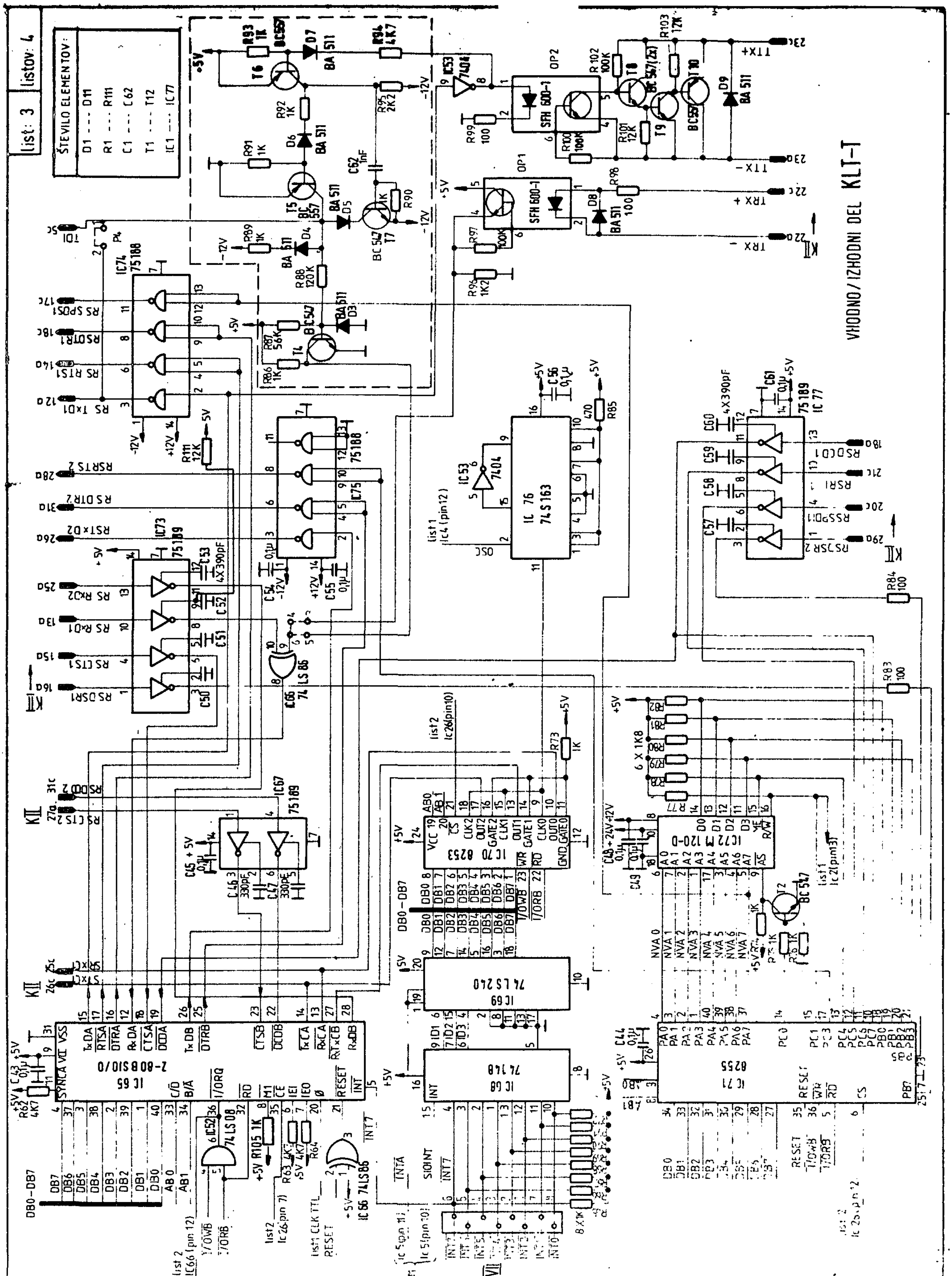
ZAKLJUČEK

V začetku novembra smo v Gorenju izdelali začetno serijo petdeset kosov KLT-T. Rezultati testiranj terminalov PAKA 3000 so zelo ugodni.

LITERATURA

1. K. Steblovnik, Povezovalni terminal - del sistema za razvijanje aparaturne in programske opreme, Mipro 82;
2. K. Steblovnik, Emulator za razvijanje in testiranje mikroračunalniških sistemov, Mipro 83;
3. K. Steblovnik, A, Križnik, Opis programske opreme terminala s pomočjo teorije o vzorčno vodenih sistemih, Mipro 84;
4. K. Steblovnik, D, Vrhovec, Sistem za testiranje mikroračunalniških modulov, Mipro 84;
5. INTEL, Component Data Catalog 1983;
6. INTEL, The MCS - 80/85 Family User's Manual
7. PHILIPS/SIGNETICS, Microprocesora, micro computers and peripheral circuitry;
8. SGS, Z80 SIO technical manual.


PROCESORSKI DEL KLT-1
list: 1
listov: 4


list: 3
listov: 4
ŠTEVILO ELEMENTOV:
D1 --- D11
R1 --- R111
C1 --- C62
T1 --- T12
IC1 --- IC77
VHODNO/IZHODNI DEL KLT-T

PRIKAZOVALNI DEL KLT-T

OPTIMIZACIJA NIVOA INDEKSA U INDEKSNIM DATOTEKAMA

SINIŠA J. DJORDJEVIĆ

UDK: 681.3.016

Nivoi indeksiranja posmatraju se u zavisnosti od srednjeg broja pristupa, vremena traženja i zauzeća memorije. Za svaki od ovih kriterijuma odredjuje se najpovoljniji broj nivoa indeksa čime se stvaraju uslovi za optimizaciju nivoa indeksa u indeksnim datotekama u celini.

INDEX LEVELS OPTIMIZATION IN THE INDEXED FILES: This paper presents optimal number of index levels for each criterion, main number of probes, time searching and memory occuping.

1. Uvod

Broj nivoa indeksa posmatra se sa tri aspekta značajna za efikasnost indeksnih datoteka. To su srednji broj pristupa, vreme traženja i zauzeće memorije. Sve tri navedene veličine zavise od broja nivoa u indeksnim datotekama i na odgovarajući način utiču na efikasnost.

Dobijeni rezultati važe ako je primenjeno sukcesivno traženje i po indeksima i u datoteci. Ovo nije veliko ograničenje jer se sa stanovišta praktične organizacije traženja u indeksnim datotekama sukcesivno traženje najčešće primenjuje.

2. Minimizacija aproksimiranog izraza za srednji broj pristupa

U sekvencijalnoj datoteci sa Q slogova gde svaki slog ima podjednaku verovatnoću traženja srednji broj pristupa za sukcesivno traženje dat je izrazom: $Z = (Q+1)/2$. Uz pretpostavku $Q >> 1$, ovi su uslovi u praksi vrlo česti, izraz za srednji broj pristupa postaje: $Z = Q/2$.

Na osnovu poslednje relacije, izraz za srednji broj pristupa u indeksnoj datoteci sa jednim nivoom indeksa ima oblik:

$$Z = N_1/2 + Q/2N_1$$

gde je N_1 broj upisa u prvi indeks.

Prema /2/ , diferenciranjem ovog izraza po N_1 i izjednačavanjem sa nulom dobijenog rezultata dobija se da je srednji broj pristupa minimalan kada je: $N_1 = \sqrt{Q}$.

Takodje prema /2/, za dva nivoa indeksa:

$$Z = N_1/2 + N_2/2 + Q/2N_1N_2$$

gde je N_2 broj upisa u drugom nivou indeksa.

Istim postupkom dobija se da je srednji broj pristupa minimalan ako je $N_1 = N_2 = \sqrt[3]{Q}$.

Za n nivoa indeksa ima se:

Stav: Srednji broj pristupa za n nivoa indeksa je minimalan ako je ispunjen uslov:

$$N_1 = \dots = N_i = \dots = N_n = \sqrt[n+1]{Q}$$

Dokaz:

$$Z = N_1/2 + \dots + N_i/2 + \dots + N_n/2 + Q/2N_1 \dots N_i \dots N_n$$

$$\partial Z/\partial N_1 = 1/2 - Q/2N_1^2 \dots N_i \dots N_n$$

$$\partial Z/\partial N_i = 1/2 - Q/2N_1 \dots N_i^2 \dots N_n$$

$$\partial Z/\partial N_n = 1/2 - Q/2N_1 \dots N_i \dots N_n^2$$

Na osnovu $\partial Z/\partial N_1 = \dots = \partial Z/\partial N_i = \dots = \partial Z/\partial N_n = 0$ dobija se:

$$\begin{aligned} Q &= N_1^2 \dots N_i \dots N_n \\ &\vdots \\ Q &= N_1 \dots N_i^2 \dots N_n \qquad \dots (1) \\ &\vdots \\ Q &= N_1 \dots N_i \dots N_n^2 \end{aligned}$$

Deljenjem i-te jednačine sistema (1) lako se dobija:

$$N_1 = \dots = N_i = \dots = N_n = \sqrt[n+1]{Q}$$

Dobijeni rezultat znači da u indeksnoj datoteci sa n nivoa indeksa za minimalni srednji broj pristupa broj upisa po svakom nivou treba da bude jednak i da iznosi $\sqrt[n+1]{Q}$.

Sada izraz za srednji broj pristupa ima oblik:

$$Z = \frac{1}{2}\left(\sqrt[n+1]{Q} + \dots + \sqrt[n+1]{Q}\right) + \frac{Q}{2\sqrt[n+1]{Q}\dots\sqrt[n+1]{Q}}$$

što daje:

$$Z = \frac{1}{2}(n+1)Q^{\frac{1}{n+1}}$$

$$\frac{dZ}{dn} = \frac{1}{2}Q^{1/(n+1)}\left(1 - \frac{1}{n+1}\ln(Q)\right)$$

na iz $dZ/dn = 0$ sledi: $n = \ln(Q) - 1$

Za broj upisa po svakom nivou indeksa dobija se $N = e$ a na osnovu:

$$N = Q^{1/\ln(Q)}$$

Sa stanovišta minimalnog srednjeg broja pristupa broj nivoa indeksa je $\ln(Q) - 1$ a broj upisa po indeksu je e .

Dobijeni rezultati moraju se shvatiti uslovno. Srednji broj pristupa u sekvencijalnoj datoteci sa Q slogova opisuje se izrazom Z=(Q+1)/2 a aproksimacija Z = Q/2 može se usvojiti za Q>>1 . Ako se posle takve aproksimacije dobije da je Q = e cela analiza se dovodi u pitanje.

3. Minimizacija bez aproksimacije

Izraz za srednji broj pristupa sada ima oblik:

$$Z = \frac{N_1 + 1}{2} + \ldots + \frac{N_n + 1}{2} + \frac{1}{2}\left(\frac{Q}{N_1 \ldots N_n} + 1\right)$$

Diferenciranjem se opet dobija sistem jednačina (1) čime zaključak:

$N_1 = \ldots = N_n = \sqrt[n+1]{Q}$ i dalje važi.

Time izraz za srednji broj pristupa postaje:

$$Z = \frac{1}{2}\left(n + 1 + (n+1)Q^{\frac{1}{n+1}}\right)$$

Stav: Kada n raste Z opada.
Da bi se stav dokazao potrebno je dokazati da iz $n_2 < n_1$ sledi $Z(n_2) > Z(n_1)$. Dokaz:

$$Q^{\frac{1}{n_1+1}}(n_1+1) - Q^{\frac{1}{n_2+1}}(n_2+1) < n_2 - n_1 \;;$$

$n_2 - n_1 < 0$ pa se dobija:

$$\frac{Q^{\frac{1}{n_1+1}}}{Q^{\frac{1}{n_2+1}}} < \frac{n_2+1}{n_1+1} \;; \quad \frac{n_2+1}{n_1+1} < 1 \;;$$

$Q > 1$ što dalje daje:

$\frac{1}{n_1+1} < \frac{1}{n_2+1}$ iz čega neposredno sledi

$n_2 < n_1$

Vrednost Z je manja kada n raste.
S druge strane, može se postaviti pitanje: Koji je maksimalan broj nivoa indeksa koji se može ostvariti, odnosno koji je minimalan broj upisa po indeksu?
Kao što je već pokazano, veza izmedju broja nivoa indeksa i broja upisa po indeksu ima oblik:

$$N = \sqrt[n+1]{Q} \qquad \ldots (2)$$

gde je N broj upisa po svakom od n nivoa indeksa.
Pošto se radi o celobrojnim vrednostima relacija (2) se mora modifikovati:

$N = \lceil \sqrt[n+1]{Q} \rceil + 1$, $\sqrt[n+1]{Q}$ -nije ceo

$N = \sqrt[n+1]{Q} + 1$, $\sqrt[n+1]{Q}$ -ceo broj

Relacijom $\lceil a \rceil$ dobija se prirodni broj m takav da važi $m \leq a$ uz uslov da ne postoji prirodni broj m_1 koji ispunjava uslov $m < m_1 \leq a$.
Ako se uzme da $n \to \infty$ ima se:

$$\lim_{n\to\infty} N = \lim_{n\to\infty} \left(\lceil \sqrt[n+1]{Q} \rceil + 1\right) = 1+1 = 2$$

$\sqrt[n+1]{Q}$ za $n > \log_2 Q$ nije ceo broj
sa N=2 dobija se: $n = \log_2 Q - 1$

To znači da je minimalan broj upisa po indeksu 2 i da nema smisla n povećavati više od $\log_2 Q - 1$ čime su dobijene vrednosti za broj nivoa indeksa i broj upisa po indeksu takve da srednji broj pristupa bude minimalan.

Primetimo da nema smisla uzimati da broj upisa po indeksu bude 1 (čime je moguće proizvoljno povećavati broj nivoa indeksa) jer se onda radi o direktnim datotekama ili invertnim listama a to izlazi iz okvira problematike koja se ovde obradjuje. To bi trebalo da bude predmet posebne analize.
Dobijene rezultate ne bi trebalo dovoditi u vezu sa binarnim traženjem u sekvencijalnim datotekama, jer binarno traženje u sekvencijalnim datotekama, po mehanizmu, pripada drugoj problematici.

4. Vreme traženja

Neka je $t_{1,i}$ vreme prelaska sa jednog nivoa indeksa na drugi $i \in (1,\ldots,n+1)$. Neka je $t_{2,i,j}$ vreme prelaska sa jednog upisa na drugi u istom nivou indeksa $j \in (2,\ldots,N)$. $t_{3,k}$ - je vreme prelaska sa jednog sloga na drugi u datoteci, $k \in (2,\ldots,Q/N^n)$. Srednje vreme traženja dato je izrazom:

$$t_s = \sum_{i=1}^{n+1} t_{1,i} + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=2}^{N} t_{2,i,j} + \frac{1}{2}\sum_{k=2}^{Q/N^n} t_{3,k} \qquad \ldots (3)$$

Kako se može uzeti da je $t_{2,i1,j} = t_{2,i2,j}$ to relacija (3) postaje:

$$t_s = \sum_{i=1}^{n+1} t_{1,i} + \frac{1}{2} n \sum_{j=2}^{N} t_{2,j} + \frac{1}{2}\sum_{k=2}^{Q/N^n} t_{3,k}$$

S druge strane, kod traženja podataka u svakom sistemu možemo definisati dominantno vreme. Ovo vreme, u oznaci t_d , predstavlja vreme potrebno da se pređe sa jedne hardverske celine na drugu a da se pri tome izvrši i mehaničko kretanje glava za čitanje i pisanje.
Ima se:

$t_{1,r_1} = t_d \quad t_{1,r_4} = t_o \quad r_1 \neq r_4 \quad r_1, r_4 \in (1,\ldots,n+1)$

$t_{2,r_2} = t_d \quad t_{2,r_5} = t_o \quad r_2 \neq r_5 \quad r_2, r_5 \in (2,\ldots,N)$

$t_{3,r_3} = t_d \quad t_{3,r_6} = t_o \quad r_3 \neq r_6 \quad r_3, r_6 \in (2,\ldots,Q/N^n)$

t_o - vreme prelaska sa jedne fizičke rečenice (sloga) na drugu (uzima se da ovo vreme ne zavisi od dužine fizičke rečenice)
r_1 - indeksi za ona vremena t_1 koja su jednaka t_d .

Funkcijom $r(r_1)$ dobija se broj vremena t_1 koja imaju indeks r .
Važi: $r(r_1) + r(r_4) = n$.

Kako je u praksi najčešće:

$t_{2,r_5} = \text{const.}$; $t_{3,r_6} = \text{const.}$

$t_{2,r_5} = t_{3,r_6}$; $t_d >> t_o$

relacija (3) postaje:

$$t_s = r(r_1)t_d + \frac{1}{2}nr(r_2)t_d + r(r_3)t_d$$

Drugi član zbira jednak je nuli, $r(r_2)=0$, jer se indeksi uvek mogu organizovati tako da se svi upisi po jednom nivou indeksa nalaze na istoj hardverskoj celini.

Prema tome:

$$t_s = r(r_1)t_d + r(r_3)t_d$$

što znači da srednje vreme traženja ne zavisi od n.

Ovo je jasno i zbog toga što $r(r_1)$ zavisi od toga kako smo projektovali datoteku. Ako bi smo svaki nivo indeksa smeštali na posebnu hardversku celinu onda bi srednje vreme traženja bilo obrnuto proporcionalno sa n što se u praksi, normalno, ne primenjuje.

Relacija (3) se može primeniti na najopštije analize.

Srednje vreme traženja zavisi od n tako što ima optimalnu vrednost onda kada i Z ima optimalnu (minimalnu) vrednost s tim što je, kao što se iz analize može videti, važno optimalno organizovati datoteku i indekse u odnosu na dominantno vreme. Ovakva organizacija ne zavisi od broja nivoa indeksa već od organizacije datoteke.

$$t_{1,i} = t_{2,j} = t_{3,k} = t'_o$$

to je na osnovu relacije (3):

$$t_s = (n+1)t'_o + \frac{1}{2}n(N-1)t'_o + \frac{1}{2}(\frac{Q}{N^n} - 1)t'_o$$

$$t_s = (n+1)t'_o + \frac{1}{2}n(Q^{\frac{1}{n+1}} - 1)t'_o + \frac{1}{2}(Q^{\frac{1}{n+1}}-1)t'_o$$

$$t_s = t'_o\frac{1}{2}(n + 1 + (n+1)Q^{\frac{1}{n+1}}) = t'_oZ$$

t'_o = const.

čime je pokazano da t_s zavisi od n na isti način kao i Z.

Ovo dalje znači da za izračunato n (u delu koji se odnosi na srednji broj pristupa) i t_s ima minimalnu vrednost u direktnim medijumima.

5. Zauzeće memorije

Neka je f dužina sloga bez ključa u bajtovima i K_f dužina ključa na osnovu kojeg se vrši indeksiranje. Neka je S dužina adrese ćelije memorije u bajtovima (u svaku ćeliju se smešta po jedan slog datoteke). Zauzeće memorijskog prostora za n nivoa indeksa (područje u koje su smešteni slogovi datoteke sa ključevima ne uzima se u obzir) dato je izrazom:

$$N_i = \sqrt[n+1]{Q}$$

$$S_n = \sum_{i=1}^{n} N_i^2(K_f + S) \quad , \quad K_f + S = A$$

S_n - ukupno zauzeće memorije po indeksima

$$S_n = A\sum_{i=1}^{n} (Q^{\frac{1}{n+1}})^i$$

$$S_n = A(Q - Q^{\frac{1}{n+1}}) \;/\; (Q^{\frac{1}{n+1}} - 1)$$

$$S'_n = \frac{A\ln(Q)}{(n+1)^2} Q^{\frac{1}{n+1}} \frac{Q-1}{(Q^{\frac{1}{n+1}} - 1)^2} > 0$$

Sa porastom n raste i funkcija S_n. Funkcija S_n nema ekstremum.

Za $N=2$ odnosno $n=\log_2 Q - 1$ za S_n se dobija: $S_n = A(Q - 2)$ Za $Q \gg 1$ dobija se: $S_n \approx AQ$

Neka je: $A = (f + K_f)/R$. Za S_n se onda dobija:

$$S_n = \frac{1}{R}(f + K_f)Q = \frac{1}{R}S_d$$

gde je S_d memorijski prostor za datoteku bez indeksa. U praksi je R najčešće oko 5 što znači da je za indekse potreban memorijski prostor u iznosu 20% od prostora za samu datoteku.

R i veličina datoteke determinišu nivo indeksa sa stanovišta memorijskog prostora. I ako danas memorijski prostor nije problem, prema uštedi memorijskog prostora trebalo bi smanjivati nivo indeksa.

Uticaj hardverskih celina memorije na zauzeće memorije nije razmatran jer na njega mnogo više utiče sama organizacija datoteke odnosno rasporedjivanje indeksa i slogova datoteke po hardverskim celinama. Ovo rasporedjivanje može imati i bitniji uticaj na zauzeće memorije od uticaja broja nivoa indeksa.

6. Zaključak

Sa aspekta organizovanja područja prekoračenja može izgledati da je indeksiranje sa brojem nivoa indeksa i brojem upisa po indeksu koji ne zavise od hardverskih celina neprihvatljivo. Ovaj zaključak nije tačan a odgovarajući dokaz je izvan okvira ovog teksta. Prelaz na područje prekoračenja a i samo područje prekoračenja mogu se takodje organizovati na više načina. Izmedju ostalog moguće je i područje prekoračenja indeksirati prema dolazu slogova s tim da u datoteci ne bi bilo nikakvog pomeranja slogova a time ni ažuriranja indeksa.

U osnovnom kontekstu indeksnih datoteka iznešena analiza ima opravdanja kod datoteka sa niskom frekvencijom dodavanja slogova.

Iz analize se moglo videti da je za optimizaciju nivoa indeksa potrebno slediti kriterijume koji daju dve protivurečne vrednosti. Prema srednjem broju pristupa i prema srednjem vremenu traženja potrebno je da je broj nivoa indeksa što veći odnosno da granično bude $n = \log_2 Q - 1$ dok je prema uštedi memorijskog prostora potrebno da taj broj bude što manji (odnosno da je nula kada nema dodatnog zauzeća memorije). I ako memorija, danas, ne predstavlja problem moguće je na osnovu iznešenih formula formirati individualne kriterijume i težinske funkcije na osnovu kojih je dalje moguće problem optimizacije nivoa indeksa rešiti na osnovu sopstvenih uslova.

Indeksiranje treba shvatiti šire, i izvan konteksta indeks sekvencijalnih datoteka. Indeksiranje predstavlja oblik formiranja veza izmedju podataka te se tako primenjuje u gotovo svakoj organizaciji podataka. Rezultati definisani u ovom tekstu u tom smislu mogu da unaprede organizaciju indeksiranja.

Ovo potvrdjuju i organizacije podataka preko sekundarnih ključeva za koje, po veličini i u orgnizacionom smislu, nisu neophodne banke podataka (ili nisu ni dostupne zbog softverske podrške). Ovakve organizacije podataka predpostavljaju upravo dodatna, različita, indeksiranja.

7. Literatura

1./ J. Martin, COMPUTER DATA - BASE ORGANIZATION, second edition, PRENTICE - HALL, ENGLEWOOD CLIFFS, NEW JERSEY 07632.

2./ H. Wedekind, ORGANIZACIJA PODATAKA, ZAK.

INFORMATICA 1/85

BARVANJA TOČK GRAFOV

VLADIMIR BATAGELJ

UDK: 519.174

UNIVERZA EDVARDA KARDELJA V LJUBLJANI
FAKULTETA ZA NARAVOSLOVJE IN TEHNOLOGIJO
VTO MATEMATIKA IN MEHANIKA

V sestavku je podan pregled najpomembnejših rezultatov o teoretičnih in algoritmičnih vidikih problema barvanja grafov. Na koncu je podan postopek, ki združuje trenutno najuspešnejša hevristična postopka za barvanje grafov: Szekeres-Wilfov in Brélazov postopek.

GRAPH COLORING

In the paper a survey of the most important results on theoretic and algorithmic aspects of the graph coloring problem is given. A heuristic procedure which combines the Szekeres-Wilf and Brélaz coloring procedures is presented at the end.

Math.Subj.Class.(1980): 05C15

NEKAJ PRIMEROV PREVEDBE PROBLEMOV NA NALOGO O BARVANJU TOČK GRAFA

Nalogo o barvanju točk grafa zastavimo takole:

Dan je graf $G = (V,E)$. Pobarvaj točke grafa G tako, da ne bo noben par sosednjih točk pobarvan z isto barvo.

Običajno zahtevamo še:

Pri tem uporabi čim manjše število različnih barv.

Naloga o barvanju točk grafa sodi v seznam "osnovnih" (kanonskih) kombinatoričnih nalog, saj lahko nanjo prevedemo celo vrsto, navidez neprimerljivih, nalog. Poglejmo si nekaj primerov:

1. Radijski oddajniki

Radijska oddajnika lahko motita eden drugega, če sta si preblizu. Kako razdeliti valovne dolžine dani množici oddajnikov, tako da se ne bodo medsebojno motili? Koliko najmanj valovnih dolžin je za to potrebnih?

Prevod:

TOČKE GRAFA: oddajniki
POVEZAVE: oddajnika sta sosednja (povezana), če sta si preblizu (lahko motita eden drugega)
BARVE: valovne dolžine

2. Turistični vodiči

Turistična agencija namerava organizirati n izletov. Za vsak izlet i poznamo datum njegovega začetka Z_i in datum njegovega konca K_i . Koliko (najmanj) vodičev je potrebnih za izvedbo teh izletov? Katere izlete bo vodil posamezni vodič?

Prevod:

TOČKE GRAFA: izleti (oziroma skupine, če je za isti izlet predvidenih več vodičev)
POVEZAVE: izlet i je povezan z izletom j natanko takrat, ko je: $(Z_i , K_i) \cap (Z_j , K_j) \neq \emptyset$
BARVE: vodiči

3. Optimizacija porabe pomnilnika

Naj bo $X = \{ X_i \}$ množica spremenljivk (istega tipa, da ne bo težav), ki nastopajo v nekem programu. Nekaj prostora prihranimo, če nekaterim spremenljivkam priredimo isti prostor; seveda tako, da dobimo ekvivalenten program. Koliko najmanj prostora je potrebnega? Katerim spremenljivkam pripada isti prostor?

Prevod:

TOČKE GRAFA: spremenljivke
POVEZAVE: točki sta povezani, če sta ustrezni spremenljivki lahko istočasno aktivni
BARVE: prostor v pomnilniku

Morda ne bo odveč, če malo natančneje opišemo, kako določimo povezanost (istočasno aktivnost) dveh spremenljivk: Sestavimo shemo programa in v njej povsod zamenjamo akcije z ustreznimi:

Use X - vrednost spremenljivke X je bila uporabljena oziroma

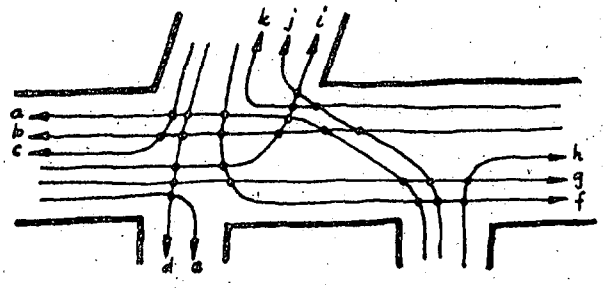
Def X - vrednost spremenljivke X je bila spremenjena

V pravilno sestavljenem programu mora biti na vsaki poti po shemi programa od začetka do nekega Use X vselej vsaj en Def X . Točki X in Y sta povezani natako takrat,

ko lahko v shemi programa najdemo pot z začetkom v Def Y in koncem v Use X , ki ne vsebuje nobene točke z oznako Def X .

4. Semaforji

Na danem križišču so predvidene določene smeri vožnje. Na primer križišče (Ajdovščina v Ljubljani):

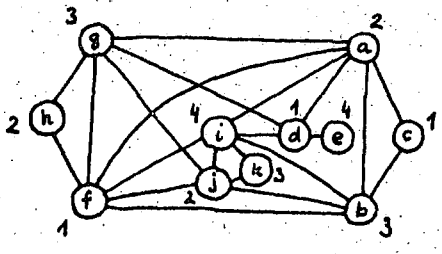


Sestavi režim delovanja semaforjev na križišču (določi istočasne smeri) z najkrajšim ciklusom

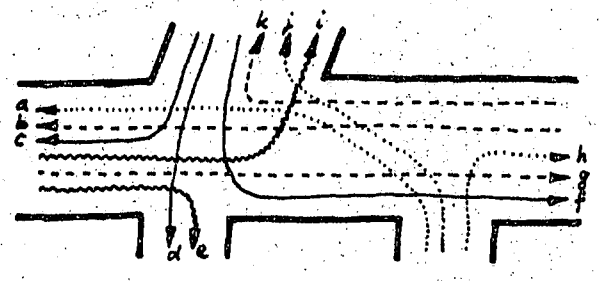
Prevod:

TOČKE GRAFA: smeri vožnje
POVEZAVE: smeri sta povezani, če se v križišču sekata (lahko pride do trčenja)
BARVE: faze semaforskega režima

V našem primeru dobimo naslednji graf, ki ga lahko pobarvamo s štirimi barvami (1,2,3,4):



in določa naslednji režim:



BARVANJA TOČK GRAFOV - OSNOVNI POJMI

Naj bo dan graf $G = (V,E)$, množica barv C in "barva" nepobarvano $\alpha \notin C$. Vpeljimo še okrajšavo $C_\alpha = C \cup \{\alpha\}$. Tedaj imenujemo **delno barvanje** točk grafa G vsako preslikavo

$$c : V \longrightarrow C_\alpha$$

ki zadošča pogoju

$$\forall u,v \in V : ((u:v) \in E \wedge c(u) = c(v) \Longrightarrow c(u) = \alpha)$$

Delno barvanje je **barvanje** natanko takrat, ko je $c(V) \subseteq C$. Torej vsako barvanje zadošča pogoju

$$\forall u,v \in V : ((u:v) \in E \Longrightarrow c(u) \neq c(v))$$

Posebna primera (delnih) barvanj sta barvanje c_α , ki je določeno s predpisom $c_\alpha : v \longmapsto \alpha$ in injektivno barvanje $\hat{c}$, ki zadošča pogoju $u \neq v \Rightarrow \hat{c}(u) \neq \hat{c}(v)$.

V nadaljevanju se bomo omejili na končne grafe - grafe, pri katerih sta množica točk V in množica povezav E končni.

Barvitost delnega barvanja c imenujemo število

$$\chi(G/c) = \mathrm{card}(c(V) \cap C)$$

Graf G je *r-(delno)obarvljiv natanko takrat, ko obstaja* (delno) barvanje c , tako da je $\chi(G/c) \leq r$. Predvsem pa je v tej zvezi zanimivo število

$$\chi(G) = \min_{c \text{ je barvanje}} \chi(G/c)$$

ki mu pravimo **barvnost** grafa G ali tudi **kromatično število.** Rečemo tudi, da je graf G χ-barven.

Podobne pojme lahko definiramo tudi za barvanja povezav. V tem sestavku se z barvanji povezav ne bomo ukvarjali.

Posamezni točki $v \in V$ grafa G lahko glede na delno barvanje c priredimo nekaj karakterističnih množic:

- **množica barv sosedov** točke v

$$\bar{C}(v/c) = \{ c(u) : (v:u) \in E \}$$

Zanjo velja ocena $\mathrm{card}(\bar{C}(v/c)) \leq d(v)$, kjer smo z $d(v)$ označili stopnjo (= število sosedov) točke v .

- **množica prostih barv** za barvanje točke v glede na c

$$C(v/c) = C \setminus \bar{C}(v/c)$$

oziroma, če dopuščamo še "barvo" nepobarvano

$$C_\alpha(v/c) = C(v/c) \cup \{\alpha\}$$

Točka $v \in V$ grafa je **zasičena** za barvanje c , natanko takrat, ko je

$$C(v/c) = \{ c(v) \}$$

Barvanje, v katerem so vse točke zasičene, imenujemo **zasičeno barvanje.**

- **barvni razred** barve $a \in C$

$$V(a/c) = \{ v \in V : c(v) = a \}$$

Očitno je vsak $V(a/c)$ neodvisna množica.

Gornje množice lahko na naraven način posplošimo na več elementov. Na primer:

$$V(a,b/c) = V(a/c) \cup V(b/c)$$

- **Kempejeve komponente** grafa G za barvi a in b imenujemo povezane komponente podgrafa

$$(V(a,b/c), E(a,b)) ,$$

kjer je

$$E(a,b) = \{ (u:v) \in E : \{ c(u), c(v) \} = \{a,b\} \}$$

Pogosto nas zanima med njimi komponenta, ki vsebuje povezavo $p(u:v)$, seveda za barvi $c(u)$ in $c(v)$. Označimo jo $K(p/c)$, oziroma, če hočemo biti določnejši $K(p;a,b/c)$, pri čemer je $a = c(u)$ in $b = c(v)$. Komponento za barvi a in b , ki vsebuje točko v , $c(v) = a$, pa označimo $K(v;b/c)$ oziroma daljše $K(v;a,b/c)$.

Pravkar definirani pojmi nam omogočajo vpeljati pri dani množici barv C naslednji transformaciji nad barvanji:

- **zamena barve** v točki v

$$c' = S(v;a/c), \quad a \in C(v/c)$$

Novo barvanje c' je določeno s predpisom

$$c'(u) = \begin{cases} c(u) & u \neq v \\ a & u = v \end{cases}$$

V primeru, ko dopuščamo tudi "barvo" nepobarvano, $a \in C_a(v/c)$, pa dobimo močnejšo transformacijo

$$c' = S^a(v;a/c)$$

ki je definirana s podobnim predpisom kot S.

- **Kempejeva zamena** glede na barvi a in b v komponenti, ki vsebuje točko v, $c(v) = a$:

$$c' = I(v;a,b/c)$$

Novo barvanje c' je določeno s predpisom:

$$c'(u) = \begin{cases} a & c(u)=b \wedge u \in K(v;a,b/c) \\ b & c(u)=a \wedge u \in K(v;a,b/c) \\ c(u) & u \notin K(v;a,b/c) \end{cases}$$

Ker nas oznake barv ne zanimajo, je smiselno izbrati nek standardni nabor barv. Pri končnih grafih lahko izberemo *na primer* $C \subset N$; *oziroma*, če želimo izbor barv še bolj omejiti, postavimo za množico barv $C_n = [1..n]$.

Naj bo c C_n-barvanje grafa G. Poglejmo si pobliže postopek:

while $\exists v : c(v) > \min C(v/c)$ **do**
$c := S(v;\min C(v/c) / c)$;

Vpeljimo funkcijo

$$\sigma(c) = \sum_{v \in V} c(v) .$$

Za vrednosti funkcije σ velja ocena $\sigma(c) \geq p$, kjer je $p = card(V)$. Ker se na vsakem koraku postopka vrednost $\sigma(c)$ zmanjša, se mora postopek v končno korakih izteči. Dobljeno končno barvanje označimo s c_{red} in mu pravimo **reducirano** barvanje.

Danemu barvanju c v splošnem pripada več reduciranih barvanj (odvisnih od zaporedja redukcij). Za vsa pa velja ocena $\chi(G/c_{red}) \leq \chi(G/c)$ in naslednja lastnost

$$\forall v \in V, \forall k \in [1..c_{red}(v)-1], \exists u \in V : \quad ((u:v) \in E \wedge c_{red}(u) = k)$$

Torej je c_{red} Grundyeva funkcija grafa G.

BARVNOST GRAFA - LASTNOSTI IN OCENE

Neposredno iz definicije barvnosti $\chi(G)$ grafa G razberemo naslednje lastnosti:

(1) $\chi(G) \leq \chi(G/c) \leq card(V)$

(2) $H \subseteq G \Rightarrow \chi(H/c) \leq \chi(G/c)$

(3) naj bodo G_i komponente povezanosti grafa G, potem je

$$\chi(G/c) = \max_i \chi(G_i/c)$$

(4) $\chi(G) \leq \Delta(G) + 1$, kjer je $\Delta(G) = \max_{v \in V} d(v)$

če v (2) in (3) postavimo namesto c barvanje c^{opt} $\chi(G/c^{opt}) = \chi(G)$, dobimo, po krajšem razmisleku

(2') $H \subseteq G \Rightarrow \chi(H) \leq \chi(G)$

(3') $\chi(G) = \max_i \chi(G_i)$

Iz lastnosti (2') dobimo takoj oceno:

(5) $\omega(G) \leq \chi(G)$

kjer je $\omega(G)$ moč največjega skupka (klike, polnega podgrafa) grafa G.

Enakost v oceni (5) velja na primer za polni graf $\omega(K_n) = \chi(K_n) = n$. Po drugi strani pa je Tutte pokazal, da obstajajo grafi z $\omega(G) = 2$ in poljubno velikim $\chi(G)$. Še močnejši je Erdös-Lovászev izrek, ki pravi, da za vsak par naravnih števil m in n obstaja n-barven graf, v katerem dolžina najkrajšega cikla presega m.

Zanimivo oceno barvnosti grafa G nam daje tudi naslednji izrek (Roy, Gallai):

Naj bo $G' = (V, A(E))$ usmerjeni graf, ki ga dobimo tako, da poljubno usmerimo povezave grafa $G = (V, E)$ in naj bo m dolžina najdaljšega enostavnega sprehoda po G', potem je

$$\chi(G) \leq m + 1$$

Königov izrek pa pravi:

$\chi(G) \leq 2$ natanko takrat, ko je G dvodelen graf.

Leta 1976 sta K. Appel in W. Haken, s pomočjo računalnika, pozitivno odgovorila na več kot sto let stari **problem štirih barv**: vsak ravninski graf je mogoče pobarvati z največ štirimi barvami. Za orientabilne ploskve višjih redov so Heawood (1890), Heffter (1891), Ringel in Youngs (1968) pokazali, da je vsak graf, ki ga lahko vložimo v orientabilno ploskev reda n 0, mogoče pobarvati z največ

$$\lfloor (7 + \sqrt{1 + 48.n}) / 2 \rfloor$$

barvami.

Zanimivi sta tudi zvezi med barvnostjo grafa G in njegovega komplementa $\bar{G}$, ki sta ju našla Nordhaus in Gaddum:

$$\lfloor 2\sqrt{p} \rfloor \leq \chi(G) + \chi(\bar{G}) \leq p + 1$$
$$p \leq \chi(G).\chi(\bar{G}) \leq \lceil ((p+1)/2)^2 \rceil$$

kjer je $p = card(V)$.

Omenimo še znani Brooksov izrek:

V povezanem nepolnem grafu G z $\Delta(G) \geq 3$ velja

$$\chi(G) \leq \Delta(G)$$

Dokaze naštetih izrekov in še nekaj drugih rezultatov najdete v (Batagelj, 1980).

ALGORITMI ZA BARVANJE GRAFOV

Raziskave na področju zahtevnosti algoritmov so pokazale, da sodi problem barvanja točk grafa med NP-polne probleme (Karp 1972, Aho, Hopcroft, Ullman 1974). Zato najbrž ne obstaja polinomsko omejen postopek za reševanje tega problema - vsi postopki, ki zagotavljajo minimalno barvanje, zahtevajo v najslabših primerih prebor skoraj vseh možnih barvanj in so pomemtakem reda $p!$.

Iz literature so znani postopki "usmerjenega" prebora, ki z domiselnim odmetavanjem neperspektivnih delnih barvanj lahko ponavadi v zmernem času (do 1000 s na računalniku, kot je CYBER) določijo minimalna barvanja grafov s tja do 100 točkami. Za najuspešnejše *med njimi so se* izkazale izpeljanke naslednje zamisli, ki jo je že leta 1949 uporabil Zykov v zvezi s kromatičnimi polinomi:

Vzemimo v grafu, ki ga želimo pobarvati poljubni nepovezani točki. Nastopita dve možnosti

- točki sta v (minimalnem) barvanju enako pobarvani;

- točki sta v (minimalnem) barvanju različno pobarvani.

Obe možnosti lahko popišemo z ustreznima transformacijama osnovnega grafa:

- če sta točki enako pobarvani, ju združimo v eno (identificiramo);

- če sta točki različno pobarvani, ju povežemo s povezavo.

Tako dobimo drevo, katerega točke so grafi, listi pa polni grafi. Stopnja polnega grafa - lista je enaka številu barv v barvanju, ki ga določa pot po drevesu od vrha do lista. Posamezni postopki se razlikujejo v načinu pregledovanja tega drevesa in v pravilih odmetavanja neperspektivnih vej.

Za zelo učinkoviti sta se izkazali naslednji, sicer preprosti, Hedetniemijevi pravili:

P1. če v grafu obstaja točka u, ki je povezana z vsemi ostalimi (po povezavi), potem po v vsakem barvanju točka u imela barvo drugačno od vseh drugih točk.

P2. če v grafu obstajata točki u in v, taki da si vsi sosedi točke u tudi sosedi točke v, potem obstaja minimalno barvanje grafa, v katerem sta obe točki enako pobarvani.

Obe pravili omogočata, da točko u "izločimo" iz grafa - nadaljnjega pregledovanja.

Nekoliko drugačen pristop je ubral Christofides, ki je pokazal, da je χ-barven graf mogoče pobarvati s χ barvami tako, da najprej pobarvamo maksimalno neodvisno množico $V_1 = N(G)$ s prvo barvo, nato z drugo barvo pobarvamo maksimalno neodvisno množico $V_2 = N(G(V \setminus V_1))$, ... Seveda moramo tudi tu pregledati vse možne maksimalne neodvisne množice.

Podrobneje so tovrstni algoritmi opisani v (Godec, 1983).

Kaj pa, če je graf prevelik ali pa je cena za iskanje minimalnega barvanja s postopkom prebora prevelika? Tedaj se moramo zateči k hevrističnim postopkom barvanja in se s tem odpovedati zagotovilu, da je dobljeno barvanje minimalno; čeprav lahko slednje včasih pokažemo, tako da najdemo polni podgraf na $\chi(G)$ točkah.

Stvar je še hujša. Garey in Johnson (1976) sta pokazala naslednje: Naj $A(G)$ označuje število barv, s katerimi pobarva postopek A dani graf G. Potem najbrž (zaradi NP-polnosti) ne obstaja polinomsko omejeni postopek, ki bi zagotavljal, da je

$$A(G) / \chi(G) \leq r < 2$$

Trenutno nista znana nobena konstanta $r > 0$ in polinomsko omejeni postopek A, za katera bi veljalo

$$A(G) / \chi(G) \leq r$$

Najboljša meja je reda $p / \log p$.

Večina hevrističnih postopkov za barvanje točk grafa zadošča shemi postopkov zaporednega barvanja:

```
c := c0
while ∃v : c(v) = ∅ do
    izberi barvo a∈C(v/c)
    c := S(v;a/c)
```

Zato, da bi opisani postopek vedno tekel, mora biti vsakozi $C(v/c) \neq 0$. Za to zadostuje, če je

$$\mathrm{card}(C) = \Delta(G) + 1 .$$

Postopki zaporednega barvanja se med seboj razlikujejo po pravilu izbire naslednjega kandidata za barvanje in po načinu izbire barve, s katero ga pobarvamo.

Preden nadaljujemo omenimo "v tolažbo in vzpodbudo" nekaj verjetnostnih rezultatov o postopkih zaporednega barvanja (Erdös, Grimmett, McDiarmid):

Naj bo $G(n,p)$ slučajni graf na n točkah, v katerem nastopa posamezna povezava z verjetnostjo p. S $\chi(n,p)$ in $\vartheta(n,p)$ pa označimo slučajni spremenljivki za barvnost oziroma število barv dobljenih z zaporednim barvanjem grafa $G(n,p)$. Tedaj veljata skoraj za vsak graf oceni:

$$\chi(n,p) \geq (1 - \varepsilon) \frac{n}{2} \log_n q^{-1}$$

in

$$\vartheta(n,p) \leq (1 + \varepsilon)\, n . \log_n q^{-1}$$

kjer je $q = 1 - p$, $p \neq 1$ in $\varepsilon > 0$.

Iz zgornjega razberemo, da velja

$$\vartheta(n,p) / \chi(n,p) \leq 2 + \varepsilon$$

Torej zaporedna barvanja v povprečju le niso tako slaba.

Povrnimo se k postopkom zaporednega barvanja. Kako lahko izberemo prosto barvo? Običajno ne razmetavamo z barvami in jih zato raje po potrebi dodajamo. Torej:

- če za dano točko obstaja prosta barva, izberemo eno izmed njih: najmanjšo, kar da reducirano barvanje; ali slučajno, kar omogoča večkratno poskušanje; ali pa najmanjkrat uporabljeno, kar da razmeroma enakomerno pobarvan graf;

- če ima točka sosede vseh dotedaj uporabljenih barv, dopolnimo množico barv z novo barvo. Včasih se temu lahko izognemo, tako da s Kempejevimi zamenami sprostimo neko barvo na sosedih dane točke.

Vsakemu zaporednemu barvanju ustreza zaporedje izbir točk, ki popisuje vrstni red barvanja. To zaporedje je v bistvu permutacija točk.

Za vsak graf G obstaja taka permutacija točk π, da da zaporedno barvanje točk v zaporedju, ki ga določa π, pri čemer vselej izbiramo najmanjšo prosto barvo, minimalno barvanje grafa G. V to se prepričamo takole: Naj bo c reducirano minimalno barvanje grafa G. Uredimo točke grafa po naraščajočih vrednostih pripadajočih barv. Dobljeno zaporedje točk določa iskano permutacijo π.

Torej je osnovno vprašanje pri zaporednih postopkih barvanja: Kako najti "ta pravo" permutacijo - določiti pravi vrstni red barvanja?

Poglejmo si nekaj možnosti:

Welsh-Powellov postopek: temelji na naslednjem receptu: permutacija π je določena s padajočim vrstnim redom stopenj d_i točk grafa G. Za WP-postopek je mogoče pokazati oceno:

Naj bo $d_1 \geq d_2 \geq d_3 \ldots \geq d_p$ padajoče zaporedje stopenj točk grafa G, potem je

$$\chi(G) \leq \mathrm{WP}(G) \leq \max \{ i : i \leq d_i + 1 \}$$

Drevesni postopki: temelje na drevesnih permutacijah: permutacija π je drevesna natanko takrat, ko za vsak $i \in [2..p]$ velja:

$$\{ u : (v_i : u) \in E \} \cap \{ v_k : k \in [1..i-1] \} \neq \emptyset$$

Ti postopki imajo naslednjo lepo lastnost

Drevesni zaporedni postopki barvanja se eksaktni za dvodelne grafe.

Primer drevesnega postopka je Brélazov postopek. Tu je permutacija π takole določena:

```
vse točke imajo vrednost 0 ,
točki, ki ima največjo stopnjo, postavi vrednost na 1
for i = 1,2, ... ,p do
    π(i) = točka z največjo vrednostjo, ki še ni v π
    vsem sosedom točke π(i)  povečaj vrednost za 1
```

V Brélazov postopek je v bistvu vgrajena ideja takojšnjega barvanja poln(ejš)ih pografov.

Szekers-Wilfov postopek: je stranski produkt pri izpeljavi naslednje ocene za barvnost grafa:

Naj preslikava $f : G \to R$ zadošča pogojema:

(1) $H \subseteq G \Rightarrow f(H) \leq f(G)$

(2) $f(G) \geq \min_{v \in V(G)} d(v)$

potem velja

$$\chi(G) \leq f(G) + 1$$

Ena izmed funkcij f, ki zadoščajo pogojema (1) in (2) je $f(G) = \max \lambda(G)$ - največja lastna vrednost grafa. Szekeres in Wilf pa sta pokazala še:

Najmanjša funkcija, ki zadošča pogojema (1) in (2) je

$$f_0(G) = \max_{H \subseteq G} \min_{v \in V(H)} d_H(v)$$

Označimo $SW(G) = f_0(G) + 1$

Ker je vedno $SW(G) \leq WP(G)$, nima smisla programirati WP-postopka. Tako nam ostaneta na izbiro še SW-postopek in Brélazov postopek. Izkaže se, da ju lahko združimo v enoten postopek odvisen od dveh parametrov r in s:

```
določi stopnje točk  d[v] , v∈V ;
sw := min {d[v] : v∈V};
val := 0 ;  W := V ;
for i := 1 to p do
    v := argmin {d[w] : w∈W};
    if d[v] > sw then
        val := val + r ;  sw := d[w]
    endif ;
    d*[v] := val ;  W := W \ {v} ;
    for u∈EXT(STAR(v)) ∩ W do
        d[u] := d[u] - 1
    endfor
endfor ;
W := V ;
for i := 1 to p do
    v := argmax {d*[w] : w∈W};
    col[v] := selectcolor ;
    W := W \ {v} ;
    for u∈EXT(STAR(v)) ∩ W do
        d*[u] := d*[u] + s
    endfor
endfor ;
```

Gornji postopek nam pri $r = 0$ in $s \neq 0$ da Brélazov postopek; pri $r \neq 0$ in $s = 0$ pa SW-postopek; zanimive pa so tudi njune kombinacije pri neničelnih r in s.

Postopek učinkovito sprogramiramo tako, da organiziramo d in d^* v kopico (lahko celo isto).

Na računalniku DEC-10, Univerze v Ljubljani je postopek vgrajen v program COLORS programskega paketa za delo z grafi GRAPH. Program COLORS v zmernem času določi dobra barvanja grafov na do tisoč točkah in nekaj tisoč povezavami.

LITERATURA

[1] Aho A.V., Hopcroft J.E., Ullman J.D.: The design and analysis of computer algorithms. Addison Wesley, Reading, Mass. 1974

[2] Batagelj V.: Barvanja točk grafov. Seminar za numerično in računalniško matematiko IMFM-201, Ljubljana, november 1980

[3] Berge C.: Graphes et Hypergraphes (2. ed.). Dunod, Paris 1973

[4] Brélaz D.: New methods to color the vertices of a graph. CACM, 22(1979)4, 251-256.

[5] Carré B.: Graphs and networks. Clarendon Press, Oxford 1979.

[6] Catlin R.A.: Brooks' graph-coloring theorem and the independence number. J. of comb. Theory, series B, 27(1979), 42-48.

[7] Christofides N.: An algorithm for the chromatic number of a graph. The Computer Journal, 14(1971), 38-39.

[8] Cole A.J.: The preparation of examination time-tables using a small-store computer. The Computer Journal, 7(1964), 117-121.

[9] Corneil D.G., Graham B.: An algorithm for determining the chromatic number of a graph. SIAM J. Comput., 2(1973), 311-318.

[10] Cvetković D., Milić M.: Teorija grafova i njene primene. Naučna knjiga, Beograd 1977.

[11] Deming R.W.: Acyclic orientations of a graph and chromatic and independence numbers. J. of Comb. Theory, series B, 26(1979), 101-110.

[12] Garey M.R., Johnson D.S.: The complexity of near-optimal graph coloring. JACM, 23(1976), 43-49.

[13] GRAPH - programi. Priročnik za DEC-10, Ljubljana, 1982.

[14] Graver J.E., Watkins M.E.: Combinatorics with emphasis on the theory of graphs. Springer-Verlag, New York 1977.

[15] Godec H.: Algoritmi za barvanje grafov. FNT, matematika, diplomsko delo 279, Ljubljana 1983.

[16] Grimmett G.R., McDiarmid C.J.H.: On colouring random graphs. Math.Proc. Camb. Phil. Soc., 77(1975), 313-324.

[17] Harary F.: Graph theory. Addison-Wesley, Reading, Mass. 1969.

[18] Hedetniemi S.T.: Review no. 22063. Comput. Rev. 12(1971), 446-447.

[19] Karp R.M.: Reducibility among combinatorial problems (Complexity of Computer Computations, Miller R.E., Thatcher J.W., Ed.). Plenum Press, New York 1972, 85-103.

[20] Karp R.M.: The probabilistic analysis of some combinatorial search algorithms (Algorithms and Complexity, Traub J.F., Ed.). Academic Press, New York 1976, 1-19.

[21] Matula D., Marble G., Isaacson J.: Graph coloring algorithms (Graph Theory and Computing, Read R.C., Ed.). Academic Press, New York 1972, 109-122.

[22] McDiarmid C.J.H.: Colouring random graphs badly. (fotokopija iz neznanega zbornika).

[23] Melnikov L.S., Vizing V.G.: New proof of Brooks' theorem. J. of Comb. Theory, 7(1969), 289-290.

[24] Nijenhuis A., Wilf H.S.: Combinatorial algorithms (2. ed.). Academic Press, New York 1978.

[25] Ore O.: The four color problem. Academic Press, New York 1967.

[26] Peršin O.Ju.: Algoritm opredelenija minimalnoj raskraski konečnogo grafa. Izvestija akademii nauk SSSR, Tehničeskaja kibernetika, (1973)6, 114-119.

[27] Szekeres G., Wilf H.S.: An inequality for the chromatic number of a graph. J. of Comb. Theory, 4(1968), 1-3.

[28] Welsh D.J.A., Powell M.B.: An upper bound for the chromatic number of a graph and its application to timetabling problems. The Computer Journal, 10(1967), 85-86.

32-BITNI MIKROPROCESOR NS 32032

BRANKO SOVDAT

DO ISKRA AVTOMATIKA

UDK: 681.3.06

Članek najprej na kratko opiše mikroprocesorsko družino NS 16000. Nato se podrobneje ustavi na članu te družine, na 32 bitnem mikroprocesorju NS 32032. V članku so podrobneje obravnavane glavne lastnosti tega mikroprocesorja in sicer: prekinitve, nabor ukazov in načini naslavljanja. Nazadnje sledi še opis glavnih perifernih enot. To sta računska enota FPU, ki omogoča računanje osnovnih štirih računskih operacij realnih števil in enota za nadzor nad pomnilnikom MMU. Pri enoti MMU je podrobneje opisan predvsem način virtualnega naslavljanja, zaščite podatkov v pomnilniku in zasledovanje poteka programa.

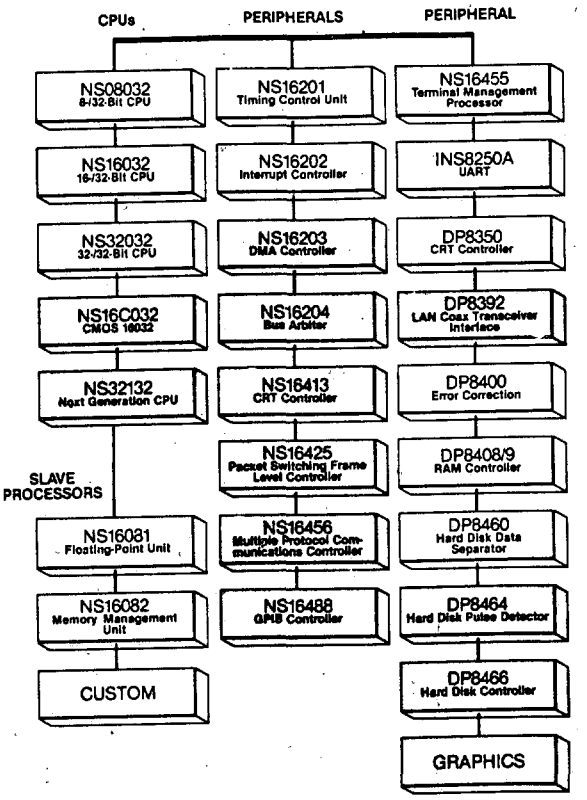
There is first a brief description of the units of a National Semiconductor's NS 16000 family. Then it follows a description of a NS 32032, a 32 bit control processing unit. The NS 32032 is a member of a generation of 32 bit CPUs, so the particular importance is given to the data about interrupts, instruction set and adressing modes.
The other important units of that family are slave processors: a floating point unit (FPU) and memory managment unit (MMU). A floating point unit is used for numerical operations with real numbers. It is also described a memory managment unit, specialy a way of virtual memory managment, program debuging and memory protection.

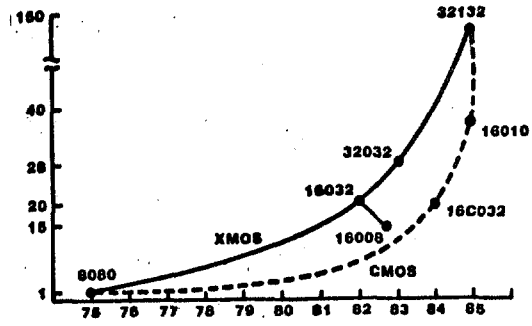
1. OPIS DELOVANJA PROCESORJA NS 32032

1.1 Družina NS 16000

NS 32032 je 32 bitni mikroprocesor iz družine NS 16000, ki je izdelek družbe National Semiconductor. Na sliki 1 vidimo enote iz družine NS 1600. Kot vidimo, vsebuje ta družina še nekaj drugih CPU, računsko enoto, enoto za nadzor nad pomnilnikom in še vrsto perifernih enot (DMA kontroler, prekinitveni(interrupt) kontroler, CRT kontroler, disk kontroler in še nekatere druge). Na sliki 2 imamo na ordinatni osi nanešeno zmogljivost posameznih CPU iz te družine v primerjavi z znanim procesorjem 8080, na abscisni osi pa je podan približen čas, ko so (ali bodo) te enote postale dostopne kupcem. Programsko so vse CPU (za 32132 in 16010 še ni podatkov) popolnoma enake. Navzven se obnašajo kot osem (08032), šestnajst (16032, 16C32) ali dvaintrideset (32032) bitni mikroprocesorji, medtem, ko je njihova notranja zgradba 32 bitna.

Nedvomno je od sedaj dosegljivih CPU iz te družine najmočnejši ravno NS 32032. V povezavi z računsko enoto FPU (floating point unit) in enoto MME (memory managment unit), ki opravlja funkcijo nadzora nad pomnilnikom, dobimo jedro dokaj močnega in sposobnega mikroračunalniškega sistema.





Slika 2: različne CPU iz družine NS 16000

1.2. Pomembnejši priključki procesorja NS 32032

Procesor ima enojno 5V napajanje, dva vhoda za urina (clock) signala, ki sta med sabo fazno premaknjena, multipleksirano podatkovno in adresno vodilo in večje število kontrolnih signalov. Adresnih linij je 24, kar omogoča naslovitev 16M spominskih lokacij. Naslavlja lahko vsak byte posebej, lahko pa tudi do štiri byte hkrati. Ima dva prekinitvena vhoda, in sicer maskirani (INT) in nemaskirani (NMI) prekinitveni vhod. Reset/abort (RST/ABT) vhod ima dve funkciji. Lahko resetira procesor in sicer v primeru, če traja aktivno stanje tega priključka dalj kot je urin cikel. Kadar pa traja aktivno stanje na reset/abort vhodu en urin cikel pride do abortiranja tekočega opravila. Izmed kontrolnih signalov velja omeniti še statusne signale ST0-ST3, ki nam podajajo informacijo v katerem izmed šestnajstih možnih stanj se nahaja procesor. Imamo še vhod RDJ, ki nam omogoča priključitev tudi počasnejših spominskih enot ali perifernih naprav. Postavitev tega vhoda v neaktivno stanje podaljša procesorjeve cikle, z vstavitvijo takoimenovanih 'WAIT' stanj, dokler se vhod RDJ ne vrne nazaj v aktivno stanje.

1.3. Programski model procesorja

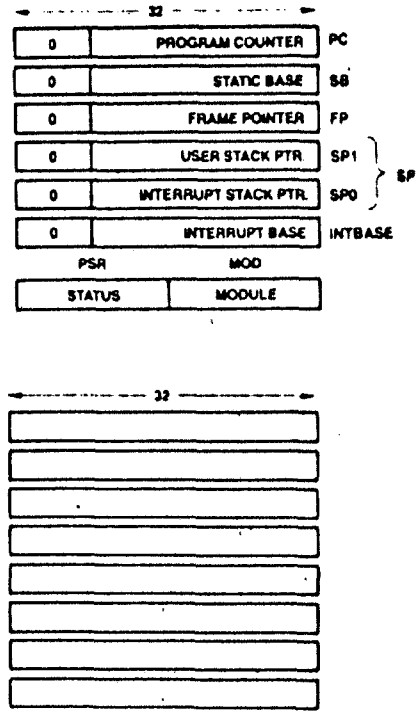
Slika 3 nam prikazuje programski model procesorja. Vidimo, da ima 16 registrov, od katerih je osem namenskih (dedicated) in osem uporabniških (general purpose). Uporabniški se uporabljajo za shranjevanje začasnih spremenljivk in naslovov, ter za operacije nad njimi. Vsi uporabniški registri so dolžine 32 bitov.

Imamo še osem namenskih registrov in sicer:

- programski števec PC

- prekinitveni skladovni kazalec (interrupt stack pointer) SP0 in uporabniški skladovni kazalec (user stack pointer) SP1. V odvisnosti od tega v kakšnem načinu delovanja je procesor (uporabniški ali nadzorni), je dostopen eden od teh dveh skladov. Kadar je procesor v uporabniškem načinu delovanja je dostopen uporabniški sklad, na katerega kaže skladovni kazalec SP1. V uporabniškem skladu se hrani začasne podatke ali vrnitvene naslove podprogramov. V primeru pa, da je procesor v nadzornem načinu (supervizor mode), pa je dostopen prekinitveni sklad, na katerega kaže skladovni kazalec SP0. Prekinitveni sklad se navadno uporablja za potrebe operacijskega sistema: shranjevanje začasnih podatkov, vrnitvenih naslovov sistemskih rutin in prekinitvenih programov.

- register FP (frame pointer), ki ga uporabljajo podprogrami za dostop do parametrov in lokalnih spremenljivk v skladu. FP register se naloži pri klicanju podprograma. Pri izhodu iz podprograma se spet obnovi na staro vrednost.



Slika 3: programski model NS 32032

- register SB (static base) vsebuje naslov začetka globalnih spremenljivk programskega modula, ki je v izvajanju.

- register INTBASE (interrupt base) vsebuje naslov začetka tabele zunanjih prekinitev in programskih ali trap prekinitev.

- MOD (module) register vsebuje naslov glavnih podatkov programskega modula, ki je v izvajanju.

- statusni register PSR (processor status register) vsebuje statusno besedo tega procesorja. Celotna beseda je dostopna samo v nadzornem načinu delovanja.

Vsi namenski registri razen MOD in PSR so 32 bitni. Zgornjih osem bitov teh registrov je zmeraj nič, registra MOD in PSR pa sta 16 bitna.

Imamo pa še en štiribitni register. To je konfiguracijski (configuration) register, ki vsebuje informacijo o prisotnosti nekaterih enot v sami hardware-ski konfiguraciji mikroračunalnika. V primeru prisotnosti enote setiramo ustrezni bit v konfiguracijskem

registru. Ti biti se nanašajo na prekinitveni (interrupt) kontroler, na računsko enoto FPU, na enoto za nadzor nad pomnilnikom in za enoto, ki jo deklarira sam uporabnik (custom).

V primeru odsotnosti prekinitvenega kontrolerja se izvaja ne-vektorski način prekinitev, ki bo opisan pozneje. V kolikor pa je resetiran kateri od ostalih treh bitov konfiguracijskega registra, ki se nanašajo na računsko enoto, enoto za nadzor nad pomnilnikom ali pa uporabniško enoto, računalnik v tem primeru ne prepozna ustreznega seta ukazov. V primeru uporabe takih ukazov računalnik generira ustrezno programsko (trap) prekinitev.

1.4. Postopek izvajanja prekinitev

V tem razdelku si bomo ogledali protokol izvajanja prekinitev. Po nastanku moramo ločiti dve vrsti prekinitev:

- hardware-ska ali zunanja prekinitev, ki nastane zaradi zahteve periferije.

- programska ali trap prekinitev, ki se sproži zaradi posebnih razmer, do katerih pride med izvajanjem programa.

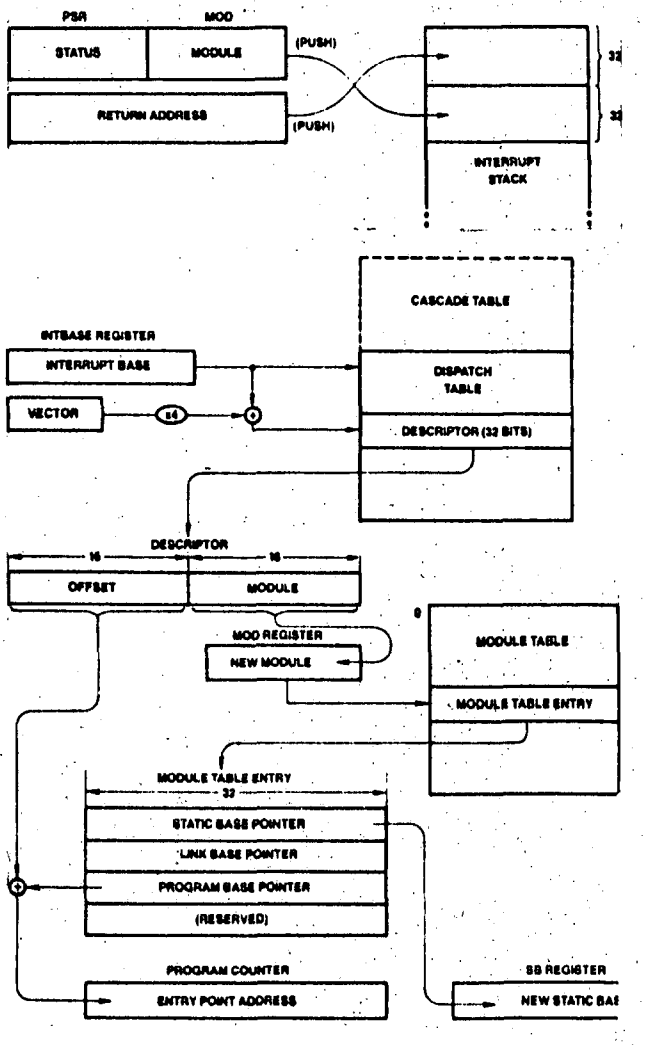
Potek izvajanja prekinitev bi lahko v grobem delili na štiri faze. Potek prekinitve nam prikazuje slika 4. Faze prekinitve:

- nastavitev registrov: nastavi se pravilna vrednost programskega števca (odvisno od vrste prekinitve), statusnega registra in tekočega skladovnega kazalca. Napravi se kopija statusnega registra, nato pa se procesor postavi v nadzorni način delovanja. S tem tudi avtomatsko pride do izbire prekinitvenega sklada.

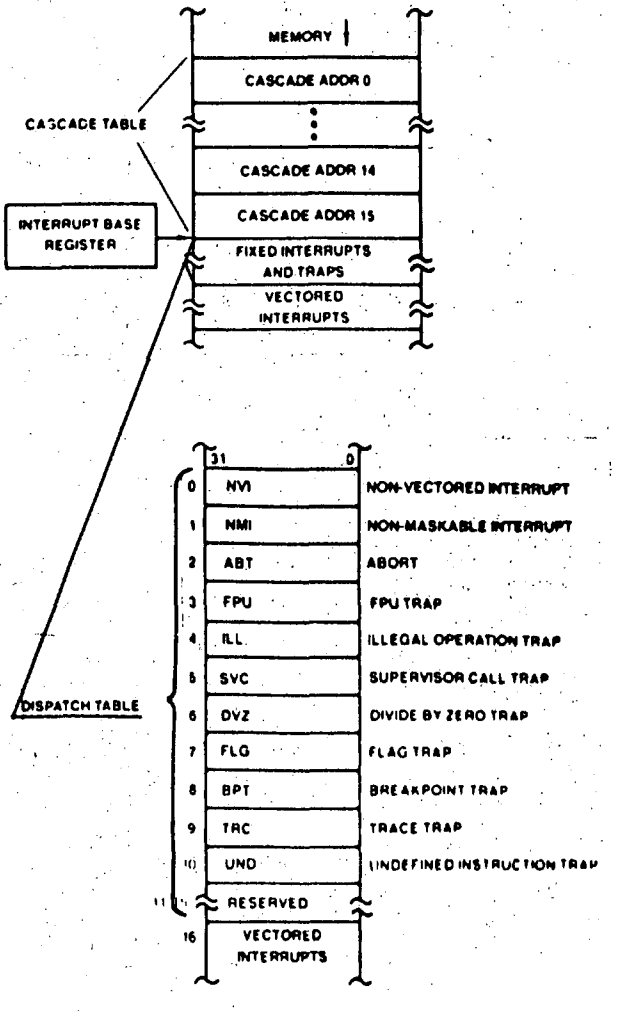
- shranitev statusnega registra (PSR), MOD registra in programskega števca, ki vsebuje vrnitveno adreso, v sklad.

- branje vektorja (8 bitna vrednost) s podatkovnega vodila.

- klic prekinitvene rutine: procesor kontrolira stanje bita, ki kaže prisotnost prekinitvenega kontrolerja (eden izmed bitov konfiguracijskega registra). V primeru, da je ta resetiran se prebrani vektor zavrže, če pa je setiran se vektor uporabi za skok v prekinitveno tabelo. Začetek prekinitvene tabele je definiran z INTBASE registrom, vektor pa predstavlja odmik od tega naslova. V primeru trap prekinitev se za vsako vrsto prekinitve nastavi določena vrednost vektorja. Slika 5 kaže prekinitveno tabelo. Pred skokom v



slika 4: klic prekinitvene sekvence

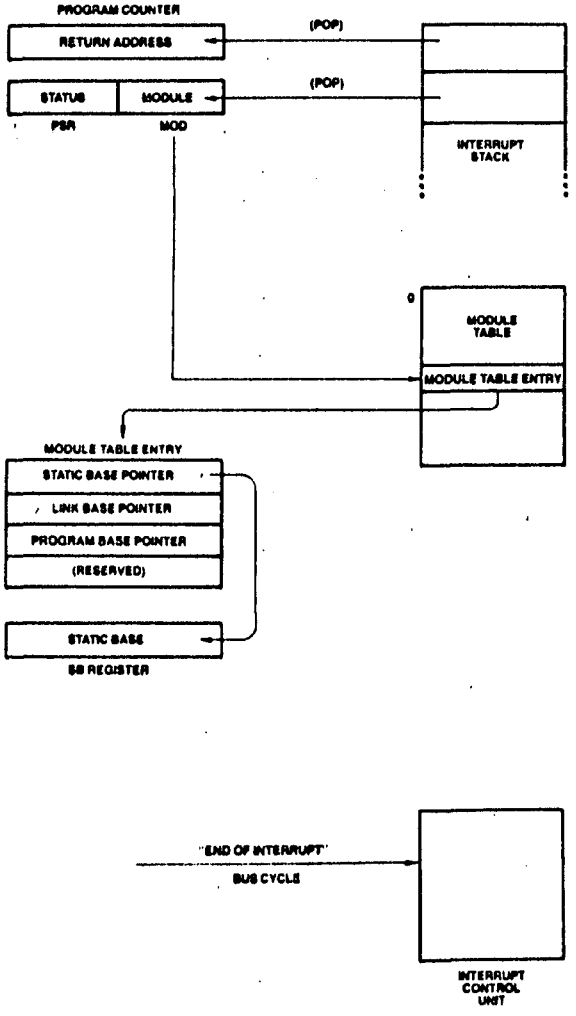


slika 5: prekinitvena tabela

prekinitveno tabelo, se shrani v sklad še MOD register (16 bitni) in vrnitvena adresa (tekoči PC). Nato pride do skoka na ustrezno mesto prekinitvene tabele, ki je definirano z INTBASE registrom in vektorjem.

V prekinitveni tabeli se nahaja 32 bitna vrednost, ki pomaga pri določitvi vseh potrebnih parametrov prekinitvenega programa. 16 bitov iz te tabele se naloži v MOD register, ki kaže na vrh tabele (module table entry), kjer so shranjeni glavni podatki prekinitvenega programa. Iz te tabele se prebere podatek o začetku programskega dela (program base pointer), ki skupaj z ostalimi 16 biti iz prekinitvene tabele (offset) določa vhodni naslov v prekinitveni program.

Vrnitev iz prekinitvenega programa se izvede z ukazom RETT (return from trap). Obnovijo se vsebine programskega števca, statusnega registra, MOD registra in SB registra. Vse to je prikazano na sliki 6. Za vrnitev iz maskirane prekinitve pa se uporablja ukaz RETI, ki še dodatno obvesti prekinitveni kontroler, da je prekinitveni program zaključen.



slika 6: zaključek prekinitve

1.5 Vrste prekinitev

1.5.1 Abort prekinitev

Do abort prekinitve pride, če se na RST/ABT priključku procesorja pojavi aktiven signal, katerega trajanje znaša eno periodo urinega signala. Abort signal lahko procesorju pošlje enota za nadzor nad pomnilnikom (MMU), in sicer zaradi enega od naslednjih dveh razlogov:

- CPU zahteva dostop do podatkov, ki so na virtualnih naslovih, niso pa fizično prisotni v pomnilniku in jih je treba vanj naložiti iz zunanjega pomnilniškega medija (npr. diska).

- CPU zahteva dostop do podatkov, do katerih ne more priti, ker so zaščiteni pred dostopom na tem nivoju.

Kadar MMU sproži abortiranje določenega ukaza, se po izvededbi abort prekinitve lahko abortirani ukaz ponovno izvede, saj je vrnitveni naslov iz abort prekinitve naslov abortiranega ukaza.

1.5.2 Nemaskirana prekinitev

Nemaskirana prekinitev se sproži, kadar pride do aktivnega prehoda na NMI vhodu. Izvajanje prekinitve je opisano v poglavju 1.4. Za vrnitev v glavni program se uporabi ukaz RETT.

1.5.3 Maskirana prekinitev

Sproži se na vhodu INT, ko ta pride v aktivno stanje, če je maskirni bit v statusnem registru setiran. V kolikor je maskirni bit resetiran, do prekinitve ne pride. V odvisnosti od vrednosti konfiguracijskega registra in konfiguracije prekinitvenih kontrolerjev imamo naslednje načine prekinitev:

- nevektorski način: pri katerem se prebrani vektor ne upošteva. Program skoči na začetek prekinitvene tabele, katere naslov vsebuje INTBASE register. Vektorja se ne upošteva, ker je v konfiguracijskem registru resetiran bit, ki kaže na prisotnost prekinitvenega kontrolerja.

- vektorski način: v tem načinu lahko sprejemamo do 16 različnih prekinitev. Vektor je pozitivno 7 bitno število.

- kaskadni način: ta način nam omogoča sprejemati do 256 različnih prekinitev. Prekinitveni kontrolerji so kaskadno povezani. Vektor je negativno število od -16 do -1. V tabeli, ki je naslovljena s tem vektorjem, pa se nahaja naslov, od koder CPU prebere končni vektor.

1.5.4 Programske ali trap prekinitve

To so prekinitve, ki se pojavijo kot rezultat izvajanja določenega ukaza in niso vezane na zunanje dogodke. Vrnitveni naslov vseh trap prekinitev razen trap(TRC) je naslov začetka ukaza, pri kateri se je prekinitev pojavila. NS32032 pozna naslednje trap prekinitve:

- trap(FPU) se pojavi zaradi detekcije napake v računski enoti FPU. CPU sprejme to obvestilo hkrati z rezultati iz računske enote.

- trap(ILL) označuje nedovoljeno operacijo. To se zgodi v primeru, da je tekoči ukaz priviligiran (se lahko izvaja samo v nadzornem načinu delovanja), procesor pa je v uporabniškem načinu delovanja.

- trap(SVC) se generira zaradi zahteve tekočega ukaza po spremembi načina delovanja iz uporabniškega v nadzorni.

- trap(DVZ) pomeni, da je bilo zahtevano celoštevilsko deljenje z nič.

- trap(FLG) se pojavi, kadar se postavi bit F v statusnem registru. Ta bit je uporabljen v več namenov. Tako npr. v celoštevilčni aritmetiki služi za indikacijo prekoračitve največje vrednosti (owerflow).

- trap(TRC) se pojavi ob zaključku vsakega ukaza, če je T bit statusnega registra setiran. Če je ta bit resetiran se ta prekinitev ne sproži. V kolikor se med trap(TRC) prekinitvijo pojavi kaka druga prekinitev, se le-ta izvede prej in se izvajanje trap(TRC) prekinitve nadaljuje šele ob koncu nove prekinitve. To se zgodi zato, ker ima trap(TRC) prekinitev najmanjšo prioriteto.

- trap(UND) se zahteva, kadar CPU naleti na operacijsko kodo neznanega ukaza. To se zgodi tudi v primeru, če se zahteva izvedba ukaza, ki sodi v nabor ukazov periferne enote, katere ustrezni bit v konfiguracijskem registru je resetiran.

Programske prekinitve se izvedejo podobno kot hardware-ske prekinitve, le da namesto branja vektorja s podatkovnega vodila, dobi vsak trap prirejeno določeno vrednost vektorja, ki je za vsak trap različna in s tem svoje mesto v prekinitveni tabeli. Vrnitev iz prekinitvene rutine poteka z ukazom RETT.

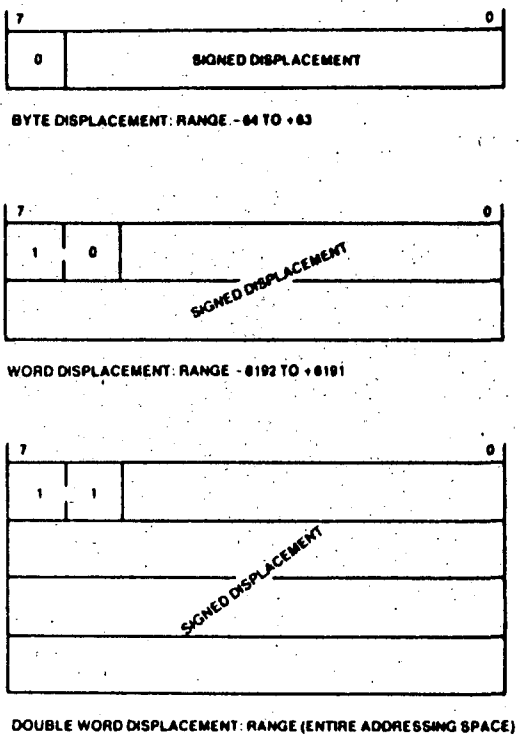
1.6 Prioriteta prekinitev

Po prioriteti so zahteve za prekinitev razporejene od najvišje proti najnižji na naslednji način:

- vse trap prekinitve razen trap(TRC)
- abort prekinitev
- nemaskirana prekinitev (NMI)
- maskirana prekinitev (INT)
- trap(TRC) prekinitev

1.7 Načini naslavljanja pri NS32032

Ta procesor pozna devet načinov naslavljanja. Vsi operandi v naštetih načinih naslavljanja so lahko 8 bitni (byte), 16 bitni (word) in 32 bitni (double word). Razlika ali odmik v ukazih pa je lahko 7, 14 ali 30 bitna, kar je razvidno s slike 7. Načini naslavljanja so:



slika 7: odmik ali razlika (displacment), ki se uporablja pri ukazih

- takojšnje (immediate) naslavljanje; operand je določen že v samem ukazu.

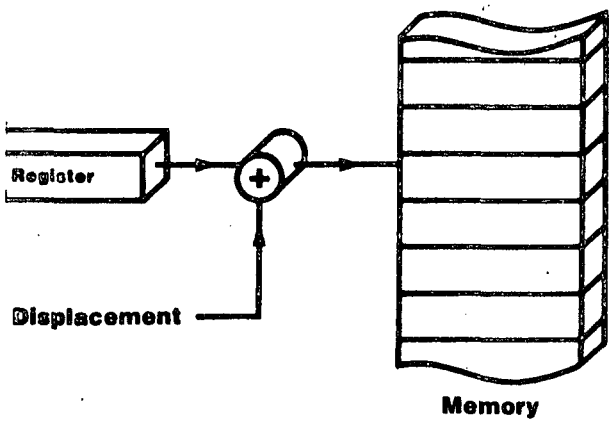
- absolutno (absolute) naslavljanje; naslov operanda je podan s poljem, ki vsebuje odmik in je v samem ukazu.

- registrsko (register) naslavljanje; operand se nahaja v enem izmed osmih uporabniških registrov.

- relativno registrsko (register relative) naslavljanje; eden izmed uporabniških registrov vsebuje naslov. K temu naslovu se prišteje še razlika (odmik) iz ukaza in dobimo naslov operanda. Relativno registrsko naslavljanje je prikazano na sliki 8.

- spominsko direktno (memory space) naslavljanje; k naslovu, ki ga vsebuje eden izmed namenskih registrov (PC, SP, SB ali FP) se prišteje še razlika iz ukaza in se tvori naslov operanda.

- spominsko relativno (memory relative) naslavljanje; k vsebini enega izmed registrov SB, SP ali FP se prišteje še razlika 1 iz ukaza in tvori naslov spominske lokacije. Na temu naslovu in še na naslednjih treh je 32 bitna vsebina. K tej novi vsebini se prišteje še razlika 2 iz ukaza in tvori naslov operanda. Ta način naslavljanja nam prikazuje slika 9.

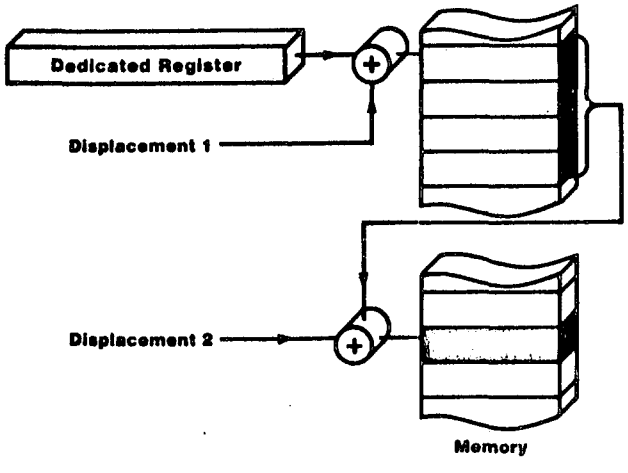


slika 8: relativno registrsko naslavljanje (register je uporabniški) ali spominsko direktno naslavljanje (register je namenski)

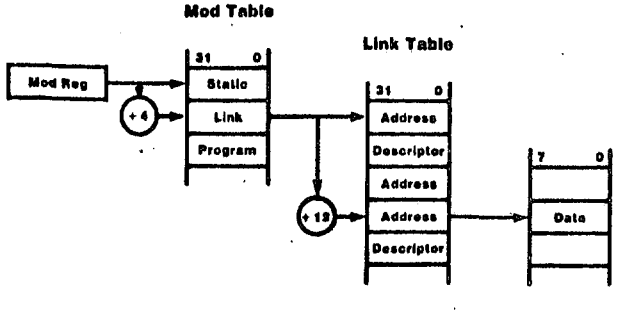
- skladovno (top of stack) naslavljanje; na lokacijo, ki je naslovljena s skladovnim kazalcem SP lahko vpisujemo ali beremo podatke z ukazoma PUSH in POP, ali pa zamenjujemo vrednost z ukazom MODIFY.

- zunanje (external) naslavljanje; V MOD registru dobimo naslov MOD tabele. V tej tabeli je na določenem mestu naslov LINK tabele, v kateri se nahajajo naslovi zunanjih podatkov. K naslovu iz LINK tabele se prišteje še razlika iz ukaza in dobimo naslov zunanjih podatkov. Zunanje naslavljanje nam prikazuje slika 10.

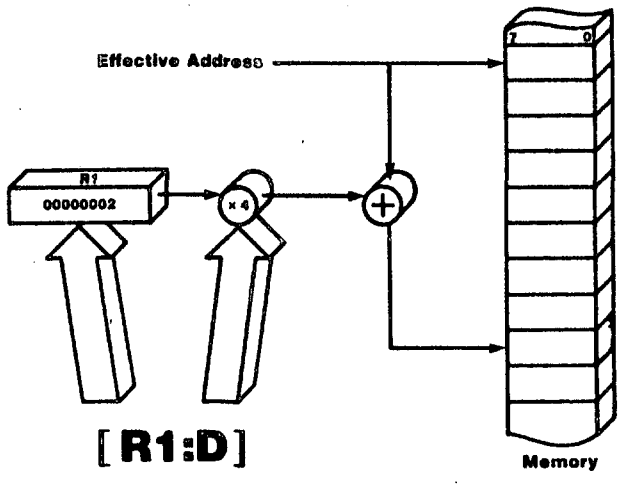
- naslavljanje s skalnimi indeksi (scaled index); to naslavljanje lahko uporabimo v kombinaciji s prejšnjimi načini naslavljanja, razen s takojšnjim (*immediate*) naslavljanjem, ali že z onim naslavljanjem s skalnimi indeksi. Naslov operanda pri tem načinu se izračuna tako, da se k obstoječemu naslovu prišteje vsebino enega uporabniškega registra, pomnoženo z vrednostjo skalnega indeksa. Skalni indeks lahko zavzame vrednosti 1, 2, 4 ali 8. Primer takega naslavljanja je prikazan na sliki 11.



slika 9: spominsko relativno naslavljanje



slika 10: zunanje naslavljanje



slika 11: naslavljanje s skalnimi indeksi

1.8 Pregled ukazov

Ukazi so po podobnosti grupirani v naslednje logične skupine:

1.8.1 Premiki

V tej grupi imamo 8 ukazov, katerih vsak lahko uporabimo za vse naštete načine naslavljanja. Z njimi lahko realiziramo premike ali zamenjave vsebin med registri, med registri in pomnilnikom ali med različnimi pomnilniškimi lokacijami. Lahko izvedemo tudi premike manjših blokov podatkov.

1.8.2 Celoštevilska aritmetika

To skupino sestavlja 14 ukazov, ki se uporabljajo za omenjene načine naslavljanja. Imamo ukaze za celoštevilsko seštevanje in odštevanje, s prenosom ali brez njega, potem negiranje, celoštevilsko množenje in deljenje, absolutno vrednost števila, ostanek pri deljenju in druge.

1.8.3 Operacije z BCD števili

V tej skupini imamo dva ukaza in sicer za odštevanje in seštevanje BCD števil.

1.8.4 Celoštevilsko primerjanje

Zastopano je s tremi ukazi, ki opravljajo funkcijo primerjanja dveh operandov. Rezultat teh operacij je nastavitev ustreznih bitov v statusnem registru. Eden izmed ukazov omogoča tudi primerjavo bloka podatkov.

1.8.5 Logične operacije

V to skupino sodi sedem ukazov, med njimi AND, OR, XOR, COMPLEMENT in še nekateri drugi.

1.8.6 Pomikanje in vrtenje

Ta grupa je sestavljena iz treh ukazov in sicer: logični pomik v levo ali desno za N mest, aritmetični pomik v levo ali desno za N mest in rotacija v levo ali desno za N mest.

1.8.7 Manipulacija s posameznimi biti

V tej skupini imamo sedem ukazov. Naslavljanje posameznih bitov je realizirano na ta način, da naslov kaže na prvi (LSB) bit naslovljenega byta. Ostali biti pa so naslovljeni z odmikom od začetnega. Operacije, ki jih izvajajo ukazi te skupine pa so: testiranje bitov, testiranje in setiranje bitov, testiranje in resetiranje bitov, testiranje in setiranje z zaklepanjem (interlock), testiranje in resetiranje z zaklepanjem, testiranje in invertiranje ter iskanje prvega postavljenega bita.

1.8.8 Skupine bitov poljubnih dolžin

Ukazi za operacije s skupinami bitov poljubnih dolžin,nam omogoča memorizacijo zapisov, katerih dolžina ni mnogokratnik osem, ne da bi ostala v pomnilniku 'prazna' mesta. Imamo pet ukazov, ki nam omogočajo vpisovanje besed različnih dolžin v spomin ali iz njega in prenose teh besed med spominskimi lokacijami.

1.8.9 Večdimenzionalna polja

Iz te grupe imamo dva ukaza. Prvi testira indekse in ugotovi ali so v deklariranem področju, drugi pa sodeluje pri izračunu fizičnega naslova podatkov, ki so podani z indeksi polja.

1.8.10 Zančne operacije

Na voljo imamo šest ukazov, ki ponavljajo izvajanje neke operacije v odvisnosti od nekega logičnega pogoja ali števca, ki šteje število ponavljanj. V to skupino spadajo tudi ukazi za premikanje blokov podatkov, za primerjanje, oziroma iskanje določene vrednosti v bloku podatkov, ter skoki na določena mesta v odvisnosti od vrednosti, ki jih zavzame neka spremenljivka.

1.8.11 Skoki, klici in vrnitveni ukazi

Ta skupina je zastopana z 18 ukazi. Skoki so absolutni in relativni. Imamo tudi ukaz, ki ima podobno nalogo kot paskalski CASE stavek. Izmed klicev imamo klice podprogramov in zunanjih podprogramov, klic nadzornega načina izvajanja, klic vhoda v podprogram in izhoda iz podprograma. Vrnitveni ukazi se uporabljajo za vrnitve iz podprogramov, zunanjih podprogramov, iz trap prekinitev in zunanjih prekinitev.

1.8.12 Manipulacija z registri CPU

To skupino imamo zastopano z osmimi ukazi. Omogočajo nam shranitev vsebine registrov v sklad ali vpis vsebine iz sklada v registre. Imamo tudi možnost postavitve posameznih bitov statusnega registra, ter nastavitve posameznih bitov konfiguracijskega registra, ki vsebuje informacijo o fizični prisotnosti nekaterih enot v sistemu.

1.8.13 Mešani ukazi

V tej skupini imamo tri ukaze. Ukaz NOP (no operation) ohrani stanje, ki je bilo pred njeno izvršitvijo. Ukaz WAIT postavi procesor v stanje, v katerem čaka na prekinitev. Ostane še ukaz DIA, ki je diagnostični ukaz in se uporablja pri testiranju sistema.

1.8.14 Ukazi podrejenih (slave) procesorjev

Ukazi te grupe se delijo na tri skupine in sicer glede na enote (slave procesorji), ki jih nadzorujejo. To so računska enota FPU, enota za nadzor nad pomnilnikom (MMU) in uporabniško definirana enota. V teh treh skupinah imamo 43 ukazov, ki omogočajo popoln nadzor nad temi enotami. Uporabljajo se lahko samo ukazi za enoto, za katero je bit v konfiguracijskem registru postavljen.

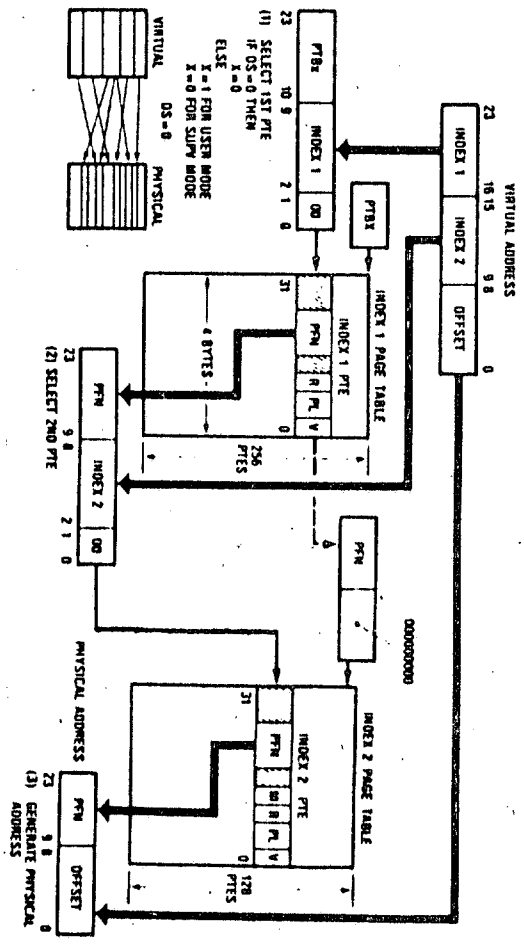
2. PODREJENI (SLAVE) PROCESORJI

2.1 Enota za nadzor nad pomnilnikom -MMU

Ta enota nam omogoča avtomatsko prevedbo virtualnih naslovov v fizične preko tabel v sami enoti ali v pomnilniku. Imamo tudi možnost slediti poteku programa, postavitve breakpoint naslovov in zaščito željenih delov pomnilnika.

2.1.1 Prevedba virtualnih naslovov v fizične

Ta prevedba nam omogoča enako obravnavanje podatkov, ki so shranjeni na disku ali drugem podobnem mediju, kot tistih, ki so v pomnilniku. Tem podatkom pripišemo nek virtualni naslov. Kadar zahtevamo podatke na nekem virtualnem naslovu, pride do avtomatske prevedbe virtualnega naslova v fizični, če so ti podatki v pomnilniku. V kolikor pa teh podatkov ni v pomnilniku, potem mora operacijski sistem poskrbeti za prenos podatkov iz zunanjega pomnilniškega medija v glavni pomnilnik. Ko je to opravljeno, lahko pride do pretvorbe virtualnega naslova v fizični. V ta namen podatke razdelimo na strani, ki imajo dolžino 512 bytov. Način prevedbe virtualnih naslovov v fizične pa nam kaže slika 12.



slika 12: prevedba virtualnih naslovov v fizične

Na sliki 12 vidimo, da je virtualni naslov, ki ima dolžino 24 bitov, sestavljen iz treh delov: dveh indeksnih vrednosti INDEX1 in INDEX2 in odmika OFFSET. Indeksni vrednosti naslavita stran, odmik pa naslovi ustrezni byte znotraj strani. Najprej se iz PTBR (page table base) registra in vrednosti INDEX1 iz virtualnega naslova sestavi naslov tabele strani prvega indeksa (index1 page table). Vrednost, ki se jo dobi iz te tabele, nam tvori skupaj z vrednostjo INDEX2, naslov tabele strani drugega indeksa. Vrednost, ki se jo dobi iz te druge tabele, pa tvori, skupaj z odmikom (OFFSET) iz virtualnega naslova, fizično adresn podatka, ki je bil naslovljen z virtualnim naslovom.

Tabele, ki jih potrebujemo za transformacijo, so shranjene v pomnilniku. V ta namen MMU prevzame nadzor nad vodili in prebere iz tabele transformacij vrednost fizičnega naslova ter jo postavi na vodila. MMU pa lahko tudi sama shrani naslove 32 najbolj uporabljenih strani v lasten pomnilnik in s tem poveča hitrost prevedbe virtualnih naslovov v fizične.

2.1.2 Iskanje poteka programa

Potek programa rekonstruiramo tako, da ugotovimo zaporedje ukazov, ki se izvedejo pred nekim breakpoint naslovom. Naslov vsakega nesekvenčnega ukaza se shrani v posoben register, v števec pa se shrani število sekvenčnih ukazov, ki se izvedejo preden se pojavi naslednji nesekvenčni ukaz. MMU ima dva registra naslovov nesekvenčnih ukazov in dva števca naslovov sekvenčnih ukazov. Pri vsakem naslednjem nesekvenčnem ukazu se naslov prejšnjega premakne v drugi register, ravno tako se tudi število sekvenčnih ukazov med njima premakne iz prvega števca v drugi. Pri prehodu skozi breakpoint naslov imamo v prvem registru naslov zadnjega nesekvenčnega ukaza, v drugem registru naslov predzadnjega, v prvem števcu število sekvenčnih ukazov med breakpoint naslovom in zadnjim nesekvenčnim ukazom in v drugem števcu število sekvenčnih ukazov med obema nesekvenčnima ukazoma. MMU ima možnost aktiviranja dveh breakpoint naslovov, ki sta lahko virtualna ali fizična.

2.1.3 Zaščita podatkov v pomnilniku

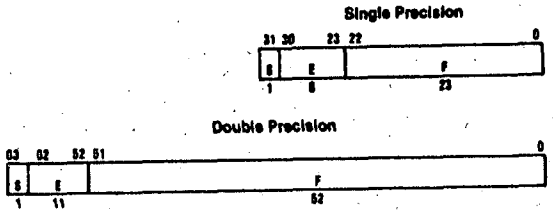
MMU omogoča tudi zaščito pred neželjenimi posegi v pomnilnik. Zaščita obsega posamezne zapise dolžine ene strani (512 bytov). Imamo štiri nivoje zaščite. Različen dostop je omogočen tudi v odvisnosti od tega ali je procesor v nadzornem ali uporabniškem načinu delovanja. To odvisnost nam kaže naslednja tabela.

		NIVO ZAŠČITE 0	1	2	3
NAČIN	uporabniški	ni dostopa	ni dostopa	samo branje	popoln dostop
	nadzorni	samo branje	popoln dostop	popoln dostop	popoln dostop

V primeru, da hoče procesor prekoračiti način dovoljenega dostopa podatkov iz pomnilnika, enota MMU takoj abortira tekoči ukaz s tem, da postavi ustrezni procesorjev vhod v aktivno stanje.

2.2 Računska enota FPU (floating point unit)

Ta enota nam omogoča izvajanje osnovnih štirih računskih operacij z realnimi števili, in sicer z navadno preciznostjo (zapis dolžine 32 bitov) ali z dvojno preciznostjo (zapis dolžine 64 bitov). Način obeh zapisov realnega števila vidimo na sliki 13.



Slika 13: zapis realnega števila v enojni in dvojni preciznosti

v drugi National Semiconductor. Ki je izdelala to družino, pravijo, da je v razvoju nova računska enota, ki bo poleg nastetega zmogla še ostale računske operacije (potenciranje, izračun trigonometričnih funkcij, logaritma, eksponentne funkcije in drugih).

3. REFERENCE

1. National Semiconductor Corporation: NS 16000 databook; 1983

2. National Semiconductor Corporation: The NS 16000 microprocessor family - Tehnical seminar.

COMPUTING

ESI was developing the microcomputer-based expert system invented by Donald Michie

Michie expert system builder goes bust

By Jenny Mill

Export Software International (ESI), the company set up to market Professor Donald Michie's Expert-Ease package, has stopped trading and suspended its 23 employees without pay.

The company's directors applied for voluntary liquidation last week after its bank account was frozen at the beginning of September. A provisional liquidator has been appointed to protect the assets of ESI until an official liquidator is appointed in the next two weeks.

The company was developing the microcomputer-based expert system invented by Michie at Edinburgh University.

Its failure was the result of 'a series of small problems over the last three months which led to the additional funding it had been expecting becoming unavailable,' according to an official of liquidator Arthur Young McClelland Moore.

ESI was hoping for a cash injection of between £150,000 to £250,000 from its existing investors, the Scottish Development Agency (SDA), Murray Technology and Hambros Advanced Technology. Some of the investors decided not to go ahead with the funding and the company was unable to continue trading.

The company's managing director and founder, Sandie Blackie, is in California, where he was setting up a US office for ESI, and according to sources close to the company he is 'left stranded'.

The marketing rights to Expert-Ease are owned by the SDA.

Boffin decries expert systems

by John Riley

A HIGH proportion of commercial companies looking for expert systems are wasting their time. That is the view of Donald Michie, father of British artificial intelligence research and the founder and head of the new Turing Institute, which aims to develop commercially viable expert systems.

Speaking to the SPL International fifth generation computing conference in London this week, he said: "The problem is, how is it after 10 years of expert systems, that very few – less than 10 – are actually doing a profitable job. The majority of commercial organisations are energetically and expensively sorting through a profusion of discarded academic prototypes.

"Everybody agrees that the problem is transplanting human know-how in to the machine. That is hard if the expert doesn't have articulate access to his own rules.

"Two years' intensive study have convinced me that if the field is defined, as it is in the US and Japan, as getting experts to tell you their rules and putting them in the machine, it is time to quit.

"At the Turing Institute we reconstruct an expert's rules through examples. The expert provides examples as if he were teaching an apprentice, and the machine infers the rules."

The Turing Institute, set up last year by Michie, completes its move from Edinburgh to Glasgow this week. It now has a dozen or so corporate members, the latest being the Harlow Laboratory of ITT Europe, and new US affiliates Westinghouse, the Radian Corporation and the US Army Corps of Engineers.

Fifth generation programs in the US, Japan and Europe will be run on the back of the Inmos Transputer chips. That is the view of Phil Treleaven, head of computer architecture at Reading University, who has spent several months with the Japanese project.

Speaking at the conference, he said: "A lot of projects in the US, Europe and Japan plan to use Inmos chips for their fifth generation programmes. There is no alternative solution."

Japan's parallel inference machine currently under development, which will merge the inference machine and knowledge-based machine and which will have the power of 1,000 microcomputers, he said, will probably use the chips.

Treleaven, who has also visited South Korea recently, reported that the country is formulating a fifth generation plan to be based in the Korean Advanced Institute of Science and Technology. It will probably take the Alvey programme as its main model.

COMPUTER WEEKLY

RAČUNSKA GEOMETRIJA

MIRO GERM

ISKRA DELTA, LJUBLJANA

UDK: 681.3:514.1

Članek govori o zahtevah in načinu podajanja oblike ter poda pregled metod. V mehaniki je oblika - geometrija osnova tako analize kot izdelave z numerično vodenimi stroji. Računska geometrija je področje uporabne matematike, ki se ukvarja s problemi oblike in je teoretična osnova CAD/CAD programov.

Computational geometry

The article describes demands and ways of defining the shape and gives an overview of the methods. In mecnanical engineering is the shape (geometry) fundamental as well in analysis as in NC production. Computational geometry is a field of applied mathematics which deals with problems of defining the shape and is theoretical background of CAD/CAM programs.

UVOD

Računalniška strojna, programska ter grafična oprema osvobodijo uporabnika od omejitev risalne deske in pospešijo celoten proces oblikovanja ali konstruiranja. Uporabnik lahko enostavno opisuje ploskve in telesa iz enostavnejših elementov, kot so točke in krivulje, dobi nov model s popravljanjem v bazi shranjenega podobnega modela itd. Uspešno uporabljanje tega orodja zahteva poznavanje matematičnih principov podajanja krivulj in ploskev brez poglobljenega znanja računalništva. Konstrukter ali oblikovalec mora poznati le program, ki mu je orodje pri njegovem delu, in pa seveda svoje delo, ne pa tudi detaljno delovanje računalnika.

Začetnik tega področja je bil v šestdesetih letih S.A.Coons. Nove matematične metode so poimenovali računska (numerična) geometrija za razliko od opisne geometrije, katero so uporabjali prej npr. pri izdelavi letal. Algoritmi računske geometrije omogočajo enostavno in hitro podajanje oblike, kar pomeni pocenitev proizvodnje.

Poleg matematičnih osnov, potrebujemo še nadomestek za risalno desko, ki takoj odgovarja ukazom. To funkcijo opravlja grafični terminal, kjer se po spreminjanju vhodnih podatkov, npr.s premikanjem peresnega držala na tablici, ustrezna krivulja ali ploskev brez večjega časovnega zaostanka narise na zaslon.

V zadnjem desetletju so geometrijsko modeliranje, interaktivna računalniška grafika in proizvodja z numerično vodenimi stroji ali roboti zelo napredovali. Vse to omogoča proizvodnjo izdelkov zapletenih oblik. Medtem ko je včasih težko podati obliko telesa na risalni deski, je računalniški opis dokaj enostaven in primeren. Prava prednost računalniskega modela je računanje določenih lastnosti telesa, preden je ta fizično proizveden (npr. trdnostna analiza, analiza prenosa toplote, analiza nihanja itd.), in avtomatična izdelava programov za NC stroje ter vizualna kontrola poti orodij. Čas med začetno idejo in koncem - izdelkom je občutno skrajšan, ravno tako je izboljšana natančnost izdelave. Npr. natančna izdelava lopatice turbine z numerično vodenim strojem zahteva vec 10 000 interpolacijskih točk. Verjetno je iluzorno pričakovati, da bi tehnolog vse te točke v 3 dimenzijah podal ročno.

1. Način podajanja oblike

Algoritmi računske geometrije so osnova CAD/CAM programov. Uspešnost uporabe teh programov pa je odvisna od razumevanja matematičnih osnov algoritmov. "Computer Aided Desisn" ali CAD se ukvarja le z obliko ali geometrijo telesa, ki je osnova tehničnih izračunov.

Konstrukter uporablja računalnik za enostavno podajanje oblike z vnosom točk in numeričnih parametrov, kot so vrednost odvodov, zvitosti, ukrivljenosti, ki so rezultat izračuna ali meritev in ki vplivajo na obliko telesa. Tako podano obliko telesa lahko uporablja tudi v kasnejših fazah dela, pri analizi obnašanja modela, proizvodnji dokumentacije in izračunu poti orodij za numerično vodene stroje.

Druga skupina uporabnikov so oblikovalci. Te zanima oblika telesa predvsem iz estetskih in vizualnih kriterijev. Od konstrukterja se razlikujejo po načinu, kako obravnavajo obliko. Manj se zanašajo na navajanje numeričnih parametrov, zanima jih le vpliv teh količin na obliko. Njihove kriterije je težko matematično

opisati. Npr. avtomobilskemu stilistu, oblikovalcu pohištva, igrač, grafičnemu umetniku, vsem, ki oblikujejo izdelke masovne proizvodnje, pomeni uporaba računalnika in matematičnega modela v začetni fazi predvsem pocenitev produkcije, iskanje večih alternativ v enakem časovnem obdobju, vizualizacijo bodočega izdelka iz različni kotov, v različnih barvah itd.

2. Zahteve pri podajanju oblike

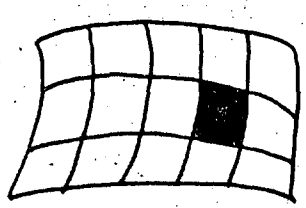
Matematični model krivulj ali ploskev splošnih oblik mora zadoščati večkrat nasprotnim željam. Začetna oblika mora biti opisana na najenostavnejši način z najmanjšim številom karakterističnih podatkov. Oblika, generirana s CAD sistemom, mora biti čimbolj je mogoče blizu zamišljeni obliki. Grafični sistem mora omogočati enostavno določitev prave oblike in nadaljne spremembe. Podajanje sprememb naj bo lokalno, da ni potrebno pri spreminjanju izračunati celotno obliko še enkrat.

Enačbe krivulj in ploskev morajo zadostiti določenim matematičnim zahtevam kot zveznosti, v večini primerov tudi zveznosti odvoda in često zveznosti ukrivljenosti. Običajno ni potrebno, da bi uporabnik sistema podajal vrednost odvoda ali ukrivljenosti numerično, kar je zelo težko, vendar mora program omogočati tudi to možnost.

Tak računalniški program mora biti visoko interaktiven, upostevati mora zadnje dosežke o komunikaciji med računalnikom in človekom. Model mora omogočati matematično jasnost definicije in izračun vseh geometrijskih lastnosti , ki določajo obliko. Še več, detaljna grafična dokumentacija in trakovi za NC stroje morajo biti generirani hitro in enostavno.

Večina CAD/CAM programov uporablja parametrične kubične krivulje. Parametrične metode imajo občutne prednosti pred ostalimi metodami. Z matematičnega in računalniškega stališča so enostavne tako v dveh kot v treh dimenzijah, niso omejene na enolične funkcije, izognemo se problemom neskončnega prevoja, so invariantne za linearne transformacije koordinatnega sistema itd.

Programi uporabljajo največkrat kubične polinome. Polinomi višjih stopenj imajo lahko nezaželjena nihanja in potrebujejo obilo parametrov za definicijo. Na drugi strani pa kompleksne oblike potrebujejo določene prostosti za opis. Parametrične kubične krivulje, kar se je pokazalo v številnih CAD/CAM sistemih, so dober kompromis med kompleksnostjo in enostavnostjo. Npr. odsekoma kubičen krivuljni segment ima štiri prostostne stopnje, tako krajevni in tangentni vektorji enolično določajo obliko. Prvi in drugi odvodi so zvezni, kar zadošča večini uporab v praksi.



Definiranje krivulj in ploskev odsekoma s segmenti ali krpami ima prednost lokalnosti, pri vsakem nadaljnjem spreminjanju je potrebno popraviti le lokalne parametre. Vendar začetna določitev prevojev, predvsem velikosti tangentnih vektorjev, povzroča resne težave predvsem pri ploskvah. Te pomanjkljivosti so povzročile uvedbo sestavljene oblike. Uporabnik podaja le nekaj točk za interpolacijo in nekaj robnih pogojev. Oblika, ki jo dobimo s pomočjo algoritma, ima avtomatično zvezen prevoj ali zvezen prevoj in ukrivljenost. Pri kubičnih krivuljah to pomeni izračun tangentnih vektorjev. Pri ploskvah pa je potreben zaradi enolične definicije tudi izračun vektorjev zvitja. Točke in robni pogoji določajo neskončno množico zveznih krivulj (ploskev). Predpisani pogoji zveznosti zmanjšajo število možnih oblik. Tangetni postopek sestavljanja izbere en element iz te množice, glede na računske ali uporabnikove zahteve. S postavljanjem točk se oblike razlikujejo ena od druge le v tangentnih vektorjih segmentov (krp).

Algoritmi so številni, lahko so lokalni ali globalni, ena metoda zahteva zvezno ukrivljenost, druge pa imajo spet drugačne zahteve. Nekateri algoritmi izhajajo iz fizikalnih, drugi iz geometrijskih lastnosti. Najbolj razširjene in sprejete metode za generacijo krivulj in ploskev so Fergusonova, Coonsova, Bezierova, ulomljene kubične krivulje, zlepki ter bazni in beta zlepki. Ne obstaja univerzalna metoda; različne zahteve uporabe zahtevajo različne algoritme.

Osnovni princip, ki povezuje te metode, so tangentni parametri, ki so določeni direktno ali izračunani na enostaven način. V primeru ploskev lahko določimo tangentne parametre s tangentnimi vektorji v obeh parametričnih smereh in z vektorji zvitosti.

Analiza problema podajanja oblike pokaže, da se ne razlikujejo samo algoritmi, ampak sta tudi vrednotenje in sprejemljivost dobljene oblike precej različna. Začetna oblika je v večini primerih blizu željene, vendar so včasih potrebni tudi popravki. Vzrok zato so predvsem estetski kriteriji, ki jih ne moremo izraziti z matematičnim orodjem. Estetske in funkcionalne zahteve so stvar intuicije, izurjenosti in izkušenj uporabnika, kar zahteva visoko interaktivnost takih programov.

3. Pregled metod za podajanje ploskev

Tip funkcije za opis ploskev je lahko :

Eksplicitni

$$z = f(x,y) \qquad (x,y) \in D$$

Implicitni

$$F(x,y,z) = 0 \qquad (x,y) \in D$$

Parametrični

$$\begin{aligned} x &= x(u,v) \\ y &= y(u,v) \\ z &= z(u,v) \end{aligned} \qquad (u,v) \in D$$

kjer sta u in v neodvisni spremenljivki ali parametra.

Krožnica

Implicitni opis

$$f(x,y) = x^2 + y^2 - 1 = 0$$

Parametrični opis

$$P(t) = (x(t), y(t)) = (\cos t, \sin t) \qquad 0 \leq t < 2\pi$$

V računalniški grafiki prevladujejo parametrični opisi, katerih predposti so naslednje :

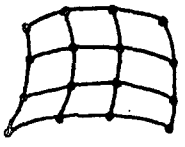
- oblika telesa je neodvisna od koordinatnega sistema, zato naj bi bila taka tudi matematična predstavitev. Parametrične metode zadoščajo temu pogoju po definiciji.

- pri parametričnih metodah ni težav pri neenoličnih funkcijah. Koordinate točke na ploskvi so izračunane neodvisno kot funkcije parametrov u in v. Vsak par (u_i, v_j) določa natančno eno točko.

- uporabnik lahko definira del ploskve enostavno z izbiro območja za u in v $(u_{min} < a < u < b < u_{max}, v_{min} < c < v < d < v_{max})$.

Če se oblika ploskve ne da izraziti analitično kot npr. sfera, valj itd., moramo uporabiti konstrukcijski način. To pomeni, da moramo funkcijo dveh spremenljivk u in v sestaviti iz enostavnejših podatkov npr. iz točk, tangentnih vektorjev (konstante) ali krivulj (funkcije ene spremenljivke). Obstajajo tri osnovni načini, s katerimi definiramo ploskev iz zgornjih podatkov :

- ploskve, dobljene s tenzorskim produktom

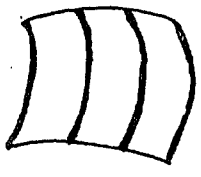


$$Q(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} P(u_i, v_j)\, U_{i,m}(u)\, V_{j,n}(v)$$

kjer vektor $P(u,v) = (X(u,v), Y(u,v), Z(u,v))$ predstavlja znane podatke in vektor $Q(u,v)$ je ustrezna ploskev izračunana iz teh podatkov.

Diskretni podatki $P(u_i, v_j)$, ki so točke ali tangentni vektorji, so interpolirani z uporabo uteznih funkcij ali interpolant $U_{i,m}(u)$, $V_{j,n}(v)$. Interpolante so lahko kubični zlepki, bazni zlepki, beta zlepki, Bezierove krivulje, racionalne krivulje itd. Oblika ploskve je odvisna od podatkov in izbire interpolant.

- podprte ploskve ('lofting')



Pred pojavom računalnikov so velike objekte, kot so ladje ali letala, oblikovali tako, da so definirali vzporedne preseke na vzdolžnih lokacijah. Ko so jih določili, so tvorili vzdolžne krivulje, ki so jih povezovale v eno samo tridimenzionalno obliko. Običajno so jih delali v naravnem merilu. Prostor v ladjedelnicah, kjer so to delali, se je imenoval 'loft', zato so imenovali ta postopek povezave 'lofting'. To je bila bolj umetnost kot znanost, ker ni bilo (se danes je ni) količinske mere, s katero bi merili zadovoljivost oblike. Odločitev je individualna.

Za definiranje ploskev uporabimo eno družino parametričnih krivulj :

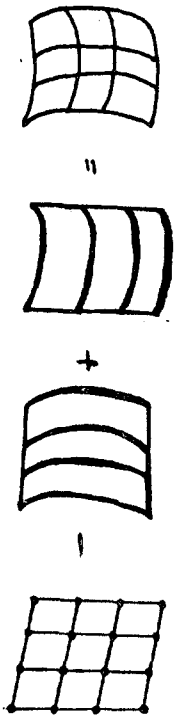
$$Q(u, v) = \sum_{i=0}^{m} P(u_i, v)\, U_{i,m}(u)$$

ali

$$Q(u, v) = \sum_{j=0}^{n} P(u, v_j)\, V_{j,n}(v)$$

Ta metoda uporablja več informacij za definiranje ploskve, krivulje namesto točk, zato ima tudi oblikovalec ali konstrukter več možnosti kontrole oblike ploskve, vendar pri večji kompleksnosti računanja kot tenzorski produkt.

- transfinitne metode

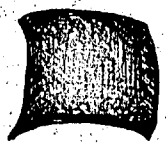


$$Q(u,v) = \sum_{i=0}^{m} P(u_i, v)\, U_{i,m}(u) + \sum_{j=0}^{n} P(u, v_j)\, V_{j,n}(v) - \sum_{i=0}^{m} \sum_{j=0}^{n} P(u_i, v_j)\, U_{i,m}(u)\, V_{j,n}(v)$$

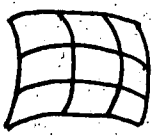
Transfinitne metode imajo večjo kontrolo nad obliko kot podprte ploskve ali ploskve tenzorskega produkta, vedar pri še večji obsežnosti računanja. Za opis ploskve lahko podamo več podatkov, ki natančneje določajo želeno obliko. Rezultat algoritmov so zvezne ploskve ali pa tudi enkrat ali dvakrat zvezno odvedljive ploskve. Pri prvih je potrebno podati le robne krivulje, pri drugih pa tudi vrednosti odvodov ter mešanih odvodov (vektorje zvitosti). Za ploskve, dobljene s tenzorskim produktom, smo kot podatke uporabili le točke, pri podprtih ploskvah pa smo podali krivulje, ki so podpirale ploskve. Ideja transfinitnih metod je tvorba Boolove vsote podprtih ploskev v obeh parametričnih smereh. Ime so dobile po 'transfinitni' interpolaciji, saj lahko interpoliramo vsako točko podanih krivulj.

Kriteriji presleda metod so bili hitrost odziva in enostavnost matematične predstavitve, zato je bila na prvem mestu metoda tenzorskega produkta. Osnovni podatki za to predstavitev so točke, katerih podajanje je seveda enostavnejše kot podajanje krivulj in tudi računsko je metoda hitrejša, kar je pomembno pri interaktivnem delu. Ostali dve metodi pa imata več kontrole nad obliko in natančnejšo aproksimacijo.

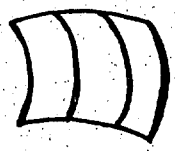
Popolni opis



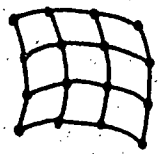
Transfinitni opis



Podprta ploskev



Tenzorski produkt



4. Zaključek

Poznavanje računske geometrije je pomembno ne samo za izdelovalce CAD/CAM programske opreme, ampak tudi za uporabnike, da bodo za reševanje svojih problemov nabavili ustrezne programe.

LITERATURA :

((1.)) I.D.Faux M.J.Pratt: Computational Geometry for Design and Manufacture, John Wiley & Sons, 1979.

((2.)) C.de Boor: A Practical Guide to Splines, Springer-Verlag, 1978.

((3.)) R.E.Barnhill W.Boehm (eds): Surfaces in Computer Aided Geometric Design, North-Holland, 1983.

((4.)) R.E.Barnhill R.F.Riesenfeld (eds): Computer Aided Geometric Design, Academic Press, 1974.

((5.)) W.Bohm G.Farin : Surface Design, Eurographics, 1983.

((6.)) W.J.Gordon : Free - Form Surface Interpolation through Curve Networks, General Motors Research Laboratories, 1969.

((7.)) B.Turk : Racunarska grafika, Skolska knjiga - Zagreb.

((8.)) R.E.Barnhill : Representation and Approximation of Surfaces , Siggraph 84.

((9.)) W.J.Gordon : The Mathematical Basis of Coons and Related Techniques for Free - Form Surface Definition, Siggraph 84.

((10.)) R.H.Bartels J.C.Beatty B.A.Barsky : An Introduction to the Use of Splines in Computer Graphics, Siggraph 84.

((11.)) S.Divjak: Računalniška grafika, Fakulteta za elektrotehniko, 1980.

INFORMATICA 1/85

STRUKTURIRANO PROGRAMIRANJE U ASEMBLERU

HRVOJE NEŽIĆ

ELEKTROTEHNIČKI INSTITUT „RADE KONČAR", ZAGREB

UDK: 519.682.8

U radu je opisan način na koji se i u jeziku niskog nivoa - asembleru, može programirati strukturirano, tj. bez korištenja JUMP naredbi u izvornom tekstu. Za to je dovoljno imati makroasembler koji ima znakovne varijable i mogućnost supstitucije teksta. Strukturiranje programa izvodi se pomoću makroinstrukcija koje su slične instrukcijama grananja i ponavljanja u ADI i PASCALU.

STRUCTURED PROGRAMMING IN ASSEMBLY LANGUAGE

The paper describes the way in which one can create structured programs, i. e. programs without JUMP instructions, even in low level language - assembly language. It suffices to have a macroassembler which includes character variables and text supstitution possibility. Structuring of the program is performed by macroinstructions which resembles branch and repetition instructions of ADA and PASCAL.

1. UVOD

Prednosti strukturiranog programiranja u odnosu na nestrukturirano, u kojem se gotovo sva grananja i ponavljanja izvode korištenjem GOTO naredbi, su očite. U nestrukturiranim programima obično postoji mnogo labela i mnogo medjusobno isprepletenih instrukcija skokova na te labele. Programirati na taj način vrlo je zamorno, jer je potrebno pratiti mnoge puteve grananja programa koji su medjusobno isprepleteni; programi su nečitki, nepregledni i nerazumljivi, kako za onoga tko ih je pisao, tako i za druge, što znači da ih je potrebno vrlo opsežno komentirati i dokumentirati. Osim toga, greške se vrlo lako pojavljuju i teško otkrivaju. Nasuprot tome, instrukcije grananja i ponavljanja u strukturiranom programiranju omogućuju pisanje programa bez korištenja GOTO naredbi. Time je izbjegnuta isprepletenost putova grananja u izvornim programima i postignuta čitljivost i preglednost, te lakoća programiranja i razumljivost programa. Takve konstrukcije grananja i ponavljanja slične su jezičnim konstrukcijama koje upotrebljavamo u svakodnevnom govoru, što znači da su bliske i prihvatljive čovjeku.
Nestrukturirani program obično ima oblik niza instrukcija u kojem je vrlo teško uočiti pojedine dijelove koji predstavljaju logičke cjeline za sebe, dok su strukturirani programi podijeljeni na dijelove koje već na prvi pogled uočavamo, tako da odmah vidimo osnovnu strukturu programa (npr. dijelove koji se ponavljaju uvjetno, dijelove koji se ponavljaju bezuvjetno itd).

Primjeri viših jezika u kojima uglavnom nije moguće strukturirano programiranje su BASIC i FORTRAN, dok su PASCAL, a pogotovo ADA, primjeri jezika koji vrlo dobro podržavaju strukturirano programiranje.

Ipak, u većini dijalekata BASIC-a i FORTRAN-a moguća je bar minimalna strukturiranost, jer postoji osnovna IF ... THEN konstrukcija, kao i FOR, odnosno DO petlja.

Nasuprot tome, u asemblerskim jezicima strukturirano programiranje gotovo uopće nije moguće - obično postoje samo instrukcije bezuvjetnog grananja (JP adresa) koje izvršavaju skok na odredjenu adresu i instrukcije uvjetnog grananja (JP uvjet, adresa) koje izvršavaju skok na specificiranu adresu samo ako je ispunjen odredjen uvjet, dok se inače izvodjenje programa nastavlja sa slijedećom instrukcijom (instrukcije uvjetnog grananja u asembleru ekvivalentne su instrukcijama

if uvjet then goto adresa u višim jezicima).

Danas postoji prirodna težnja da se viši, odnosno srednji jezici koriste za razvoj programske podrške u što više područja, pa tako i u onim za koje se je donedavno upotrebljavao samo asembler.

Aplikacioni programi pišu se gotovo isključivo u višim jezicima. Od pojave PASCALA i jezika srednjeg nivoa kao što su C i FORTH, viši i srednji jezici sve više prodiru i u područje sistemskih programa kao i u procesne primjene.
Ipak, u tim područjima su viši jezici ponekad nedovoljno efikasni, pogotovo u procesnim primjenama u kojima se traži velika brzina rada ili zgusnutost koda.
U takvim slučajevima je overhead koji unose viši jezici neprihvatljiv i nužno je korištenje asemblera, a slično je i u sistemskim programima, gdje se obično

barem dio koda mora napisati u asembleru.

Ukratko, upotreba asemblera još uvijek je dosta rasprostranjena unatoč tendenciji za smanjivanjem. Kao što je već ukazano, programiranje u asembleru je gotovo posve nestrukturirano. Velikim dijelom zbog toga razvoj takvih programa je zamoran i spor. Makroasembleri znatno olakšavaju programiranje, ali sami po sebi obično ne pružaju mogućnosti strukturiranog programiranja.

2. OSNOVNE IDEJE STRUKTURIRANOG PROGRAMIRANJA U MAKROASEMBLERU

Ovaj rad opisuje način na koji se strukturirano programiranje u asembleru može postići upotrebom makroasemblera koji podržava znakovne varijable i ima mogućnost supstitucije teksta. Ta svojstva nisu baš uobičajena za makroasemblere, no takvi makroasembleri ipak postoje. Jedan primjer je makroasembler za mikroprocesor Z8Ø koji se nalazi u sklopu sistemske programske podrške Tektronixovog višekorisničkog razvojnog sistema za mikroprocesore - sistema 85ØØ.

Ovdje ćemo ukratko opisati njegova svojstva koja se mogu iskoristiti za strukturirano programiranje. Najvažnije svojstvo je mogućnost dinamičke supstitucije teksta bilo gdje unutar bilo koje izvorne linije za vrijeme asembliranja. Supstitucija teksta se vrši posredstvom znakovnih varijabli, tako da se ime znakovne varijable stavi pod dvostruke navodne znakove (npr. "IME"). Kada asembler naidje na takvu pojavu u tekstu, on je zamjenjuje sa tekstom koji trenutno sadrži dotična znakovna varijabla. Za vrijeme obrade neke linije teksta asembler najprije provodi sve naznačene supstitucije teksta, a tek onda ispituje značenje tako dobivenog teksta.

Dinamička supstitucija teksta omogućuje da se i simboličke adrese (labele) kao i uvjetni i bezuvjetni skokovi na te labele generiraju dinamički, tj. automatski, a da se uopće ne moraju nalaziti u izvornom tekstu programa. Razrada ove osnovne ideje pruža mogućnost strukturiranog programiranja u asembleru. Instrukcije strukturiranog programiranja u izvornom programu se izvode kao makroinstrukcije, a pripadne makroekspanzije provode generiranje svih potrebnih labela i skokova. Makroekspanzije generiraju nove linije u tekstu. Neke od takvih linija sadrže samo labelu, što asembler dozvoljava.

Da bi bilo moguće umetanje jedne konstrukcije grananja ili ponavljanja unutar druge i to ne samo na dva nego na više nivoa, potrebno je voditi neke numeričke varijable (npr. broj trenutnog nivoa). Asembler omogućava pretvorbu numeričkih u znakovne vrijednosti, tako da je moguće znakovne ekvivalente takvih varijabli ugraditi u generirane labele.

Na kraju spomenimo svojstvo asemblera da bilo koji makropoziv može imati proizvoljan broj argumenata, jer pripadna makrodefinicija nikad ne sadrži specifikaciju broja argumenata. Ovo svojstvo koristi se kod jedne makroinstrukcije strukturiranja.

3. PREGLED INSTRUKCIJA STRUKTURIRANOG PROGRAMIRANJA

Ovdje odabrane makroinstrukcije strukturiranog programiranja slične su odgovarajućim instrukcijama u ADI odnosno PASCALU. Da bi se te instrukcije razlikovale od asemblerskih direktiva uvjetnog i ponavljanog asembliranja koje su također slične, one imaju na kraju tri znaka " _ " . To ujedno pridonosi preglednosti programa, jer se makroinstrukcije strukturiranja lako razlikuju od asemblerskih instrukcija.

3.1 Struktura IF

Osnovna struktura grananja ima ovakav oblik:

```
IF___   uvjet
  .
  .
  .
ELSETEST___
  .
  .
  .
WHEN___ uvjet
  .
  .
  .
  .
ELSETEST___
  .
  .
  .
WHEN___ uvjet
  .
  .
  .
  .
ELSE___
  .
  .
  .
  .
ENDIF___
```

i slična je konstrukciji

```
if ... then
    :
    :
elsif ... then
  .
  .
  :
elsif ... then
```

```
        else
            .
            .
            .
        ond if
```

u ADI, pri čemu makroinstrukcije

```
        ELSETEST _ _ _
            .
            .
        WHEN _ _ _ uvjet
```

odgovaraju instrukciji elsif ..... then.
Instrukciju IF _ _ _ može slijediti proizvoljan broj parova ELSETEST _ _ _, WHEN _ _ _ (ili nijedan), koje može (ali ne mora) slijediti instrukcija ELSE _ _ _.
Argument "uvjet" u makroinstrukcijama IF _ _ _ i WHEN _ _ _ može poprimiti slijedeće vrijednosti:

Z, NZ, C, NC, P, M, EQ, NE, LT, GE.

Mnemonici Z, NZ, C, NC, P i M jednaki su mnemonicima uvjeta u instrukcijama mikroprocesora Z8Ø i označavaju stanja pojedinih bitova u status registru, dok su ostali ekvivalentni nekima od njih. Tako su ekvivalentni EQ (equal) i Z, NE (not equal) i NZ, LT (less then) i C, GE (greater or equal) i NC.
Dio koda iza IF _ _ _ instrukcije izvoditi će se samo ako je ispunjen specificirani uvjet, tj. ako se odredjeni bit status registra nalazi u odredjenom stanju.
Makroinstrukcija IF _ _ _ generira uvjetni skok na dio koda koji počinje sa ELSETEST _ _ _ (odnosno ELSE _ _ _ odnosno ENDIF _ _ _) instrukcijom. Na taj dio koda se skače ako uvjet u IF _ _ _ instrukciji nije ispunjen.
Ako se u programu želi ispitati neki uvjet i ovisno o tome nešto uraditi, potrebno je najprije izvršiti neophodne instrukcije CP, OR, AND, XOR, BIT itd, koje utječu na status registar mikroprocesora, a iza njih postavlja se instrukcija IF _ _ _ koja samo testira odrodjeni bit status registra.
Ako uvjet u IF _ _ _ instrukciji nije ispunjen izvodi se dio programa izmedju ELSETEST _ _ _ i WHEN _ _ _ instrukcija (naravno, samo ako one postoje).
Taj dio koda obavlja testiranje nekog uvjeta, odnosno obnavljanje status registra, a WHEN _ _ _ instrukcija testira odredjeni bit status registra - ako je uvjet ispunjen izvodi se dio koda koji slijedi.

3.2 Struktura CASE

Druga struktura grananja takodjer ima oblik sličan analognoj strukturi u jeziku ADA:

```
CASE _ _ _
    .
    .
    .
    .
WHEN _ _ _ vrijednost 1, vrijednost 2, ...
    .
    .
    .
    .
    .
WHEN _ _ _ vrijednost 1, vrijednost 2, ...
    .
    .
    .
    .
WHEN _ _ _ OTHERS
    .
    .
    .
    .
ENDCASE _ _ _
```

Instrukciju CASE _ _ _ može slijediti proizvoljan broj WHEN _ _ _ instrukcija, pri čemu se kao posljednja od njih može pojaviti instrukcija

```
WHEN _ _ _ OTHERS.
```

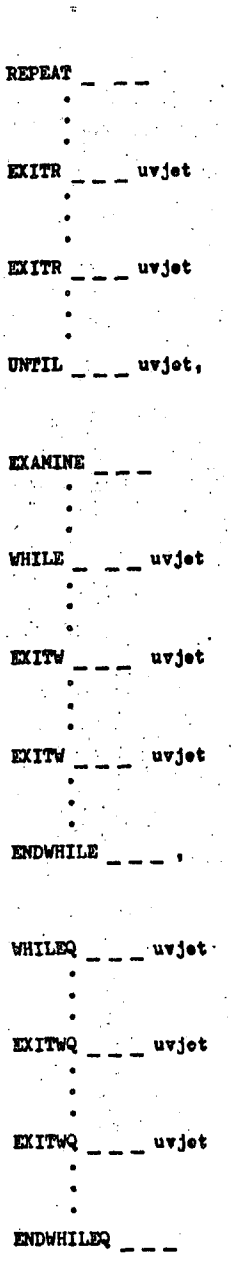
U svim ostalim WHEN _ _ _ naredbama može se nalaziti proizvoljan broj argumenata (ali barem jedan).

Struktura CASE _ _ _ provodi grananje ovisno o sadržaju A registra, sa kojim se usporedjuju vrijednosti zadane u WHEN _ _ _ naredbama. Zato se prije CASE _ _ _, ili izmedju CASE _ _ _ i prve WHEN _ _ _ instrukcije moraju nalaziti naredbe koje pune A registar sa željenom vrijednošću. Argumenti u instrukcijama WHEN _ _ _ mogu imati numeričku vrijednost koja ne prelazi 8 bita, ili znakovnu vrijednost od jednog znaka.

3.3 Strukture ponavljanja

Predvidjene su 4 strukture ponavljanja i to:

```
LOOP _ _ _
    .
    .
    .
    .
EXITL _ _ _ uvjet
    .
    .
    .
EXITL _ _ _ uvjet
    .
    .
    .
ENDLOOP _ _ _,
```

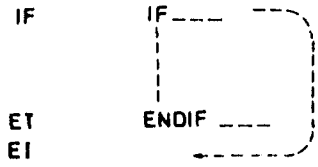


Prva od njih je struktura bezuvjetnog ponavljanja, dok je kod ostalih ponavljanje uvjetno.
U svim ovim strukturama izlaz iz petlji omogućuju pripadne EXIT naredbe, kojih unutar svake strukture može biti proizvoljan broj (ili nijedna).
Ovdje su predvidjene dvije konstrukcije koje odgovaraju uobičajenoj WHILE strukturi u višim jezicima. Kod prve od njih naredbe izmedju EXAMINE _ _ _ i WHILE _ _ _ izvršavaju ispitivanje nekog uvjeta, tj. obnavljanje statusa registra, a sama WHILE _ _ _ instrukcija generira uvjetni skok na kraj petlje. Pritom instrukcija ENDWHILE _ _ _ generira bezuvjetni skok na kod iza EXAMINE _ _ _ naredbe, što znači da se svaki put prilikom ponovnog ispitivanja uvjeta izvršavaju dvije instrukcije skoka. Da bi se to izbjeglo, a u cilju postizanja što veće efikasnosti koda, dodana je četvrta struktura (mnemonik: quick while), kod koje instrukcija WHILEQ _ _ _ generira naredbu uvjetnog skoka na kraj koja se izvodi samo pri prvom prolazu, a ENDWHILEQ _ _ _ generira uvjetni skok na početak, koji se izvodi nakon svakog prolaza. Medjutim, zbog toga se kod koji ispituje uvjet mora nalaziti i prije WHILEQ _ _ _ i prije ENDWHILEQ _ _ _ instrukcije.

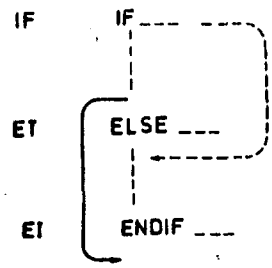
Ovime smo završili predstavljanje makroinstrukcija za strukturirano programiranje. Sada prelazimo na opis realizacije, tj. na detaljniji opis načina generiranja labela, skokova i ostalih instrukcija (u CASE _ _ _ strukturi potrebno je osim labela i skokova generirati i COMPARE instrukcije).

4. GENERIRANJE LABELA, SKOKOVA I OSTALIH INSTRUKCIJA

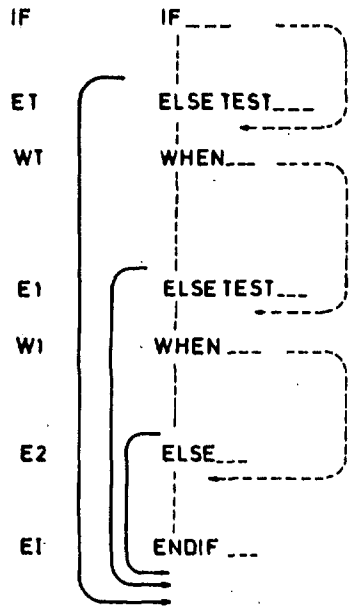
Najprije ćemo promatrati generiranje labela i skokova unutar samo jedne konstrukcije, ne uzimajući u obzir činjenicu da u programu može biti više konstrukcija istog tipa, te da se konstrukcije mogu gnijezditi jedna unutar druge. Ovdje je odabrano da automatski generirane labele mogu imati najmanje 4, a najviše 8 znakova.
Način generiranja prva dva znaka labela ilustriran je slikama 1 do 9, na kojima su prikazane sve konstrukcije (konstrukcija IF _ _ _, ENDIF _ _ _ dana je u 4 varijante), kao i prva dva znaka generiranih labela, te generirani skokovi. Bezuvjetni skokovi prikazani su punim, a uvjetni crtkanim linijama sa strelicama.
Prva dva znaka labele su potpuno ili djelomično mnemonička (npr. ET ima značenje "End then"). Ostali znakovi labele ovise, izmedju ostalog, i o nivou na kojem je dotična konstrukcija ugniježdena u izvornom programu. Sve labele koje su prikazane na slikama zapravo nije neophodno potrebno generirati jer se na njih ne skače (takve labele su npr. IF, WT, W1, X1, X2, WH), no one mogu biti dosta korisne u fazi ispitivanja programa.
Osim toga, neke instrukcije u nekim slučajevima mogu generirati dvije labele, koje se odnose na istu memorijsku adresu. Tako je npr. na slici 1 labela ET potrebna zbog generiranja skoka na nju, a labela EI (mnemonik "End if") generira se zbog boljeg pregleda pri ispitivanju programa.
Ovdje ćemo za strukturu IF _ _ _ , ENDIF _ _ _ detaljno opisati način generiranja svih labela i skokova, dok ćemo za ostale strukture samo ukratko prikazati specifičnosti.



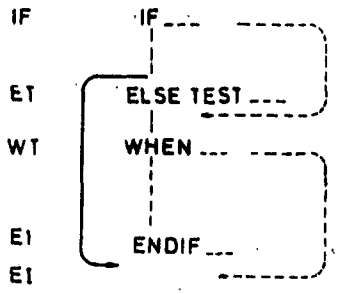
Slika 1



Slika 2



Slika 3



Slika 4

4.1 Struktura IF

Ako je uvjet u instrukciji IF _ _ _ ispunjen izvodi se dio programa koji slijedi, a ako nije tada se izvodjenje nastavlja na labeli ET (pripadnu labela ET generira prva od instrukcija ELSETEST_ _ _ , ELSE_ _ _ ili ENDIF _ _ _ na istom nivou). Zato makroinstrukcija IF _ _ _ najprije mora odrediti uvjet koji je suprotan specificiranom uvjetu (npr. suprotni uvjeti su Z i NZ, C i NC itd.) i tada generirati instrukciju slijedećeg oblika

JP suprotan uvjet, ET .

Prva od instrukcija ELSETEST _ _ _ ili ELSE _ _ _ na istom nivou (ako postoji) stvorit će slijedeće dvije linije koda:

JP EI

ET .

Prva linija provodi bezuvjetni skok na labelu EI, tj. na završetak konstrukcije, jer nakon što se izvede kod izmedju IF _ _ _ i ELSETEST _ _ _ odnosno ELSE _ _ _ naredbe, nastavak programa mora slijediti na instrukciji iza ENDIF _ _ _ naredbe. Druga ELSETEST naredba stvara labelu E1, treća labelu E2 itd., sve do labele E9. (odabrano je do parova ELSETEST _ _ _, WHEN _ _ _ može biti najviše 1Ø). Svaka WHEN _ _ _ naredba stvara labelu čije je prvo slovo W, a drugo slovo jednako je drugom slovu labele koju je stvorila pripadna (a to znači prethodna) ELSETEST _ _ _ naredba, te generira uvjetni skok (sa uvjetom suprotnim specificiranom uvjetu) na prvu slijedeću ELSETEST _ _ _ naredbu ako ona postoji.

Ako je pripadna ELSETEST _ _ _ naredba ujedno i posljednja, tada WHEN _ _ _ naredba generira skok na kod iza ELSE _ _ _, odnosno ENDIF _ _ _ naredbe.
Da bi se to ostvarilo potrebno je voditi varijablu ELST koju IF _ _ _ instrukcija inicijalizira u Ø, a svaka WHEN _ _ _ instrukcija je povećava za 1. Kada je potrebno ta varijabla može se iz numeričkog pretvoriti u znakovni oblik (npr. u varijablu TØ) i tada se odgovarajuće labele (ako je varijabla veća od Ø) generiraju na slijedeći način:

E"TØ"
W"TØ" .

Umjesto "TØ" makroasembler uvrštava u tekst trenutni sadržaj znakovne varijable TØ.
Dok prva ELSETEST _ _ _ instrukcija uvrštava linije

JP EI
ET

sve slijedeće stvaraju linije slijedećeg oblika:

JP EI

E"TØ"

Instrukcija WHEN _ _ _ najprije generira labelu W"TØ", zatim poveća ELST za 1 i obnovi TØ, te generira liniju

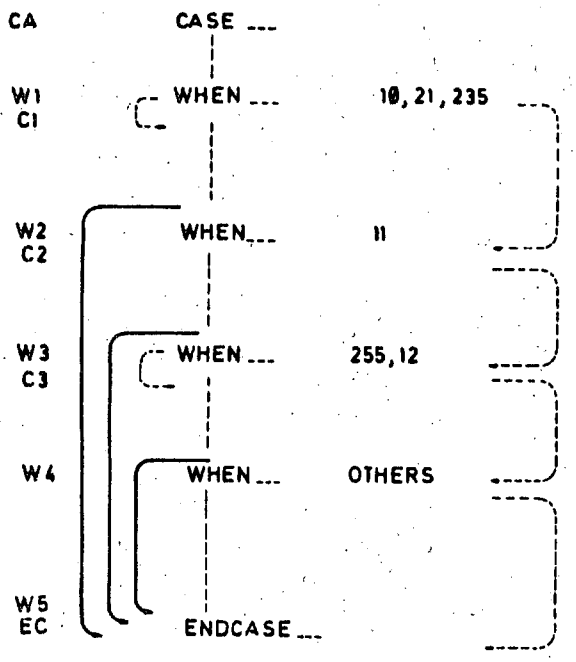
JP suprotni uvjet, E"TØ" .

Ako je ELST = Ø instrukcija ELSE _ _ _ generira labelu ET, a inače generira labelu E"TØ".

Instrukcija ENDIF _ _ _ generira labelu u ovisnosti o postojanju naredbi ELSETEST _ _ _ i ELSE _ _ _. Zato je potrebno voditi i varijablu ELS koju instrukcija IF _ _ _ postavlja u Ø, a ELSE _ _ _ u 1.

Ako je ELS = 1 tada ENDIF _ _ _ stvara samo labelu EI, ako je ELS = Ø, ELST = Ø, stvaraju se labele ET, EI, a u slučaju ELS = Ø, ELST > Ø labele E "TØ", EI.

4.2. Struktura CASE



Slika 5

Simboličke adrese i ovdje se stvaraju na način analogan onome koji je detaljno prikazan kod prethodne strukture. Za ovu strukturu je specifična upotreba COMPARE naredbi. Za svaki od svojih argumenata osim za zadnji, instrukcija WHEN _ _ _ stvara naredbe

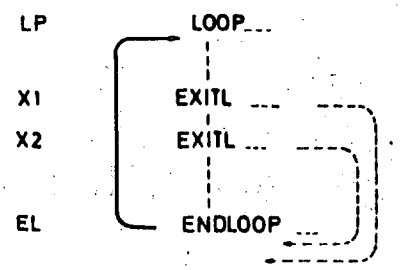
CP vrijednost
JP Z, Ci

dok za zadnji argument stvara linije

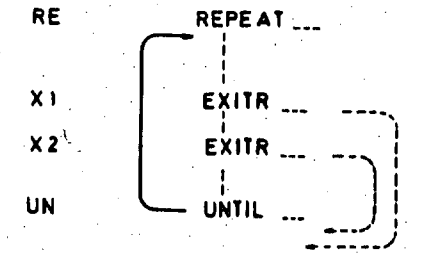
CP vrijednost
JP NZ, Wj

Ci

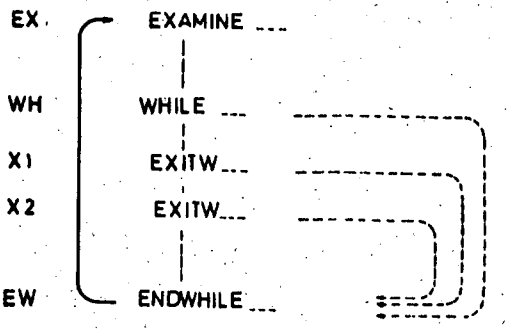
gdje je i broj WHEN _ _ _ naredbe od početka strukture, a j = i + 1 .



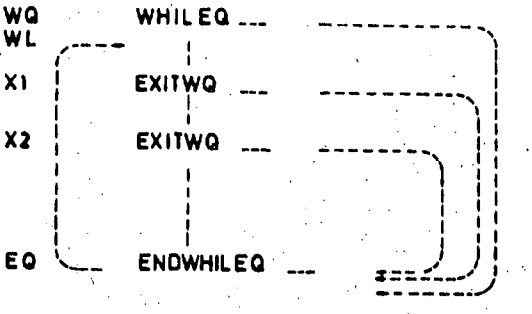
Slika 6



Slika 7



Slika 8



Slika 9

4.3. Strukture ponavljanja

Pošto se i kod ovih struktura skokovi i labele stvaraju na način analogan opisanom, ukratko ćemo opisati samo WHILEQ _ _ _ strukturu. Uvjet zadan u WHILEQ _ _ _ instrukciji ispituje se dvaput: jednom na početku, a drugi put na kraju konstrukcije. Naredba WHILEQ _ _ _ stvara linije

```
WQ          JP  supruvj,  EQ
WL
```

a naredba ENDWHILEQ - - -

```
            JP  uvj, WL
```

pri čemu se uvjet iz WHILEQ _ _ _ naredbi pamti u jednoj varijabli, da bi ga mogla upotrijebiti i naredba ENDWHILEQ _ _ _ .

5. RJEŠAVANJE PROBLEMA KORIŠTENJA VIŠE KONSTRUKCIJA ISTOG TIPA I PROBLEMA GNIJEŽĐENJA

Generiranje labela i skokova unutar samo jedne konstrukcije (koja se promatra kao cjelina za sebe, nezavisna od ostalih) što je opisano u prethodnom tekstu, u suštini je prilično jednostavno.
Medjutim, samo rješenje tog problema ne pruža mogućnost korištenja više konstrukcija istog tipa u izvornom programu, kao ni medjusobno gniježđenje konstrukcija jer u takvim slučajevima labele generirane istim makroinstrukcijama na raznim mjestima u programu moraju biti različite i prema tome takvo rješenje samo za sebe bilo bi od male koristi.
Pretpostavimo za trenutak da se ne dozvoljava gniježđenje, već da se konstrukcije u programu pojavljuju samo sekvencijalno jedna iza druge. U tom slučaju problem se može jednostavno riješiti uvodjenjem varijable koja predstavlja redni broj odredjene konstrukcije počevši od početka programa i uvrštavanjem znakovnog oblika te varijable u sve generirane labele i skokove. Na početku programa se ta varijabla postavlja u -1, a zatim je svaka instrukcija koja predstavlja početak odredjene konstrukcije povećava za 1. Sada svaka generirana labela pored već opisana prva do znaka sadrži još dva znaka - znakovni ekvivalent spomenute varijable. Npr. ako se u program najprije pojavljuje konstrukcija IF _ _ _, ENDIF _ _ _ ona će generirati labele IFØØ, ETØØ itd.

Preostaje još riješiti problem gniježđenja. Da bismo ga riješili uvodimo varijablu NIVO koja označava trenutni nivo gniježđenja u programu. Na početku programa se postavlja NIVO = -1, a nakon toga početna instrukcija svake konstrukcije povećava NIVO za 1, dok ga završna instrukcija svake konstrukcije smanjuje za 1. Prema tome, ako je trenutni nivo na nekom mjestu u programu jednak -1, to znači da su do tog mjesta sve započete konstrukcije i završene, ili da ih još uopće nije bilo.

Ekvivalent varijable NIVO također se uvrštava u generirane labele i skokove i na taj način se razlikuju labele generirane istim instrukcijama na raznim nivoima (iznimka je nivo Ø za koji je ovdje odabrano da se broj nivoa zbog preglednosti ne umeće u labelu).
Medjutim, ni sa uvodjenjem varijable NIVO nismo u potpunosti riješili problem, jer unutar svakog nivoa može postojati više raznih konstrukcija koje se nižu sekvencijalno jedna za drugom. Zbog toga za svaki nivo mora postojati varijabla koja broji konstrukcije na tom nivou.

Označimo te varijable sa BRØ, BR1, ... , BR9 (odabrano je da nivoa može biti najviše 1Ø). Zbog preglednosti odabiremo da se brojanje konstrukcija izvodi na slijedeći način. Varijabla BRØ, kako je već prije opisano, broji konstrukcije na nivou Ø od početka programa. Za razliku od toga, preostale varijable broje konstrukcije na višim nivoima samo od početka pripadne konstrukcije na nultom nivou. Završetak svake konstrukcije na nultom nivou ponovo postavlja sve varijable BR1 do BR9 u -1.

Početna makroinstrukcija svake konstrukcije povećava varijablu NIVO za 1, odredjuje njen znakovni ekvivalent NØ i obnavlja brojilo konstrukcije na trenutnom nivou naredbom

```
BR"NØ"      ASET      BR"NØ"  +  1 .
```

Labela se u općem slučaju sastoji od slijedećih polja (ispod svakog polja napisan je broj znakova):

Mnemonik	Broj konstrukcije na nivou Ø	.	Trenutni nivo	Broj konstrukcije na trenutnom nivou
2	2	1	1	2

Peti znak (točka) upotrebljava se zbog preglednosti.
Ako je NIVO = Ø tada se labela sastoji samo od prva 4 znaka, a ako je NIVO > Ø ali je broj konstrukcije na trenutnom nivou jednak Ø labela se sastoji samo od prvih 6 znakova. Takav način generiranja labela je također odabran zbog preglednosti.
Dio labele od trećeg do osmog znaka zajednički je za sve labele stvorene od instrukcija iste konstrukcije. Taj dio ćemo nazivati ekstenzijom labele odnosno ekstenzijom adrese.
Ekstenziju adrese izračunava početna instrukcija konstrukcije i sprema je u jednu varijablu (zbog mogućnosti gniježđenja konstrukcija mora postojati posebna varijabla za svaki nivo) da bi je mogli koristiti i ostali dijelovi konstrukcije.
Napomenimo na ovom mjestu da također zbog gniježđenja umjesto svake od prije opisanih varijabli ELS i ELST mora postojati po 1Ø varijabli - posebna varijabla za svaki nivo.

Varijable koje sadrže ekstenzije adresa označavamo sa AEXTØ, ... , AEXT9. Sve instrukcije strukturiranja generiraju labele i skokove tako da varijabli AEXT pridruže vrijednost ekstenzije adrese na trenutnom nivou:

```
AEXT    ASET    AEXT"NØ"
```

i zatim sve labele oblikuju na ovaj način:

```
Mnemonik "AEXT".
```

Na kraju napomenimo da prije bilo koje makroinstrukcije strukturiranja u programu mora doći instrukcija BEGIN _ _ _ koja postavlja varijable NIVO, BRØ, ... , BR9 jednakim -1.

6. ZAKLJUČAK

Opisani način strukturiranog programiranja u asembleru realiziran je na Tektronixovom sistemu 85ØØ. Strukturirano programiranje omogućuje skup od 28 makrodefinicija sa oko 3ØØ linija koda (korisnički programi direktno mogu za strukturiranje upotrebljavati 21 makroinstrukciju).

Ovakav način strukturiranog programiranja, naravno, povećava vrijeme asembliranja programa u odnosu na nestrukturirano programiranje, jer se simboličke adrese i skokovi stvaraju za vrijeme asembliranja a za to je potrebno odredjeno vrijeme (svaka makrodefinicija strukturiranja sadrži više naredbi). To je naravno jedna od mana ovog načina, no autoru se čini da su koristi mnogostruko veće, o čemu je već bilo govora u uvodu.

Realizacija pomoću makroinstrukcija, odnosno makroasemblera nije jedini način na koji se opisani model strukturiranog programiranja u asembleru može realizirati. Najprirodnija mogućnost je realizacija unutar samog asemblera, pri čemu bi instrukcije strukturiranja imale oblik asemblerskih direktiva, a moguća je i realizacija pomoću programa napisanog npr. u nekom višem jeziku, koji bi instrukcije strukturiranja u izvornom tekstu zamjenio labelama i skokovima i na taj način formirao novu datoteku čiji je sadržaj prihvatljiv za asembler.

Instrukcije strukturiranog programiranja oslobadjaju programera od zadataka stvaranja labela i skokova, pri čemu on i dalje ima potpunu slobodu u korištenju instrukcija asemblera za stvaranje efikasnih programa. Ipak, opisana realizacija ne pruža jednaku efikasnost koda kao kod direktnog programiranja u asembleru, jer se u nekim slučajevima (primjer-slika 1Ø) prijenos kontrole programa na odredjeno mjesto vrši sa više skokova, umjesto sa samo jednim skokom ili RET instrukcijom.

```
IF _ _ _
        .
        .
    IF _ _ _
        .
        .
    ENDIF _ _ _
ELSE _ _ _
        .
        .
ENDIF _ _ _
```

Slika 1Ø

Medjutim, u poboljšanoj realizaciji svi se ovi problemi mogu riješiti i na taj način generirani kod može biti jednako efikasan kao onaj proizveden direktno u asembleru.

LITERATURA

1. - : 8500 Modular MDL series assembler core users manual for B series assemblers, Tektronix, Inc. Beaverton, Oregon, 1983.

2. P. Wegner: Programming with ADA, Englewood Cliffs, N.J., Prentice Hall, 1980.

DODATAK

U dodatku rada, kao ilustracija, dan je kratki potprogram za decimalno binarnu pretvorbu 8-bitnog broja. Najprije je dan izvorni oblik, a zatim oblik sa automatski generiranim labelama i skokovima, koji otvara makroasembler.

```
DECBIN  BEGIN ___
        LD C,02
        LD HL, BIN
        LD A, (BROJ)
        LOOP___
          LD D,00
          EXAMINE ___
            CP 16H
          WHILE ___ GE
            SUB 16H
            DAA
            INC D
          ENDWHILE ___
          CP 10H
          IF ___ GE
             SUB 06
          ENDIF ___
          RRD
          LD, A,D
          OR A
        EXITL ___ Z
          DEC C
        ENDLOOP ___
        LOOP___
          DEC C
        EXITL ___ Z
          RRD
        ENDLOOP ___
        RET


        DECBIN      BEGIN ___
             LD C,02
             LD HL,BIN
             LD A,(BROJ)
             LOOP ___
        LP00
               LD D,00
               EXAMINE ___
        EX00.1
                 CP 16H
               WHILE ___ GE
        WH00.1         JP C,EW00.1
                 SUB 16H
                 DAA
                 INC D
               ENDWHILE ___
             JP EX00.1
        EW00.1
               CP 10H
               IF ___ GE
        IF00.101        JP C,ET00.101
                  SUB 06
               ENDIF ___
        ET00.101
        EI00.101
               RRD
               LD A,D
               OR A
             EXITL ___ Z
        X100 JP Z,EL00
               DEC C
             ENDLOOP ___
             JP LP00
        EL00
               LOOP ___
        LP01
                DEC C
               EXITL ___ Z
        X101 JP Z,EL01
              RRD
             ENDLOOP ___
             JP LP01
        EL01
             RET
```

KOMUNIKACIJSKO USMERJEN INFORMACIJSKI SISTEM ZA UPRAVLJANJE S PROIZVODNJO – IBM COPICS

VLADO STARIČ

IMV, NOVO MESTO

UDK: 681.3:007

COPICS predstavlja način reševanja komunikacijskih problemov v proizvodni delovni organizaciji. COPICS koncepti so usmerjeni na izdelovanje in povezovanje proizvodnih aplikacij. Niso pa neposredno povezani z drugimi področji (finančno, marketing, kadrovsko) , čeprav COPICS zbira podatke, ki so tem službem v veliko pomoč.

The Communications Oriented Production Information and Control System (COPICS) presents approches to the communications problems facing manufacturing companies. The COPICS concepts are oriented to production and related manufacturing applications. They are not concerned directly with other major areas, such as finance, marketing and personnel, although the COPICS approach collects data that will be helpful to these areas.

1. UVOD

COPICS (Communication Oriented Production Information and Control System) je IBM-ov komunikacijsko usmerjen informacijski sistem. Namenjen je proizvodnim delovnim organizacijam za zbiranje in posredovanje informacij, potrebnih za planiranje in vodenje proizvodnje. Vodstvu delovne organizacije in strokovnim službam omogoča spremljanje proizvodnih postopkov in hitro ukrepanje v primeru odstopanja od planov. COPICS je sestavljen iz organizacijskih in programskih rešitev. Delovni organizaciji nudi:

- standardiziran pristop za uvedbo proizvodnega informacijskega sistema v celotni delovni organizaciji
- standardno zgradbo upravljalnih programov, ki omogočajo bodoči razvoj informacijskega sistema
- preizkušene programske pakete
- banko podatkov o proizvodnji
- zbiranje podatkov iz proizvodnje za druge info. sisteme (kadrovski, finančni, marketing itd.)

Komunikacijsko usmerjen sistem predvideva povezavo uporabnikov sistema preko terminalnih enot (ekran s tastaturo, pisalnik) z računalnikom. Tak način dela omogoča:

- vnašanje podatkov v računalnik brez časovnih zakasnitev
- enostaven in hiter dostop do podatkov
- hitro odločanje na osnovi primerno pripravljenih podatkov

2. PLANIRANJE, IMPLEMENTIRANJE, NADZIRANJE

COPICS nudi komunikacijsko usmerjeno pomoč na treh širokih področjih:

- planiranje (izbiranje med več možnimi rešitvami)
- implementiranje (izdelava podrobnejših planov sledе na glavni plan)
- nadziranje (nadzor izvajanja planov in avtomatično opozarjanje odgovorne službe na odstopanja)

2.1. Planiranje

Planiranje se deli na strateško in taktično. COPICS se s strateškim planiranjem ne ukvarja. Pri taktičnem planiranju pa pomaga pri vsakodnevnih odločitvah (razporeditev delavne sile, proizvodni plan, določanje zalog itd.). Velika pomoč pri planiranju je simulacija. Na ta način lahko dobimo odgovore na vprašanja kot so:

- Kako vpliva sprememba izdelka na stroške?
- Za koliko bo nova strojna oprema znižala proizvodne stroške ?
- Za koliko se zmanjša celoten izdelovalni čas, če zmanjšamo zasedenost določenega stroja ?

Simulacija pomaga reševati tudi probleme pri spremembi proizvodnega plana. Planer dobi rezultate simulacije že v nekaj minutah. Prikazani so lahko številčno ali v obliki diagramov.

2.2. Implementacija

Med uvajanjem plana nastopijo razne spremembe. Informacije o teh spremembah morajo čim hitreje priti do odgovornih delavcev. COPICS nudi pri implementaciji sledečo pomoč:

- Omogoča pregled vplivov določenega delovnega naloga na proizvodne zmogljivosti.
- Zagotavlja pravočasno uvajanje konstrukcijskih in tehnoloških sprememb.
- Omogoča kontrolo potrebnega materiala in orodij.
- Prikazuje pregled delavnih obveznosti in proizvodnih aktivnosti.
- Omogoča pisanje delovnih nalogov.

2.3. Nadziranje

Sistem nadzira planirane aktivnosti in odkriva nepravilnosti, kot so:

- Na naročilu za nabavo je napačen datum dostave materiala.
- Nastopil je zastoj v proizvodnji.
- Količina izmeta je povečana.

Sistem rešuje nepravilnosti na sledeči način:

- V računalniškem spominu so shranjene vrednosti vseh nadziranaih veličin. Glede na razliko med predpisanimi in dejanskimi vrednostmi se sistem odloča. COPICS nudi tudi pomoč pri razširitvi predpisanih vrednosti.
- V spominu so shranjeni tudi ukrepi v slučaju nepravilnosti. Sporočila o ukrepih pošilja odgovornim osebam preko terminalnih enot.

3. GLAVNI CILJI SISTEMA

- prilagodljivost spremembam
- zmanjšanje zakasnitev
- učinkovito upravljanje

3.1. Prilagodljivost sistema

Info. sistem je tako oblikovan, da se je mogoče prilagajati zunanjim spremembam. Vzroki za spremembe sistema so lahko:

- spremembe na tržišču
- uvajanje novih proizvodov
- sprememba organizacije podjetja
- sprememba zasedenosti delavnih mest in s tem drugačen način dela

COPICS vse podatke shranjuje v banke podatkov. Te banke so tako organizirane, da niso odvisne od uporabniških računelniških programov. Odpravljena je dvojnost podatkov in s tem zmanjšan čas pristopa do podatkov. Reorganizacije banke podatkov zaradi novih uporabniških programov ne vplivajo na stare programe.

3.2. Zmanjšanje zakasnitev

COPICS dosega ta cilj na dva načina:

- obdelava podatkov v realnem času
- mrežna obdelava sprememb

Obdelava podatkov v realnem času pomeni:

- Podatki potujejo med terminalno enoto in računalnikom po telefoski liniji ali koaksialnem kablu. Podatke lahko preko računalnika pošiljamo tudi od ene do druge terminalne enote. Tako podatkov ni potrebno več pisati na papir in prenašati ročno.
- Spremembe vnešene preko terminalne enote so takoj zabeležene v računalniškem spominu. Ažurirani podatki so stalno na razpolago na magnetnih spominskih enotah.
- Sistem posreduje informacije preko terminalnih enot brez običajnih zamud zaradi prenašanja dokumentov.

Mrežna obdelava sprememb:

- Spremebe podatkov, ki so nastale v katerikoli točki sistema, COPICS takoj obdela. Prilasoditve planov se lahko na ta način izvedejo v vsakem trenutku.

3.3. Učinkovito upravljanje

COPICS pomaga pri vodenju proizvodnje na sledeče načine:

- Smulacija končnih učinkov sprememb.
- Vodilnim delavcem omogoča pregleden nadzor.
- Možen je hiter vpogled v banko podatkov. Mnogo setankov in telefonskih pogovorov ni več potrebnih.

4. PODROČJA UPORABE

Celoten COPICS informacijski sistem je razdeljen na 13 modulov, ki so namenjeni različnim službam v delovni organizaciji.

4.1. Upravljanje s tehnično-tehnološkimi podatki

V proizvodnih delavnih organizacijah je zelo važno, da so ažurni podatki v sestavnicah dostopni v vsakem trenutku. Pri uvajanju novega proizvoda, ki je podoben kateremu od obstoječih, nastopi problem kopiranja sestavnice obstoječega proizvoda. Take probleme pomaga reševati COPICS.

Cilji:

- Kreirati in vzdrževati tehnično - tehnološke podatke.
- Možnost dostopa do podatkov za vse službe v vsakem trenutku.

Funkcije:

- Kreiranje in spreminjanje banke podtkov preko ekrana in paketno.
- Prikazovanje sestavnic (na ekranu ali izpisano na papirju):
  - modularno (prikaz samo tistih delov, ki so vgrajeni v določen sklop v eni proizvodni stopnji)
  - strukturno (prikaz strukture proizvoda z vsemi deli preko vseh stopenj)
  - količinsko (prikaz vsakega dela samo enkrat s skupno količino za določen proizvod)
  - vsi našteti prikazi z uporabo izbirnih pogojev
- Pregled potreb (na ekranu ali izpisano na papirju):
  - modularno (prikaz vseh komponenet v keterih je določen del neposredno vgrajen)
  - strukturno (prikaz stopenj vgradnje vseh komponent v katere se del vgrajuje)
  - količinsko (prikaz skupne potrebne količine vseh komponent v katere se del vgrajuje)
  - vsi našteti prikazi z uporabo izbirnih pogojev
- Za vsako sestavnico je možno shraniti več variant.
- Kontrola pravilnosti podatkov in stopnje vgradnje.
- Kopiranje sestavnic.
- Pregled stanja tehničnih sprememb po datumu, statusu itd.
- Prikaz matičnih podatkov o materialih.

4.2. Naročila kupcev

Tržišče ceni poleg ustrezne kvalitete in usodne cene tudi efektivnost pri obravnavanju naročil. Veliko kupcev dodatno postavlja zahteve glede prejemnika, cene, dobavnih posojev, plačilnih posojev itd.
Ta programski modul omogoča obdelavo naročil s centralno kontrolo v vsaki fazi obdelave in vpogled v trenutne podatke.

Cilji:
- Omogočiti hiter odgovor na vprašanja v zvezi s stanjem naročila.

Funkcije:
- Interaktivno dodajanje, vzdrževanje in vpogled v podatke o kupcih, izdelkih in naročilih.
- Interaktivno dodajanje, vzdrževanje in vpogled v standardne tekstualne podatke in pozicije v naročilu.
- Ekranski prikaz podatkov o kupcih in izdelkih izbranih po določenem kriteriju.
- Ekranski prikaz stanja naročila po številki naročila, oznaki kupca ali izdelka.
- Paketno kreiranje banke podatkov:
  - kupcev
  - naročil
  - izdelkov
  - tekstov
- Paketno vzdrževanje bank podatkov o izdelkih in naročilih.
- Analiza stanja naročil po različnih kriterijih.
- Avtomatsko dodajanje številke naročila.
- Obdelava štirih tipov naročil:
  - posamična
  - pogodbena
  - odpoklici iz pogodbenih naročil
  - ponudbe
- Obdelava dveh tipov dobav:
  - takojšna dobava
  - dobava v prihodnosti

4.3. Napovedi

Vse proizvodne delovne organizacije se ukvarjajo z različnimi metodami napovedovanja bodočih dogodkov. COPICS-ov programski modul za izdelavo napovedi analizira pretekla povpraševanja in določa model za napovedi.

Cilji:
- Glede na to, da napovedi ne morejo biti nezmotljive, je cilj tega programskega modula vsaj izboljšati napovedi glede na trenutno uporabljeno tehniko in določiti tudi meje natančnosti napovedi.

Funkcije:
- Pretvarjanje podatkov v obliko primerno za izbor modela.
- Spremljanje odstopanja podatkov povpraševanja od izračunane povprečne vrednosti in opozarjanje na nenormalnosti.
- Po želji izračunavanje vrednosti prometa materiala.
- Izpisovanje liste izračunanih rezultatov za usotavljanje potrebnih korekcij.
- Izračunavanje vrednosti prvega in drugega izglajevanja.
- Izračunavanje baznih indeksov za sezonske artikle.
- Izpisovanje napovedi v obliki tabele ali diagrama.

4.4. Glavni plan proizvodnje

Glavni plan proizvodnje vsebuje planirane količine izdelkov datumsko porazdeljene. COPICS omogoča tudi korekcije plana (nujno naročilo kupca se lahko na zelo enostaven način naknadno vnese v plan).

Cilji:
- Izdelava dobre osnove za usotavljanje kratkoročnih potreb po surovinah, podsklopih in montažnih sklopih.
- Usotavljanje dolgoročnih potreb po proizvodnih zmogljivostih, delavni sili in finančnih sredstvih.

Funkcije:
- Interaktivno kreiranje glavnega proizvodnega plana glede na preteklo povpraševanje.
- Mrežne spremembe (spremenjena naročila kupcev, napovedi za prihodnost) direktno vplivajo na tekoči glavni plan.
- Prikaz glavnega plana za določen proizvod tabelarično ali grafično na ekranu ali listi.
- Spreminjanje, dodajanje in vpogled v banko podatkov o končnih proizvodih, potrebnem materialu, strojih, delavni sili in finančnih sredstvih. Za proizvodne delovne organizacije, ki izdelujejo kompleksne proizvode (strojna oprema, avtomobili, letala) je bolje, da namesto končnih izdelkov obravnavajo sestavne dele na najvišjem nivoju (podvozje, karoserija, motor).
- Analiza vpliva različnih sprememb glavnega plana s pomočjo simulacije preko ekrana.

4.5. Upravljanje z zalogami

V mnogih delovnih organizacijah je pregled nad zalogami zelo slab. Zaloge navadno naraščajo neplanirano. S finančnega stališča je najbolje, da zalog sploh ni. Vendar so zaloge potrebne. Mnoge zaloge so samo zaradi stanja na tržišču. Mnogokrat so zaloge direktno povezane z kvaliteto usluge do kupcev in zastoji v proizvodnji.
Ta programski modul je pripomoček za planiranje in kontrolo nivoja zalog končnih izdelkov, osnovnih materialov in delov v prizvodnji.

Cilji:
- Izboljšati usluge do kupcev z zmanjšanjem zakasnitev.
- Zmanjšati vlaganje v zaloge.

Funkcije:
- Analiza in klasifikacija skladiščnih delov na osnovi letne porabe in stroškov.
- Izračun ekonomičnih količin naročanja upoštevajoč različne metode za dele, ki jih vodimo po točki naročanja in tiste, ki jih obravnavamo s planiranjem potreb.
- Določanje varnostnih zalog po različnih metodah za dele vodene po točki naročanja in planske dele.
- Ekranski vpogled v zaloge.
- Interaktiven vnos, kontrola in obdelava vseh skladiščnih premikov.
- Možno je vodenje istega materiala na različnih skladiščnih mestih.
- Vodenje dnevnika prometa z vsemi premiki materiala v posebni banki podatkov.
- Pregled vseh premikov določenega materiala.
- Tiskanje dnevnikov prometa za kontrolo.

4.6. Plan proizvodnje

Glavni problemi v zvezi s planiranjem proizvodnje so: določanje prioritete delovnih nalogov, planiranje zmogljivosti (ozka grla) in pomankanje določenega materiala. V večini delovnih organizacij delovni nalog največ časa čaka na obdelavo.

Cilji:
- Čim manjše vmesne zaloge med dvema delovnima operacijama.
- Zmanjšanje izdelovalnih časov.
- Čim večji izkoristek strojne opreme.

Funkcije:
- Planiranje potrebnih zmogljivosti
  - interaktivno določenje datumov začetka operacij
  - usotavljanje viškov ali manjkov zmogljivosti
  - izpis poročil o zasedenosti zmogljivosti
  - izpis poročil o stanju delovnih nalogov
- Planiranje razpisa delovnih nalogov
  - določanje prioritete delovnih nalogov in datumov razpisa
  - ocenjevanje časa zaključitve delovnih nalogov
  - kreiranje banke podatkov o planiranem vrstnem redu razpisa nalogov
  - simuliranje sprememb zmogljivosti in razporeditve nalogov
- Terminiranje delovnih operacij
  - izračun prioritete delovnih operacij
  - kreiranje banke podatkov o vrstnem redu delovnih operacij
  - prikaz planiranega zaporedja operacij na ekranu in izpis na papir

4.7. Odpiranje in razpis delovnih nalogov

Programski modul za razpis delovni nalogov povezuje sistem planiranja s sistemom izvrševanja. Delovne naloge je potrebno odpreti in nato še razpisati. Rezultat odpiranja je rezervacija materialov, katerih potrebo usotovi program na podlagi sestavnice. V trenutku razpisa pa kontrolira razpoložljivost materialov in tiska delovno dokumentacijo.

Cilji:
- Poskrbeti je potrebno, da se vsak delovni nalog razpiše na planirani datum.
- Prekontrolirati razpoložljivost potrebnega materiala, orodij in delavniških risb za razpisani nalog.

Funkcije:
- Kreiranje in vzdrževanje banke podatkov o delovnih nalogih.
- Identificiranje delovnih nalog, ki jih je potrebno razpisati.
- Odpiranje nalogov preko ekranskega terminala.
- Razpis delovnih nalogov s kontrolo razpodožljivosti materialov.
- Priprava delavniške dokumentacije.
- Interaktivno vzdrževanje razpisne dokumentacije (količina, datum, lokacija, brisanje, zapiranje).
- Interaktivna odsovori na vprašanja o statusu delovnega naloga.

4.8. Nadzor in upravljanje z obrati

Mnogo težav v obratu nastane zaradi nepravilnega ali zakasnelega planiranja. Uporaba COPICS modulov za planiranje proizvodnje in razpis delovnih nalogov zmanjša te težave. Vendar kljub dobremu planiranju lahko nastopijo težave v proizvodnji (napaka na strojni opremi, nepredviden izmet). Vse naštete težave pa povzročijo zakasnitev zaključka delovnega naloga. COPICS-ov modul -Nadzor in upravljanje z obrati- zmanjšuje te zakasnitve s pomočjo povratnih informacij iz proizvodnje. Te informacije vnaša delavec preko ekranskega terminala ali pa je računalniški spomon povezan s CNC obdelovalnim strojem.

Cilji:
- Zagotavljanje planirane proizvodnje.
- Boljša koordinacija med proizvodnim in planerskim delom.
- Povečanje učinkovitosti proizvodnje z neposrednim nadzorom in računalniškim krmiljenjem strojev.

Funkcije:
- Zapisovanje vseh premikov delovne sile v obratu.
- Interaktivno ažuriranje potreb po materialu, dostave in izdaje iz skladišč.
- Ekranski pregled in spremembe dokumentacije.
- Interaktivno spreminjanje razporeditve dela.
- Pošiljanje sporočil delavcev preko ekrana (zmanjkalo materiala, ni električne energije, delo uspešno opravljeno).
- Interaktiven dostop do navodil za izvajanje kontrole in rezultatov kontroliranja.
- Avtomatsko sprejemanje podatkov o stanju stroja preko senzorjev na stroju. V slučaju uporabe CNC strojev pa tudi računalniško krmiljenje.

4.9. Vzdrževanje strojne opreme

Proizvodne delovne organizacije veliko sredstev vlagajo v strojno opremo. COPICS pomaga podaljševati življenjsko dobo in razpoložljivost strojne opreme. Na ta način pa se zmanjšujejo tudi proizvodni stroški.

Cilji:
- Čim bolj zmanjšati število okvar strojne opreme s čim nižjimi stroški vzdrževanja.

Funkcije:
- Pomoč pri izdelavi standardnih postopkov za vzdrževanje strojev.
- Avtomatska izdelava časovne razporeditve preventivnega vzdrževanja (čim manj okvar s čim manjšimi stroški vzdrževanja).
- Priprava delovnih nalogov za vzdrževanje.
- Vzdrževanje banke podatkov o rezervnih delih.
- Vrednotenje vzdrževanja in izsub zaradi neuporabnosti strojne opreme v času popravila.
- Poročila o vzdrževanju (oznaka in naziv stroja, čas, opis poravila, rezervni deli).

4.10. Nabava in prevzem

Mnoge delovne organizacije ne posvečajo dovolj pozornosti nabavi in prevzemu. Kompleksnost in zahtevnost planiranja in krmiljenja proizvodnje je veliko bolj očitna. Vendar pa nastane veliko problemov v proizvodnji prav zaradi zakasnele dobave ali neustrezne kvalitete materiala. COPICS pomaga zagotavljati potrebno količino in kvaliteto materiala ob pravem času.

Cilji:
- Seznaniti nabavo z vsemi podatki o dobaviteljih.
- Popis vsega prejetega materiala.
- Zbiranje podatkov o kontroli prevzetega materiala.

Funkcije:
- Kreiranje in vpogled v banko podatkov o

dobaviteljih in materialih, ki jih ti ponujajo.
- Izbiranje dobaviteljev sledé na njihove karakteristike (cena, kvaliteta, rok dobave).
- Priprava naročil (posameznega izdelka, več pozicij, planirana dobava, v konsignaciji, popusti in dodatki, odpoklici posodbenih naročil).
- Ekranski nadzor:
  - načinov nabave
  - zahtevkov ponudb
  - naročil pri določenem partnerju
  - materialov v konsignaciji
- Izpis naročil nabave, zahtevkov za ponudbe, statistik in poročil.
- Paketno brisanje neaktualnih podatkov z možnostjo shranjevanja na magnetni trak.
- Ekranski vnos podatkov o prejemu.
- Kreiranje in vpogled v banko podatkov o kontrolnih zahtevkih pri prevzemu.
- Zasledovanje gibanja prejetega materiala do končnega vskladiščenja.

4.11. Upravljanje s skladišči

Ta COPICS-ov program se ukvarja s fizičnimi problemi skladiščenja (določanje prostora za nove materiale, evidenca prostorske razmestitve, iskanje materiala).

Cilji:
- Vzdrževanje kontrole nad vsemi zalogami v delovni organizaciji.
- Zmanjšanje gibanja materiala med skladišči.
- Zmanjšanje stroškov skladiščenja.

Funkcije:
- Kreiranje banke skladiščnih podatkov o materialih.
- Ekransko beleženje vsakega premika materiala.
- Izpis liste iskanih pozicij (material, lokacija, količina) na osnovi ekranskega vnosa naloga za dvig materiala.
- Avtomatsko določanje prostora za nove materiale.
- Kontrola stanja na skladišču preko ekranskega terminala.

4.12. Planiranje in upravljanje s stroški

Za pravilne finančne odločitve je potrebna dobra analiza stroškov proizvodnje. Odgovoriti je potrebno na vprašanje, kolikšni so bili in kolikšni bodo stroški za vsak izdelek ter kje se pojavljajo nepravilnosti in kako jih odpraviti. Pomoč pri odgovorih na ta vprašanja nudi COPICS.

Cilji:
- Pomoč finančnim službam pri izračunu stroškov proizvodnje in pri odločanju.

Funkcije:
- Shranjevanje stroškov (material, delo, režija) po nivojih v skladu s sestavnico.
- Obračuni planiranih in stvarnih stroškov interaktivno in paketno.
- Interaktivno in paketno vzdrževanje podatkov o stroških.
- Ekranska simulacija predvidenih sprememb in obračuna.
- Interaktivni pregledi gibanja stroškov ter primerjava planiranih in stvarnih stroškov.

4.13. Prenos sporočil

To je COPICS-ov programski modul, ki omogoča prenos sporočil. Omogoča prenos podatkov med uporabniki in računalniškimi programi, med posameznimi uporabniki ali med posameznimi programi. Ta programski proizvod ni omejen samo na COPICS programe, ampak sa lahko vključujemo tudi v nove uporabniške programe.

Cilji:
- Pospeševati izmenjavo informacij z uporabo terminalnih enot in brez prenašanja dokumentov.
- Zmanjšanje naporov in materialnih stroškov pri spreminjanu sporočil.

Funkcije:
- Kreiranje, prikaz, spreminjanje in pošiljanje sporočil preko terminalnih enot.
- Usmerjanje sporočil med uporabniki.
- Povezovanje z interaktivnimi programi za obdelavo nekega sporočila.
- Aktiviranje uporabniškega programa glede na vnaprej določen dogodek (časovna točka, časovni interval, doseg podatkov).
- Shranjevanje sporočil in na zahtevo prikaz na ekranu po datumu in prioriteti.
- Onemogočanje dostopa do sporočil nepooblaščenim osebam.

5. POGOJI ZA UVEDBO COPICS SISTEMA

Za uspešno uporabo komunikacijsko usmerjenega sistema COPICS morajo biti izpolnjeni določeni pogoji. To je še toliko bolj pomembno za tiste uporabnike, ki se prvič srečujejo z obdelavo podatkov v realnem času.

5.1. Vodilni delavci

Uspešnost računalniško podprtega sistema je odvisna od tega kako vodilni in vodstveni delavci sodelujejo pri definiranju, uvajanju in uporabi sistema. Od njih se zahteva:
- poznavanje načina obdelave podatkov v realnem času
- sodelovanje pri definiranju ciljev sistema
- zagotavljanje dovolj velikih zmogljivosti delovne organizacije za dosegane predvidenih ciljev
- organiziranje učinkovitega uvajanja računalniško podprtega informacijskega sistema

5.2. Izobraževanje

Zelo važno je, da so vsi uporabniki seznanjeni z možnostmi sistema in načinom uporabe. Izobraževanje vodilnih delavcev naj vodijo delavci iz računalniške obdelave podatkov ali zunanji sodelavci. Vodilni delavci pa naj potem predstavijo sistem svojim podrejenim. Potrebno je tudi seznaniti vse uporabnike z načini oporabe terminalnih enot (ekran, tastatura, pisalnik).

5.3. Strojna računalniška oprema

Računalniki za paketno obdelavo podatkov se lahko z dodatnimi enotami za kontrolo linij, razširjenimi zmogljivostmi spomina in terminalnimi enotami uporabljajo za delo v realnem času.

- Terminali
  Potrebni so za vnos podatkov in prikaz rezultatov računalniških obdelav. Za

vnos so primerne tastature, magnetne kartice in svetlobna peresa. Za prikaz pa ekrani in matrični pisalniki za izpis manjših količin podatkov.

- Spomin
  Razširitev glavnega spomina je potrebna zaradi dodatnih krmilnih programov in več uporabniških programov, ki se istočsno izvajajo. Ker mora biti veliko število podatkov dostopnih v vsakem trenutku, morajo biti ti podatki shranjeni na magnetnih diskih.
- Računalnik
  V COPICS sistemu ni potrebno, da so vsi podatki shranjeni na centralnem mestu. Del podatkov in funkcij je lahko razporejenih po manjših računalnikih, ki so preko komunikacijski linij povezani s centralnim. Ti računalniki so uporabni tudi za direktno krmiljenje obdelovalnih strojev.

5.4. Programska oprema

Sistemi za delo v realnem času zahtevajo bolj zapleteno sistemsko programsko opremo kot paketno orientirani sistemi. Podatki se prenašajo med enotami brez računalniškega operaterja. Podpora sistemskih programov je potrebna na naslednjih področjih:

- Operacijski sistem
  mora omogočiti istočasno izvajanje več poslov. Napake pri vnosu podatkov je potrebno ugotoviti in po možnosti odpraviti avtomatsko. Potrebna je tudi statistika zasedenosti računalnika.
- Banke podatkov
  DL/1 (Data Language) sistemski proizvod je potreben za sestavo in vzdrževanje bank podatkov. Sprememba organizacije banke podatkov v okviru enega uporabniškega programa ne vpliva na definicijo drugih programov.
- Komunikacije
  Za vzdrževanje komunikacij med posameznimi terminalnimi enotami in računalnikom je potreben sistemski proizvod CICS/VS (Customer Information Control Sistem / Virtual Storage). Omogoča ponovno vspostavljanje komunikacij v primeru prekinitve in preprečuje dostop do podatkov nepooblaščenim delavcem.

5.5. Uvajanje COPICS-a

Pri uvajanju COPICS informacijskega sistema se navadno pojavi nekaj vprašanj.

- Ali je potrebno uporabniške programe najprej izvajati v paketni obliki potem pa šele v realnem času?

  Tehnike programiranja za paketne obdelave in obdelave v realnem času se med seboj zelo razlikujejo. Težko je pretvarjati programe iz ene oblike v drugo. Večinoma je najbolje, da se takoj uporablja programe v realnem času. Za obdelave, kjer je količina vhodnih podatkov velika, rezultati pa niso potrebni v nekaj sekundah, je primernejša paketna obdelava.

- Katere podatke bo možno videti in spreminjati?

  Za tiste, ki so namenoma zadrževali informacije bo ta sistem neustrezen. Za mnoge bo ta sistem pomenil spremembo dela. Vse te možnosti je potrebno predvideti vaprej. Dostop do podatkov pa je večkratno zavarovan.

- Ali mora biti obdelava v realnem času centralizirana?

  To vprašanje se pojavlja pri opravljanju določenih obdelav za lokacijsko ločene obrate. Potrebno je analizirati stroške za komunikacijske linije, dodatno računalniško opremo in primerjati hitrosti obdelav. Uvajanje določenega informacijskega sistema je lažje, če je računalnik na lokaciji uporabnika sistema.

- Koliko časa traja uvajanje?

  Možno je uvajati več modulov COPICS-a istočasno. Vendar pa uvajanje vseeno traja nekaj let. Sevada pa sistem ni dokončen. Z uporabo novih načinov dela in izpopolnjevanjem opreme moramo spremeniti tudi informacijski sistem.

6. OCENA UPORABNOSTI

COPICS programski paket ima naslednje pomanjklivosti:

- Pred uporabo je potrebno prevesti vse izpise na ekranih in listih v slovenski jezik in programe ponovno prevesti ter katalogizirati.
- Potrebno je vse banke podatkov prilagoditi zahtevam delovne organizacije glede dolžine in števila podatkov.
- Večina programov ni dosdopna v izvorni obliki.
- Mesečna najemnina celotnega paketa je nekaj 1.000 $.

Uporaba COPICS-a pa omogoča naslednje izboljšave:

- Vse odločitve so glede na isto banko podatkov. Ni dvojnosti podatkov.
- Zmanjšanje števila dokumentov. Prihranek pri papirju, pisanju in prenašanju dokumentov. Informacije so kratke, točne in hitre.
- Zmanjšanje zalog končnih izdelkov, sklopov, surovin in rezervih delov.
- Boljše delo prodajne službe. Manjše zakasnitve pri dobavi in boljši pregled nad naročniki.
- Boljša izraba proizvodnih pripomočkov.
- Skrajšanje izdelovalnega časa.
- Povečanje produktivnosti in kvalitete zaradi krmiljenja proizvodnih aktivnosti z računalnikom.
- Večja učinkovitost služ, ki direktno podpirajo proizvodnjo (vzdrževanje, orodjarna, transport, skladišča).
- Bolj realistično planiranje.
- Natančno določanje stroškov izdelka. Predvidevanja bodočih stroškov.
- Skrajšanje časa dogovarjanja različnih služb pred končno odločitvijo.
- Zmanjšanje nabavnih stroškov. Nabavnik ima podatke prej in bolj natančne.
- Manj težav pri vzdrževanju in uvajanju novih računalniških sistemov.

Literatura:

IBM G320 - Communication Oriented Production Information and Control System - Volume I - VIII

INFORMATICA 1/85

METODOLOGIJA TESTIRANJA MAGNETNIH MEHURČNIH POMNILNIKOV

P. KOLBEZEN,
S. MAVRIČ,
J. ŠILC,
B. MIHOVILOVIĆ

UDK: 681.3.325.6.08

INSTITUT „JOŽEF STEFAN", LJUBLJANA

Osnovne operacije delovanja magnetnega mehurčnega pomnilnika so generacija, prenos in podvojitev mehurčka. Te operacije se včasih lahko pojavljajo tudi spontano zaradi okvar na pomnilniškem elementu. Nepravilnosti v delovanju moremo ločiti na štiri glavne tipe: izginitev, deformacija, samogeneracija in samopodvojitev mehurčka. Testiranje magnetnega mehurčka lahko izvajamo na več različnih nivojih. Obravnavani so posamezni nivoji pristopa k testiranju detektiranih napak v materialu, napak zaradi nezaželjenih magnetnih vplivov ali pomanjkljive časovne kontrole.

TEST METHODOLOGY OF MAGNETIC-BUBBLES MEMORIES. Although bubble generation, transfer, and replication are normal operations in a magnetic-bubble memory these events can occur in an unwanted fashion because of defects in the device. The anomalies of bubble memory function that cause device failureare of four major types: bubble collapse, strip-out, self-generations and self-replication. The testing of magnetic-bubbles memories can be done at several different levels of complexity. Various levels of incoming testing detect material defects, improper magnetic biasing, or poor timing control are discussed.

1. UVOD

Magnetni mehurčni pomnilniki (MMP) si počasi, pa vendarle vztrajno utirajo pot na mnogotera področja uporabe. Uporabniki so prišli do zaključka, da poleg odličnih in tudi nekaterih izjemnih lastnosti MMP za vrsto aplikacij ne predstavja več predrag neizbrisljiv medij. Čeprav tovrstno pomnilniško napravo testira že proizvajalec, želi napravo testirati tudi uporabnik sam, predno jo vgradi v svoj produkt.

S problemi testiranja se sooča tudi grupa sodelavcev Instituta "Jožef Stefan", ki ob finančni pomoči RSS razvija prvi večji mehurčni pomnilnik za potrebe našega industrijskega okolja. Razvila je tiskano vezje, ki z vgrajenim krmilnikom in štirimi 1 megazložnimi pomnilničkimi elementi Intel 7110 predstavlja osnovni pomnilniški modul kapacitete 128, 256 ali 512 K zlogov. Kapacitete so odvisne od števila (1,2 ali 4) vgrajenih pomnilniških elementov. Osnovni pomnilniški modul je mogoče razširiti na kapaciteto 1 M zlogov z razširitvenim modulom kapacitete 512 K zlogov. Ekspanzijski modul nima vgrajenega krmilnika, saj more krmilnik na osnovnem modulu izkrmiliti do osem pomnilniških elementov. Osnovni in razširitveni modul predstavljata skupaj enmegazložni pomnilnik. Takšne enmegazložne pomnilnike je mogoče še nadalje združevati v večje mehurčne pomnilniške sisteme.

V delu [1] je podrobneje prikazana zgradba osnovnega MMP elementa. Sestavljajo ga pomnilniški čip, dve med seboj pravokoni navitji in dva permanentna magneta. Te elemente obdaja še kovinski oklep, ki ščiti notranjost čipa pred zunanjimi magnetnimi polji.

Permanentna magneta, ki sta vgrajena v pomnilniški element, ustvarjata osnovno polje. To ohranja magnetne domene ali mehurčke, od katerih vsak predstavlja en bit. Pod vplivom tega polja obstajajo mehurčki tudi tedaj, ko pomnilniški element ni pod napajalno napetostjo, in daje elementu lastnost neizbrisljivosti pomnenja. Poljsko jakost izberemo tako, da se navzljic spremenljivemu magnetnemu polju zaradi temperaturne odvisnosti magnetni mehurčki v epitaxialnem filmu ohranjajo. Nahajati se morajo v delovnem temperaturnem območju MMP. Glej črtkano polje na sliki 1!

Če postane polje premočno, se domene v filmu zvrstijo v linijo. Mehurčki, ki pa postajajo z večanjem poljske jakosti vse manjši, izginejo. V nasprotnem primeru, če poljska jakost upada, velikost mehurčka narašča in pri dovolj majhni jakosti preidejo domene v podolgovato vijugasto obliko, kakršna je značilna za naravno stanje. To pa pomeni, da mehurčki ne obstajajo več.

Operacije generiranja, propagacije, prenosa in podvojitve mehurčkov, ki so potrebne za shranjevanje in čitanje podatkov v MMP, se dosežejo z določenimi vplivi na dva sloja, nanešena na epitaxialni film. Aluminij-bakrov sloj je vzorčen s funkcijskimi elementi, ki ustvarjajo lokalna magnetna polja. Ta polja nastajajo pod vplivom tokovnih impulzov, ki imajo natančno določeno dolžino in amplitudo. Lokalna polja se prištevajo ali odštevajo zunanjemu polju, ki ga ustvarjata oba permanentna magneta, in omogočajo v filmu operacije generiranja, zamenjave ali cepitve magnetnih mehurčkov.

Drugi sloj vsebuje permalojne metalne vzorce, navadno imenovane škarnice, ki so nanešeni na oksidni sloj preko aluminij-bakrovih vzorcev. Rotirajoče magnetno polje, ustvarjeno z dvema med seboj pravokotnima navitjema, magnetizira te škarnice in s tem omogoča propagacijo magnetnih mehurčkov od ene škarnice do druge.

V MMP-napravah najbolj pogosto srečujemo dve osnovni arhitekturi:
- arhitektura z "major-minor" zanko, in
- arhitektura s kopiranjem blokov.

Pri obeh arhitekturah so kritične časovne karakteristike signalov, ki omogočajo posamezne operacije MMP. V primeru arhitekture z major-minor zanko se podatek shrani tako, da se najprej generira v enojni major zanki, ki rabi za vpisovanje podatkov, in nato prenese v številnejše minor zanke.

2. ČASOVNI VPLIVI

Prenos mehurčkov iz ene zanke v drugo omogoča posebno oblikovan propagacijski element v permalojnem vzorcu z aluminj-bakrovim kontrolnim elementom. V pravem času, ko tokovni impulz steče skozi kontrolni element in povzroči kakšno lokalno spremembo magnetnega polja, se mehurček odtrga od škarnice glavne (major) zanke in vgnezdi v škarnico manjše (minor) zanke. S spremenjenimi časovnimi razmerami tokovnega impulza je možen prehod mehurčkov iz minor zank v glavne zanke in s tem dostop do shranjenih podatkov.

V MM pomnilnikih, katerih arhitektura omogoča kopiranje blokov, se podatek, ki ga želimo shraniti, najprej generira v vhodni sledi in nato z izmenjavo blokov shrani v minor zanke. Čitanje podatka poteka v obratni smeri, pri čemer se izmenjajo bloki podatkov med minor zankami in posebno izhodno sledjo. Ta rabi za detekcijo podatkov. Bistvena razlika med obema arhitekturama obstaja v obliki prenosnih elementov, ki povzročajo cepitev mehurčka v dva mehurčka. Slednje imenujemo kopiranje, kar učinkuje kot prenos mehurčka.

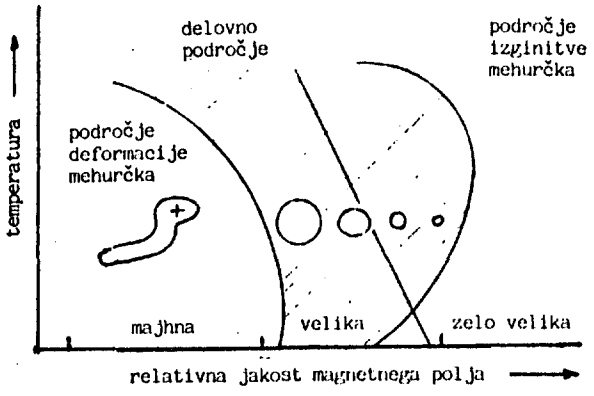
Generiranje, prenos in podvojitev (kopiranje) so običajne in vedno nadzorovane operacije v MMP. Vendar se te operacije lahko pojavljajo tudi nezaželjeno zaradi okvar na pomnilniškem elementu. Nepravilnosti v delovanju MMP moremo opredeliti v štiri glavne tipe:
- izginitev mehurčka
- deformacija mhurčka
- samogeneracija mehurčka, in
- samopodvojitev mehurčka.

Kot smo že omenili, izvirata prvi dve nepravilnosti od premočnega ali prešibkega lokalnega magnetnega polja. Nepravilnosti drugih dveh tipov pa so lahko odvisne od več faktorjev, kot so: nepravilno oblikovani propagacijski elementi, nepravilno izdelan permalojni material, ki je nanešen med minor zankami, ali nenatančna nastavitev jakosti magnetnega polja.

MMP lahko testiramo tako, da povečujemo kvarne vplive, ki povzročajo omenjene tipe napak. Pred tem napolnimo vse podatkovne pozicije v major in minor zankah z mehurčki ali uporabimo druge podatkovne vzorce, ki omogočajo lažje razspoznavanje napak.

3. MAGNETNI VPLIVI

Zaradi magnetne narave in interakcij med mehurčki je možnost napak v delovanju pomnilnika odvisna od gostote mehurčkov in njihove



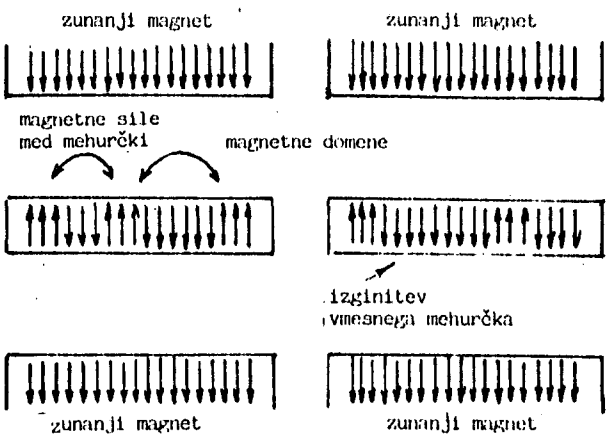
Slika 1. Robni pogoji delovanja MMP

porazdelitve pri uporabljenem vzorcu. Vsak mehurček namreč generira nasprotni si zunanji magnetni polji in z njima deluje na sosednje mehurčke. Ta situacija je podobna situaciji treh paličastih magnetov, ki so položeni paralelno eden poleg drugega. Če je poljska jakost dovolj velika, se magnet v sredini zasuka in tako združi nasprotni polji.

Taisti fenomen se pojavlja v epitaksialnem filmu MMP. Če je magnetna poljske jakost permanentnih magnetov previsoka, lahko interakcije med mehurčki pri "polno naloženem" pomnilniku povzročajo, da vmesno ležeči mehurčki spremenijo polariteto (glej sliko 2 !). To pa pomeni, da mehurčki izginjajo.

Vzrok izginitve mehurčka je lahko tudi neprimerno oblikovan propagacijski element ali nepravi permalojni material, ki je nanešen med ali okrog propagacijske zanke. Element z napako lahko povzroči, da mehurček "lebdi" ali se giblje prepočasi in ovira druge mehurčke. Zaradi nastale ovire lahko eden od mehurčkov izgine. V drugih primerih je lokalno magnetno polje, ki ga povzroča okvarjen element, tako močno, da mehurček na takšnem elementu izgine.

Testiranje MMP na izginitev mehurčka v najneugodnejših okoliščinah (worst-case test) opravimo z zapisovanjem logične "1" na vse pozicije bitov v napravi. Na ta način kontroliramo, če ni poljska jakost permanentnih magnetov prevelika na celotnem območju delovanja mehurčnega sloja, če magneti omogočajo delo-



Slika 2. Medsebojni vpliv sosednih mehurčkov

vanje v temperaturnem območju, ki ga predpisuje proizvajalec, in če kakšni nepravilno oblikovani elementi ne povzročajo izginitev mehurčka.

4. NAPAKE PROPAGACIJE MEHURČKOV

Deformacija magnetnega mehurčka se pojavlja pri prekomerno zmanjšani jakosti permanentnih magnetov. Polje je tedaj prešibko, da bi preprečilo večanje in s tem vračanje domen v njihovo normalno prvobitno vijugasto obliko. Takšen pojav zasledimo tudi zaradi okvar na propagacijskih elementih, vsled katerih se lahko spremeni jakost permanentnega magnetnega polja.

Testni postopek odkrivanja tovrstnih napak se razlikuje od postopka odkrivanja napak zaradi mehurčne izginitve. Razlika je v podatkovnih vzorcih, ki jih v enem ali drugem primeru uporabljamo. Pri analizi nepravilnega delovanja MMP, ki nam omogoča odkriti prešibko magnetno polje pri robnih pogojih delovanja, uporabljamo vzorec, ki ima nekaj redko razmeščenih mehurčkov preko vseh delujočih zank. Če se mehurčki pod takšnimi pogoji deformirajo (dobijo podolgovato vijugasto obliko), se bodo pojavili podatki tudi na drugih lokacijah bitov, na katere predhodno nismo vpisali podatka. Takšen test nam ne pomaga odkriváti le napake vsled prešibkega magnetnega polja, ampak tudi napake, ki se pojavljajo v MMP zaradi nepravilno oblikovanih škarnic. Omenili pa smo že, da včasih tudi takšne škarnice povzročajo deformacijo mehurčka.

O samogeneraciji govorimo, če se spontano izoblikujejo nezaželjeni mehurčki v MMP. Navadno se to dogaja pogojno zaradi premočne magnetne poljske jakosti, ki naraste zaradi previsoke notranje temperature na mejo delovnega območja. Samogeneracijo v propagacijskih zankah odkrivamo z vzorcem, ki vsebujejo same logične "0" na vseh podatkovnih pozicijah v MMP. Nato z večkratnim odčitavanjem vsebine MMP ugotavljamo, če so se spontano generirali kakšni mehurčki kjerkoli v pomnilniškem čipu. Generiranje nezaželjenih mehurčkov odkrijemo z detektiranjem logične "1" v izhodnem podatku.

Vzrok samopodvojitve mehurčka najčešče tiči v nepravilno oblikovanih elementih. Na sliki 3 vidimo, da je podvojitev lahko vertikalna (znotraj iste zanke) ali horizontalna (med zankami).

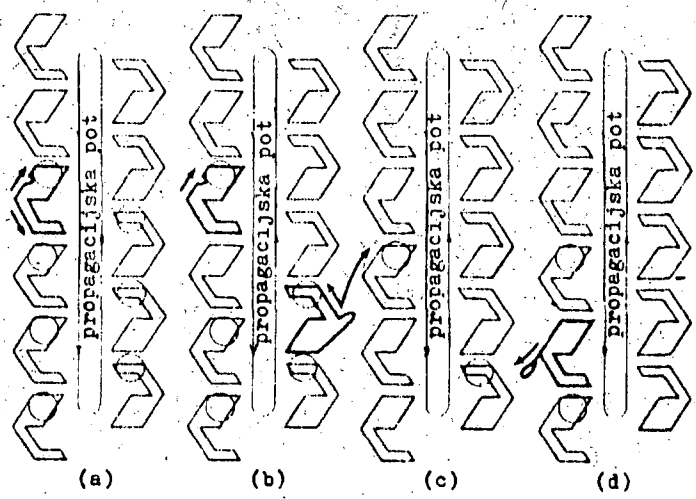
Tovrstne napake odkrijemo tako, da v MMP vpišemo več strani samih logičnih "1", le-tem pa pridružimo več strani samih logičnih "0". Zaradi vertikalne podvojitve se bodo pojavili mehurčki v podatkovnih straneh, ki bi sicer morale biti prazne. Test s takšnim vzorcem odkrije tudi napako v zanki, če "pade" zadnji mehurček v nizu nazaj v zanko zaradi magnetne interakcije med mehurčki.

Slika 3.a kaže zarezo v eni izmed škarnic na levi strani zanke. Magnetna polarizacija, ki nastane v tem škarniku, in interakcija med mehurčki lahko povzročita, da se poslednji mehurček v nizu podvoji in pade nazaj na prvotno mesto ali tudi za več mest nazaj. Rezultat tega je, da se pojavi pri čitanju pomnilnika na straneh podatkov, ki so vsebovali samo logične "0", tudi nekaj logičnih "1". Na sliki 4.b je prikazan primer, ki kaže nekakšno nagnenje mehurčka k "padcu" le-tega nazaj na prvotno lokacijo. To nagnjènost odklanjamo tako, da najpreje aktiviramo serijo start-stop operacij, s katerimi dosežemo takojšno osamitev poškodovanih elementov.

Horizontalna samopodvojitev nastane, če se podatek nepravilno podvoji in prenese iz ene zanke v drugo. Testiranje takšnega tipa napak izvajamo z vpisovanjem vzorcev, sestavljenih iz več "1", katerim sledi več "0" znotraj ene same stranice podatkov. Ponavljanje takšnih podatkovnih vzorcev v MMP-čipu omogoča odkrivanje poškodb, kakršne so prikazane na sliki 3.c in 3.d. Le-te povzročajo, da se mehurčki ali podvojijo ali preprosto skočijo iz polne v prazno zanko [4, 6].

5. NIVOJI TESTIRANJA

Razlikovati moramo predvsem testiranje in odkrivanje napak pri razvoju ali vgradnji MMP v uporabniški sistem na eni strani, ter testiranje in odkrivanje, kot tudi popravljanje napak MMP med eksploatacijo le-tega v uporab-



Slika 3. Vpliv poškodb na propagacijskih elementih

niškem sistemu na drugi strani. Zato govorimo o testiranju MMP na dveh makro nivojih.

Na prvem makro nivoju izvajamo testiranje MMP na treh različnih nivojih kompleksnosti. Ker pomnilniške komponente in napravo kot celoto testira že sam proizvajalec, večina uporabnikov meni, da je že iz ekonomskega vidika neupravičeno, da na prvem nivoju pomnilniške elemente še sami ponovno testirajo.

5.1. Prvi nivo testiranja obsega vse v prejšnih poglavjih obravnavane teste v predpisanem temperaturnem delovnem področju mehurčnega pomnilniškega sistema, ki je sestavljen iz pomnilniškega elementa in kontrolnih vezij na tiskanini. Govorimo o testiranju na nivoju tiskanega vezja MMP. Po odkritju napake moramo s ponavljajočim čitanjem pomnilnika ugotoviti, če obstaja napaka mehke ali trde narave. Mehke napake se pojavljajo predvsem zaradi napačnega detektiranja mehurčkov med operacijo čitanja. Navadno lahko tovrstne napake odpravljamo preprosto z večkratnim odčitavanjem podatkov.

Skrbno izdelani kliŠeji tiskanih vezij mehurčnih pomnilnikov v marsičem pripomorejo k zmanjšanju števila pojavljajočih se mehkih napak na minimum. To dosežemo s tem, da poskrbimo za ustrezne preseke vodnikov, njih razporeditev, kot tudi primerno razmestitev posameznih komponent vezja na tiskanini. Pomembno je, da vodimo vodnike od izhoda detektorja do ojačevalnika izhodnih impulzov (sense amplifier SA) čim dlje proč od vodnikov, ki so lahko generatorji šumov, kot tudi od vodnikov vzbujevalnih tokov pravokotnih navitij. Pri tem pa naj bodo poti čim krajše.

Problematika trdih napak tiči znotraj pomnilniškega elementa. Za tovrstne napake je značilno, da se pojavljajo ob vsakokratnem čitanju pomnilnika. Prav tako, kot v prejšnem primeru, pa je potrebno testni postopek ponavljati, da moremo ugotoviti, če izhaja napaka iz pomnilničkega elementa samega, ali le vsled slabega delovanja celotnega vezja. Sistemu omogočimo nepretrgano delovanje v čitalno - pisalnem ciklu in posebej le v čitalnem ciklu. Na osnovi detektiranih napak določimo povprečni čas javljanja napak (MTTE) v sistemu.

5.2. Večji komplekstnostni nivo testiranja je drugi nivo. Na tem nivoju tesiranja upoštevamo digitalno - analogne pretvornike in diskretna vezja, ki se priključujejo pomnilniškemu elementu na tiskanem vezju MMP. Na tem nivoju testa spreminjamo napetosti in amplitude funkcijsko odvisnih tokov v območju danih specifikacij in preko temperaturnega območja pomnilniškega elementa. Ta nivo testiranja lahko še razširimo na gradnike diskretnega mehurčnega krmilnika tako, da spreminjamo časovne karakteristike različnih kontrolnih signalov v določenem temperaturnem območju.

Opisana metoda je za večino uporabnikov prepočasna in predraga. Alternativen pristop k testiranju na tem nivoju je takšen, da testiramo podporna vezja že pred vgraditvijo pomnilniškega elementa na tiskano vezje. Ta alternativa je primerna še posebej za razvijalce MMP vezij.

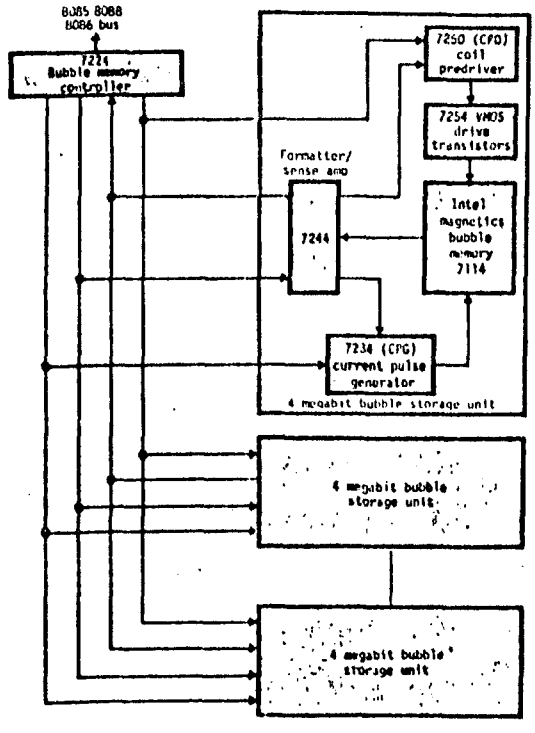
Komercijalno dosegljive naprave za testiranje tiskanih vezij MMP merijo dolžino in amplitudo izhodnih signalov funkcijskih gonilnikov in časovnih generatorjev [2]. Ko uspešno končamo testiranje podpornih vezij na tiskanem vezju MMP-sistema, dodamo še pomnilniške elemente. Nato testiramo sistem v celoti in napravimo analizo napak na nivoju vezja MMP.

5.3. Na tretjem nivoju testiranja poskušamo določiti nastavitev magnetnega polja z merjenjem robnih pogojev delovanja mehurčnega pomnilniškega elementa. Jakost permanetnega magnetnega polja ni mogoče meriti direktno, ker so magnetni pomnilniški elementi zaščiteni s kovinskim oklopom. Zato lahko normalne delovne pogoje določimo le z zelo močnim zunanjim poljem. Z večanjem in zmanjševanjem vpliva takšnega zunanjega polja opazujemo, kdaj nastopi izginitev oziroma deformacija mehurčkov. Normalizirane delovne pogoje dobimo z jakostjo polja, katerega vpliv spreminjamo v mejah, znotraj katerih se ne pojavljajo napake enega ali drugega tipa.

Na nivoju uporabnika je opisani postopek zelo drag in lahko povzroči več vmesnih, nastavitvenih in korekcijskih problemov. S tem postopkom lahko nehote vplivamo na posebne zanke, kot so "masked-off" ali "redundancy-map" zanke, in s tem uničujemo porajajoče se mehurčke. Zato mora imeti uporabnik pri testiranju na tem nivoju na voljo naprave, s katerimi lahko doseže pogoje za popolno izginitev mehurčkov, za regeneriranje porajajočih se mehurčkov ali nekih redundančnih podatkov za zanke preslikav (map loops). Zato je za večino uporabnikov lažje, da prepostijo testiranje vpliva magnetnega polja na pomnilniški element proizvajalcu le-tega.

Vrteče magnetno polje, ki je potrebno za delovanje magnetnega mehurčnega elementa, dobimo z dvema sinusnima signaloma. Ta sta med seboj fazno premaknjena za približno 90 stopinj. Neizbrisljivost shranjenega podatka je zajamčena le, če se vrteče magnetno polje ustavi in požene v točno določenem faznem zamiku.

Načrtovalci pomnilniških modulov uporabljajo za določanje faznih zamikov običajen polari-

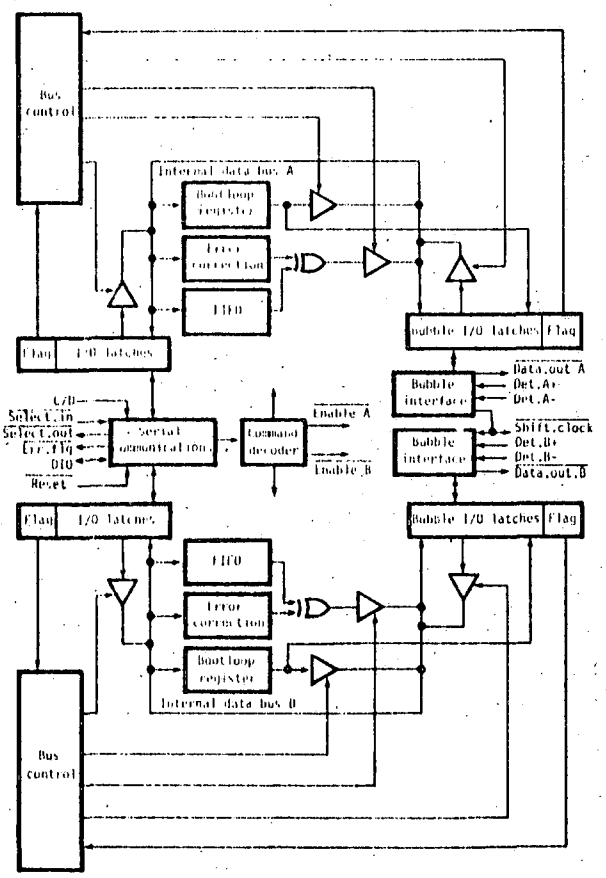


Slika 4. Osnovni podsistem MMP

zacijski registrator vzbujevalnih tokov rotirajočega magnetnega polja. Lissajousove slike so učinkovit pripomoček metode za analizo omenjenih sinusnih signalov, saj nam omogočajo opazovanje faznih zasukov obeh signalov na zaslonu osciloskopa in s tem nastavitev startnih karakteristik vzbujevalnih tokov vrtilnega magnetnega polja [3].

5.4. Drugi makro nivo testiranja se uporablja pri eksploataciji MMP. Tedaj se sprotno izvaja odkrivanje, lokalizacija in popravljanje napak. Sodobna tehnologija in organizacija mehurčnega pomnilniškega elementa, krmilnika in FSA [6,7] omogočajo z izvajanjem omenjenih aktivnosti veliko zanesljivost v delovanju pomnilniškega sistema [5, 7]. Nadzor nad naštetimi aktivnostmi je dodeljen krmilniku. Pri tem je posebno pomembno, da omenimo odkrivanje dobrih in slabih zank MMP neposredno po izdelavi čipa že pri proizvajalcu samem, ki tovrstni podatek posreduje kupcu.

Pretok podatkov med gostiteljskim sistemom in MMP mora najpreje potekati preko oblikovalno-čitalnih ojačevalnikov (FSA na sliki 4). Kot je razvidno iz slike 5 ima FSA dva identična kanala, ki imata vgrajeno vezje za korekcijo napak in serijske "prvi v - prvi iz" (FIFO) registre. Krmilnik nadzoruje aktivnost omenjenih ojačevalnikov z ukazi, ki jih pošilja na enojne, serijske, obojestrano usmerjene podatkovne vhodno-izhodne linije (DIO). Štiri bitni ukazi, ki jih pošilja krmilnik ojačevalnikom FSA, so časovno multipleksirani na isti enojni liniji (DIO). Tako imenovani



Slika 5. Ojačevalnik in oblikovalnik podatkov (realiziran z Intelovim 7224 FSA)

"boot-loop register" ima pomembno nalogo pri obojestranem prenosu podatkov med gostiteljskim računalnikom in pomnilnikom. 160 bitni register [7] vsebuje informacijo, ki natančno podaja konfiguracijo dobrih in slabih zank v ustreznem kanalu vsakega MMP.

Vsakemu bitu registra odgovarja določena minor zanka v pomnilniškem čipu (npr. v Intelovem čipu 7114 s kapaciteto 4 Mbitov). Pri prenosih podatkov skozi vhodno/izhodne registre MMP se med čitanjem s pomočjo vsebine "boot-loop registra" upoštevajo slabe zanke. Med vpisovanjem v MMP element se namreč vpisujejo ničle na vse tiste lokacije bitov, ki odgovarjajo slabim zankam. Tedaj, ko je omogočena korekcija napake, vsebuje "boot-loop register" natanko 135 enic pri 270 dobrih zankah na kanal mehurčnega pomnilnika.

V bloku korekcije napak je uporabljena posebna 14-bitna "goreča koda", ki je namenjena detekciji in korekciji napak. Če uporabnik omogoči proces detekcije in korekcije, posebno vezje za korekcijo napak doda na koncu vsakega 256 bitnega bloka, ki med vpisovanjem potuje skozi FIFO register, še omenjeno 14 bitno kodo. Med operacijo čitanja pa vezje za korekcijo napak pregleduje blok podatkov in preko zastavce ERR FLAG sporoča pomnilniku, če se v podatkovnem bloku pojavi kakšna napaka.

Na ukaz "read MBM data" FSA čita podatke iz MMP. Prečitani podatki so na osnovi vsebine "boot-loop registra" selektirani tako, da se v FIFO registre FS ojačevalnikov vpisujejo le tisti podatki, ki prihajajo iz dobrih zank. V primeru korekcije napak, se podatek prebere na poseben način. Tedaj mora biti blok podatkov (to je 270 bitov) v celoti prenešen v FIFO, še predno se kateri bit prebere iz FIFO registra. Le na ta način lahko vezje za korekcijo napak le-te odkrije in s programsko prekinitvijo krmilnika pravočasno prepreči nadaljni prenos podatkov. V primeru, da napak ni, se 270-bitni blok prebere iz FIFO registra v krmilnik, a v FIFO register se med tem že vpiše naslednji blok.

Ukaz "internally correct data" prisili FS ojačevalnik, da v notranjem ciklu ojačevalnika podatki potujejo skozi vezje za korekcijo napak, ne da bi bil kateri od njih poslan krmilniku. Na koncu te operacije pošlje ojačevalnik CORRERR ali UNCORRERR bit v statusni register krmilnika. Če je napaka popravljiva, prične krmilnik izvajati ukaz "read corrected data".

V ciklu izvajanja ukaza "read correct data" potujejo podatki skozi ECC vezje. Nato se vseh 256 bitov prenese nazaj v krmilnik. FSA status register označuje prisotnost napake, ki je lahko popravljiva ali ne. Gornji ukaz se pojavi tudi tedaj, ko so podatki predhodno popravljeni z ukazom "internally correct data". Oba ukaza združeno omogočata tri nivojski ECC, ki je natančneje opisan v delu [7].

6. ZAKLJUČEK

Delo je nastalo v okviru sodelovanja Instituta "Jožef Stefan" z Iskra-Telematiko, Kranj. Razvit je bil osnovni modul magnetnega mehurčnega pomnilnika kapacitete 128, 256 ali 512 kilozlogov. Zasnovan pa je tudi ekspanzijski modul, ki povečuje kapaciteto pomnilnika na en megazlog.

V prvih treh razdelkih smo obravnavali problematiko odkrivanja in določanja vrste napak v delovanju magnetnega mehurčnega pomnilni-

škega modula na tako imenovanem prvem makro nivoju. Obravnavana metodologija testiranja na tem nivoju je uporabljiva pri razvoju in vgrajevanju MMP v uporabniški sistem. Proizvajalcu MMP elementa omogoča zanesljivo odkrivanje slabih zank. Odpravljanje posledic le-teh v pomnilniškem čipu pa smo obravnavali v razdelku 5.4.

Zaključimo lahko, da je za uporabnika mehurčnega pomnilniškega medija bistven prvi nivo testiranja. To je testiranje na nivoju tiskanega vezja, pri čemer uporabnik izvaja funkcionalno testiranje z rabo obravnavane metodologije testiranja na tem nivoju. V dodatku je podan opis programa funkcionalnega testiranja 0,5 megazložnega mehurčnega pomnilnika na plošči tiskanega vezja, ki je bil najprej testiran na Intellecovem razvojnem sistemu, po preprojektiranju in prilagoditvi vodila pa še na gostiteljskem 16-bitnem mikro računalniku TK 68000.

7. LITERATURA

[1] P.Kolbezen, R.Trobec, J.Šilc, B.Mihovilovic: Mehurčni pomnilniki, IJS Ljubljana, Raziskovalna študija, št.pogodbe: 03-BR-PK-1226/81, junij 1981.

[2] Alina Deutsch, John D.Mackay, Mark H Kryder, Mitchell S. Cohen, Arnold Halperin, Fred W. Stukey: Magnetic Bubble Memory Exerciser,IEEE Transactions on magnetics, vol.MAG-16, No.2, March 1980.

[3] Joe E. Neuhauser: Oscilloscope Technique for Checking Bubble Memories, Electronics Test, pp. 10-11, May 1979.

[4] Dan Harmon: Test strategies find faults in user's bubble memories, Electronics, pp. 145-148, June 2, 1981.

[5] John E. Davies: The 7110, A One Megabit Magnetic Bubble Memory, Intel Magnetics, Realibility Report.

[6] J.E.Davies: Realibility Considerations in the Design of One-megabit Bubble Memory Chips, IEEE Transactions on Magnetics, Vol.MAG-16, No. 5, pp. 1106-1110, September 1980.

[7] D.Dossetter, H.Washburn, S.Nicolino and D.Pierce: New bubble memory packs in 4 M bits, Part two, Electronic Engineering, pp. 47-53, January 1983.

DODATEK

Testiranje mehurčnega pomnilniškega elementa i7110 na mikroračunalniškem sistemu Intellec MDS in sistemu TK 68000. Testni program je napisan v zbirnem jeziku Intel 8080 in Motorola 6800.

Algoritem testiranja:

```
begin

 inicializacija pomnilniškega sistema ;
 stanje := 0 ;  {trenutno stanje: testiranje }
 page := 0 ;    { trenutno testirana stran }
 stnapak := 0 ; { število napak }
 sttestov := 0 ;{ število testov }

 repeat
 case stanje of
  { generiranje testnih vzorcev 0,55h, aah,
    ffh }
  0: begin
       generiranje testenega vzorca ;
       stanje := 1 ;
     end ;
  { vpis testnih vzorcev iz RAMa v mehurčni
    pomnilnik }
  1: begin
       vpis testenega vzorca na stran page ;
       if BMCstatus then
         begin
           if page = 4096 then
             begin
               sttestov := sttestov + 4096 ;
               page := 0 ;
               stanje := 2 ;
             end ;
           else page := page + 1 ;
         end ;
       else status := 4 ;
     end ;
  { branje testnih vzorcev iz mehurčnega
    pomnilnika v RAM }
  2: begin
       branje testnega vzorca iz strani page ;
       if BMCstatus then stanje := 3 ;
       else stanje := 4 ;
     end ;
  { primerjava vpisanih in branih testnih
    vzorcev }
  3: begin
       komparacija vpisanega in branega
       testenega vzorca ;
       if razlika then
         begin
           stnapak := stnapak + 1 ;
           write ( stnapak ) ;
         end ;
      if page = 4096 then stanje := 0 ;
      else
        begin
          page := page + 1 ;
          stanje := 2 ;
        end ;
      end ;
  { nasilna prekinitev }
  4: begin
       nasilna prekinitev ;
       programski reset ;
       stanje := 0
     end ;
 end ;
until konec ;
end.
```

UPORABNI PROGRAMI

```
==========================================
=                                        =
=    Prilagodljivo numerično integriranje =
=                                        =
==========================================
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Informatica UP 18                      %
% Adaptive Numerical Integration         %
% oktober 1984                           %
% pripravil Vladimir Batagelj            %
% sistem DEC-10                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

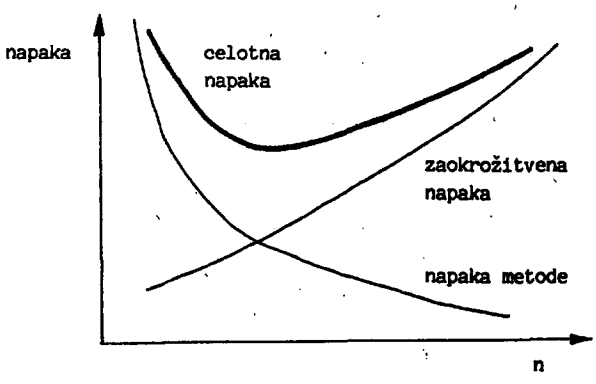
POSTOPKI ZA PRILAGODLJIVO NUMERIČNO INTEGRIRANJE

Vladimir Batagelj
VTOZD Matematika in mehanika
Univerza E. Kardelja v Ljubljani

Namen tega sestavka je opozoriti na postopke za prilagodljivo (adaptivno) numerično integriranje in na vire [2, 4, 5, 6], kjer so podrobneje opisani.

Postopki za prilagodljivo numerično integriranje so področje, na katerem se uspešno prepletata numerična analiza in računalništvo, natančneje, načrtovanje in analiza algoritmov. Tako je na primer v numerični knjižnici NAG podprogram D01AGA/F, ki temelji na Oliverjevi prilagodljivi metodi, priporočen kot najboljši splošni integracijski podprogram.

Če želimo pri običajnih postopkih za numerično integriranje [1] povečati natančnost izračuna, povečujemo število n točk ali vozlov, ki jih upoštevamo pri izračunu. S tem sicer zmanjšamo napako metode, žal pa se nam pri tem zaradi večanja števila operacij večata zaokrožitvena napaka in čas (cena) računanja. Zato se obnaša celotna napaka tako, kot prikazuje slika:



Pri prilagodljivih postopkih pa drobimo interval samo tam, kjer je treba. Zato običajno za dosego željene natačnosti potrebujejo manj korakov - pri (skoraj) isti napaki metode bo manjša zaokrožitvena napaka pa tudi postopek je hitrejši in s tem bolj ekonomičen.

Osnovna zamisel prilagodljivih postopkov numeričnega integriranja je znana v računalništvu kot pristop "deli in vladaj". Recimo, da želimo izračunati vrednost P(a,b,eps) integrala:

$$I(a,b) = \int_a^b f(x)\,dx$$

pri čemer dopuščamo napako eps . Naj bo še S(a,b) ena izmed kvadraturnih formul, ki jih poznamo iz numerične analize [1] , in dajejo oceno za vrednost I(a,b) .

Določimo oceno $S_1 = S(a,b)$ za neznani I(a,b) . Nato razpolovimo interval [a,b] s točko m = (a+b)/2 in izračunamo še oceno $S_2 = S(a,m) + S(m,b)$. Če se oceni razlikujeta za manj kot eps , vzamemo S_2 za P(a,b,eps) . Sicer isti postopek rekurzivno ponovimo na obeh podintervalih [a,m] in [m,b] , pri čemer pa na vsakem podintervalu dopuščamo le pol manjšo napako eps/2 . Ko enkrat poznamo P(a,m,eps/2) in P(m,b,eps/2) , lahko izračunamo tudi

P(a,b,eps) = P(a,m,eps/2) + P(m,b,eps/2)

V praksi se izkaže, da je zmanjšanje dovoljene napake na eps/2 pri rekurzivni ponovitvi postopka preveč pesimistično. Zato v postopek vpeljemo nov parameter q , $1 < q \leq 2$ in dovoljujemo napako eps/q . Izkaže se [3], da je čisto v redu že q = 1.4 .

V tem sestavku je opisani postopek razdelan za primer, ko oceno vrednosti integrala na intervalu [a,b] računamo po Simpsonovem obrazcu:

$$S(a,b) = \frac{b-a}{6}\left(f(a) + 4.f\left(\frac{a+b}{2}\right) + f(b) \right)$$

Treba pa je takoj pripomniti, da podprogrami iz numeričnih knjižnic uporabljajo veliko bolj "zvite" ocene in tudi sam postopek je še nadalje izpopolnjen.

Za primerjavo je dodan še navadni postopek za numerično integriranje po Simpsonu.

V obeh programih se nahajajo spremenljivke, za merjenje posameznih značilnih količin:

- i - zaporedna številka
- countf - število izračunanih funkcijskih vrednosti
- depth - globina rekurzije

Te spremenljivke in stavki, ki jih vsebujejo, ne vplivajo na samo računanje, zato jih lahko tudi izločimo.

Za to, da bodo prednosti prilagodljivega postopka očitne, je bila primerjava narejena na integralu (ploščina četrtine enotskega kroga):

$$\int_0^1 \sqrt{1 - x^2}\,dx$$

ki je za navadni postopek neugoden. Njegova štirikratna vrednost je število pi .

Priloženi tabeli govorita sami zase. Omenimo le pomen posameznih stolpcev:
NAVADNO: zaporedna številka, izračunana vrednost integrala, dejanska napaka, število izračunanih funkcijskih vrednosti;
PRILAGODLJIVO: zaporedna številka, izračunana vrednost integrala, dovoljena napaka, dejanska napaka, število

izračunanih funkcijskih vrednosti, največja globina rekurzije.

Izračuni so bili opravljeni na računalniku DEC-10 , Univerze v Ljubljani.

```
PROGRAM simpson(output);
CONST pi = 3.141592653589793 ;
      errmin = 0.00000005 ;
VAR error, int : real; i, n, countf : integer;
FUNCTION f( x: real ): real ;
BEGIN
   countf := countf + 1 ;
   f := sqrt(abs(1-sqr(x)))
END (* f *) ;
FUNCTION integral
      ( FUNCTION f:real; a,b:real; n:integer ): real;
VAR  h, p, x : real ; c, i : integer ;
BEGIN
   h := (b - a)/(2*n) ; p := f(a) + f(b);
   x := a ; c := 4 ;
   FOR  i := 1  TO  2*n-1  DO BEGIN
      x := x + h ;
      p := p + c*f(x) ;
      c := 6 - c
   END;
   integral := h*p/3
END (* integral *) ;
BEGIN
   writeln(' ':5,'NAVADNO INTEGRIRANJE PO SIMPSONU ');
   writeln;
   n := 1; i := 0 ;
   REPEAT
      countf := 0 ; i := i + 1 ;
      int := 4*integral(f,0,1,n) ;
      error := pi - int;
      writeln( i:7, int, error, countf:8 );
      n := 2*n
   UNTIL abs( error ) <= errmin
END.
```

```
   NAVADNO INTEGRIRANJE PO SIMPSONU

    1   2.976068E+00   1.655249E-01      3
    2   3.083595E+00   5.799752E-02      5
    3   3.121189E+00   2.040347E-02      9
    4   3.134398E+00   7.194966E-03     17
    5   3.139052E+00   2.540439E-03     33
    6   3.140695E+00   8.975565E-04     65
    7   3.141276E+00   3.171861E-04    129
    8   3.141481E+00   1.121163E-04    257
    9   3.141553E+00   3.978610E-05    513
   10   3.141579E+00   1.385808E-05   1025
   11   3.141588E+00   4.917383E-06   2049
   12   3.141591E+00   1.758337E-06   4097
   13   3.141592E+00   5.662441E-07   8193
   14   3.141592E+00   2.980232E-07  16385
   15   3.141592E+00   4.172325E-07  32769
   16   3.141592E+00   5.066395E-07  65537
   17   3.141593E+00  -8.940697E-08 131073
   18   3.141591E+00   1.817942E-06 262145
   19   3.141593E+00  -8.940697E-08 524289
   20   3.141601E+00  -8.434057E-06 1048577
```

```
PROGRAM simpson(output);
CONST pi = 3.141592653589793 ;
      errmin = 0.00000001 ;
VAR   error, int : real; i, countf, depth : integer;
FUNCTION f( x: real ): real ;
BEGIN
   countf := countf + 1 ;
   f := sqrt(abs(1-sqr(x)))
END (* f *) ;
FUNCTION integral
   ( FUNCTION f:real; a,b,error:real ): real;
VAR   m, c, fa, fm, fb : real ; d : integer ;
FUNCTION irec
   ( a, m, b, fa, fm, fb, estimate, error:real ): real;
CONST q = 1.5 ;
VAR   c, ma, mb, fma, fmb, esta, estb, newest : real ;
BEGIN
   d := d + 1 ; IF d > depth THEN depth := d ;
   ma := (a + m)/2 ; fma := f(ma) ;
   mb := (m + b)/2 ; fmb := f(mb) ;
   c  := (b - a)/12 ;
   esta := c * ( fa + 4*fma + fm ) ;
   estb := c * ( fm + 4*fmb + fb ) ;
   newest := esta + estb ;
   IF abs(estimate-newest) > error THEN
      irec := irec(a,ma,m,fa,fma,fm,esta,error/q) +
              irec(m,mb,b,fm,fmb,fb,estb,error/q)
   ELSE
      irec := newest ;
   d := d - 1
END (* irec *) ;
BEGIN
   d := 0 ; m := (a + b)/2 ; c := (b - a)/6 ;
   fa := f(a) ; fm := f(m) ; fb := f(b) ;
   integral := irec(a,m,b,fa,fm,fb,c*(fa+4*fm+fb),error)
END (* integral *) ;
BEGIN
   write(' ':15);
   writeln('PRILAGODLJIVO INTEGRIRANJE PO SIMPSONU');
   writeln ; error := 1 ; i := 0 ;
   WHILE error > errmin DO BEGIN
      countf := 0 ; depth := 0 ; i := i + 1 ;
      int := 4*integral(f,0,1,error) ;
      writeln(i,int,error,pi-int,countf:6,depth:6);
      error := error / 2
   END
END.
```

```
PRILAGODLJIVO INTEGRIRANJE PO SIMPSONU

 1  3.083595E+00  1.000000E+00  5.799752E-02    5   1
 2  3.083595E+00  5.000000E-01  5.799752E-02    5   1
 3  3.083595E+00  2.500000E-01  5.799752E-02    5   1
 4  3.083595E+00  1.250000E-01  5.799752E-02    5   1
 5  3.083595E+00  6.250000E-02  5.799752E-02    5   1
 6  3.083595E+00  3.125000E-02  5.799752E-02    5   1
 7  3.121189E+00  1.562500E-02  2.040350E-02    9   2
 8  3.134383E+00  7.812500E-03  7.209718E-03   13   3
 9  3.139032E+00  3.906250E-03  2.560467E-03   17   4
10  3.141253E+00  1.953125E-03  3.397763E-04   25   6
11  3.141458E+00  9.765625E-04  1.348853E-04   29   7
12  3.141530E+00  4.882813E-04  6.246567E-05   33   8
13  3.141556E+00  2.441406E-04  3.686547E-05   37   9
14  3.141565E+00  1.220703E-04  2.783537E-05   41  10
15  3.141583E+00  6.103516E-05  9.894371E-06   49  11
16  3.141588E+00  3.051758E-05  4.410744E-06   57  12
17  3.141590E+00  1.525879E-05  2.533197E-06   65  13
18  3.141591E+00  7.629395E-06  1.847744E-06   73  14
19  3.141592E+00  3.814697E-06  9.238720E-07   85  15
20  3.141592E+00  1.907349E-06  3.576279E-07  105  17
21  3.141592E+00  9.536743E-07  2.682209E-07  117  18
22  3.141593E+00  4.768372E-07  1.490116E-07  133  19
23  3.141593E+00  2.384186E-07  8.940697E-08  153  20
24  3.141593E+00  1.192093E-07  2.980232E-08  185  21
25  3.141593E+00  5.960464E-08  0.000000E+00  213  22
26  3.141593E+00  2.980232E-08  0.000000E+00  253  23
27  3.141593E+00  1.490116E-08  0.000000E+00  285  24
```

LITERATURA:

[1] Bohte Z.: Numerične metode. (v I. Vidav: Višja matematika III). Ljubljana, 1973

[2] De Boor C.: CADRE: an algorithm for numerical quadrature. (v Mathematical software, J.R.Rice, ed.). Academic Press, New York, 417-449

[3] Dennis J.E. Jr., More J.J.: Computer solution of mathematical problems. (v Conway R., Gries D.: An Introduction to Programming). Winthrop, Cambridge, Mass., 1975, 358-366

[4] Kahaner D.K.: Comparision of numerical quadrature formulas. (v Mathematical software, J.R.Rice, ed.). Academic Press, New York, 229-259

[5] NAG fortran library manual, mark 6. NAG Ltd., 1977

[6] Oliver J.: A doubly-adaptive Clenshaw-Curtis quadrature method. The Computer Journal, 15(1972), 141-147

Vsa literatura je dosegljiva v Matematični knjižnici, Jadranska 19, Ljubljana.

```
==============================================
=                                            =
=    MAGIČNI KVADRATI SODE IN LIHE STOPNJE   =
=                                            =
==============================================


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                             %
%  Informatica UP 19                          %
%  Magic Squares of Even and Odd Degree       %
%  september 1984                             %
%  priredil Anton P. Železnikar               %
%  sistem CP-M, Delta Partner                 %
%  prevajalnik Janus-Ada, verzija 1.5.0       %
%                                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

1. Področje uporabe

Problematika magičnih kvadratov sodi v področje matematične in programirne rekreacije. Tu si je mogoče izmišljati različne algoritme za določevanje magičnih kvadratov sode in lihe stopnje.

2. Opis programa

Procedura magiceven v listi 1 določa za kvadratno matriko n krat n elemente z vrednostmi od 1 do n krat n in jih razmešča v leksikografskem zaporedju v polje x(1 .. n; 1 .. n), ko je n sodo celo število, ki ni manjše od 4 in oblikuje na ta način tkim. magični kvadrat.

V magičnem kvadratu so vsote elementov poljubne vrstice, poljubnega stolpca in poljubne diagonale medseboj enake.

Procedura magicodd v listi 1 oblikuje magični kvadrat podobno kot procedura magiceven, le da je stopnja n matrike x liha in ni manjša od 3.

Funkcija magicterm v listi 1 izračuna element s(i,j) magičnega kvadrata s(1 .. n, 1 .. n) za liho vrednost n, ki ni manjša od 3.

Lista 1 vsebuje še procedur za izpis magičnega kvadrata kvadr_izpis in proceduro za preizkušanje magičnosti kvadrata magic_test, ki najprej določi magično vsoto in nato preizkusi na to vsoto vse vrstice, vse stolpce in vse diagonale.

Z glavnim preizkusnim programom liste 1 se uporabijo zadevne procedure in funkcija (magiceven, magicodd in magicterm) z izpisom magičnih matrik in njihovim testiranjem.

3. Izvajanje programa

Lista 2 prikazuje izvajanje programa. Magični kvadrata 1 in 2 sta posledici uporabe procedure magiceven za n = 4 in za n = 6. Magična kvadrata 3 in 4 nastaneta z uporabo procedure magicodd za n = 3 in za n = 5. Magični kvadrat 5 se pojavi z uporabo funkcije magicterm za n = 5. Magični kvadrat 6 je rezultat uporabe procedure magiceven za n = 16 in magični kvadrat 7 je rezultat uporabe procedure magicodd za n = 17.

```
------------------------------------------------
--        Masični kvadrati sode in            --
--              in lihe stopnje               --
------------------------------------------------

WITH util;

PACKAGE BODY magici IS

  n: CONSTANT := 17;
  SUBTYPE d IS integer RANGE 1 .. n;
  TYPE stolpec IS ARRAY (d) OF integer;
  TYPE polje IS ARRAY (d) OF stolpec;

--  Naslednja procedura je predmet naše  pozor-
--     nosti (magični kvadrat sode stopnje)
--*********************************************

  PROCEDURE magiceven ( n: IN integer;
                        x: OUT polje ) IS

    a, b, n2, nn, i: integer;
    p, q, r: Boolean;

    PROCEDURE alpha ( p, q, a: IN integer;
                      h: IN Boolean ) IS
      hh: Boolean;
      BEGIN
        hh := h;
        FOR i IN p .. q LOOP
          hh := NOT hh;
          IF hh THEN
            x(i)(a) := nn - a*n + 1 + n - i;
          ELSE
            x(i)(a) := a*n - n + i;
          END IF;
        END LOOP;
      END alpha;
```

Lista 1. Procedure in programi za generiranje magičnih kvadratov (nadaljevanje na drugi str.)

```
    PROCEDURE beta ( p, q, a: IN integer;
                     h: IN Boolean ) IS
      hh: Boolean;
      BEGIN
        hh := h;
        FOR i IN p .. q LOOP
          hh := NOT hh;
          IF hh THEN
            x(i)(a) := a*n + 1 - i;
          ELSE
            x(i)(a) := nn - a*n + i;
          END IF;
        END LOOP;
      END beta;

    PROCEDURE gamma ( p, q, a: IN integer;
                      h: IN Boolean ) IS
      hh: Boolean;
      BEGIN
        hh := h;
        FOR i IN p .. q LOOP
          hh := NOT hh;
          IF hh THEN
            x(i)(a) := a*n + 1 - i;
          ELSE
            x(i)(a) := nn - a*n + n - i + 1;
          END IF;
        END LOOP;
      END gamma;
    ------------------------------------------------

    BEGIN
<<start>>
      n2 := n/2; nn := n*n;
      p := (n = (n/4)*4); q := p; r := true;
      FOR a IN 1 .. n2-1 LOOP
        beta(1,a-1,a,r);
        alpha(a,n2-1,a,true);
        IF q THEN
          x(n2)(a) := nn - a*n + n2 + 1;
        ELSE
          x(n2)(a) := nn - a*n + n2;
        END IF;
        alpha(n2+1,n,a,NOT q);
        q := NOT q; r := NOT r;
      END LOOP;
      alpha(1,n2-1,n2,NOT p);
      alpha(n2+2,n,n2,false);
      gamma(1,n2-1,n2+1,p);
      gamma(n2+2,n,n2+1,true);
      q := p; r := true;
      FOR a IN n2+2 .. n LOOP
        beta(1,n-a,a,q);
        x(n-a+1)(a) := a*n - a + 1;
        beta(n-a+2,n2-1,a,true);
        IF NOT r THEN
          x(n2)(a) := nn - a*n + n2;
          x(n2+1)(a) := a*n -n2 +1;
        ELSE
          FOR b IN n2 .. n2+1 LOOP
            x(b)(a) := nn - a*n + n - b + 1;
          END LOOP;
        END IF;
        beta(n2+2,a-1,a,NOT r);
        alpha(a,n,a,true);
        q := NOT q; r := NOT r;
      END LOOP;
      FOR a IN n2 .. n2+1 LOOP
        FOR b IN n2 .. n2+1 LOOP
          IF p THEN
            x(b)(a) := a*n - n + b;
          ELSE
            x(b)(a) := nn - a*n +n - b +1;
          END IF;
        END LOOP;
      END LOOP;
      IF NOT p THEN
        FOR a IN n2 .. n2+1 LOOP
          x(n2-1)(a) := a*n - n2 + 2;
        END LOOP;
        FOR b IN n2 .. n2+1 LOOP
          x(b)(n2+2) := n*n2 - 2*n + b;
        END LOOP;
      END IF;
    END magiceven;
--*********************************************

--  Tudi ta procedura je predmet naše  pozorno-
--    sti (magični kvadrat lihe stopnje)
--*********************************************

  PROCEDURE magicodd ( n: IN integer;
                       x: OUT polje ) IS

    i, j, k: integer;

    BEGIN
      FOR i IN 1 .. n LOOP
        FOR j IN 1 .. n LOOP
          x(i)(j) := 0;
        END LOOP;
      END LOOP;
      i := (n + 1)/2; j := n;
      FOR k IN 1 .. n*n LOOP
        IF x(i)(j) /= 0 THEN
          i := i - 1; j := j - 2;
          IF i < 1 THEN i := i + n; END IF;
          IF j < 1 THEN j := j + n; END IF;
        END IF;
        x(i)(j) := k; i := i + 1;
        IF i > n THEN i := i - n; END IF;
        j := j + 1;
        IF j > n THEN j := j - n; END IF;
      END LOOP;
    END magicodd;
--*********************************************

--  Tudi ta funkcija je predmet naše pozornosti
--    (magični člen za kvadrat lihe stopnje)
--*********************************************

  FUNCTION magicterm ( i, j, n: IN integer )
                       RETURN integer  IS
    b, c: integer;
    BEGIN
      b := j - i + (n - 1)/2; c := 2*j - i;
      IF b >= n THEN
        b := b - n;
      ELSIF b < 0 THEN
        b := b + n;
      END IF;
      IF c > n THEN
        c := c - n;
      ELSIF c <= 0 THEN
        c := c + n;
      END IF;
      RETURN  b*n + c;
    END magicterm;
--*********************************************

  PROCEDURE kvadr_izpis (i, n: IN integer;
                         x: IN polje) IS
    k, p: integer;
    BEGIN
      new_line;
      put("Magični kvadrat");
      put(i,2); put(" = ");
      new_line; put("--------------------");
      new_line;
      FOR k IN 1 .. n LOOP
        FOR p IN 1 .. n LOOP
          put(x(k)(p),5);
          IF p = n THEN new_line; END IF;
        END LOOP;
      END LOOP;
      new_line;
    END kvadr_izpis;
-------------------------------------------------
```

Lista 1. (Nadaljevanje s prejšnje in na naslednjo stran). Bistvene procedure te liste so magiceven, magicodd in magicterm, pomožni (oziroma preizkusni) proceduri pa sta kvadr_izpis in magic_test.

```
PROCEDURE magic_test ( i, n: IN integer;
                       x: polje) IS
  aa, bb, c, d, e: integer;
  BEGIN
    aa := 0;
  ---------- Določitev magične vsote ---------
    FOR c IN 1 .. n LOOP
      aa := aa + x(1)(c);
    END LOOP;
  ---------- Preizkus vrstičnih vsot --------
    FOR c IN 1 .. n LOOP
      bb := 0;
      FOR d IN 1 .. n LOOP
        bb := bb + x(c)(d);
      END LOOP;
      IF bb /= aa THEN
        put("Kvadrat ni magičen!");
        new_line;
        GOTO fin;
      END IF;
    END LOOP;
---------- Preizkus stolpnih vsot ---------
    FOR c IN 1 .. n LOOP
      bb := 0;
      FOR d IN 1 .. n LOOP
        bb := bb + x(d)(c);
      END LOOP;
      IF bb /= aa THEN
        put("Kvadrat ni magičen!");
        new_line;
        GOTO fin;
      END IF;
    END LOOP;
---------- Preizkus diagonalnih vsot -------
    bb := 0; e := 0;
    FOR d IN 1 .. n LOOP
      bb := bb + x(d)(d);
      e := e + x(d)(n-d+1);
    END LOOP;
    IF bb /= aa OR e /= aa THEN
      put("Kvadrat ni magičen!");
      new_line;
      GOTO fin;
    END IF;
    put("Magična vsota kvadrata");put(i,2);
    put(" je ");put(aa,5);new_line;
<<fin>> null;
  END magic_test;
-----------------------------------------------

--------- Glavni preizkusni program ---------

  x: polje; r, s: integer;

  BEGIN
    magiceven (4, x);
    kvadr_izpis (1, 4, x);
    magic_test (1, 4, x);

    magiceven (6, x);
    kvadr_izpis (2, 6, x);
    magic_test (2, 6, x);

    magicodd (3, x);
    kvadr_izpis (3, 3, x);
    magic_test (3, 3, x);

    magicodd (5, x);
    kvadr_izpis (4, 5, x);
    magic_test (4, 5, x);

    FOR r IN 1 .. 5 LOOP
      FOR s IN 1 .. 5 LOOP
        x(r)(s) := magicterm(s,r,5);
      END LOOP;
    END LOOP;
    kvadr_izpis (5, 5, x);
    magic_test (5, 5, x);

    magiceven (16, x);
    kvadr_izpis (6, 16, x);
    magic_test (6, 16, x);

    magicodd (17, x);
    kvadr_izpis (7, 17, x);
    magic_test (7, 17, x);

  END magic1;
-----------------------------------------------
```

Lista 1. (Nadaljevanje s prejšnjih dveh strani) Glavni preizkusni program kliče deklarirane procedure in funkcijo magicterm in izpiše rezultate v obliki liste 2.

```
Magični kvadrat 1 =
--------------------
   1   12    8   13
  15    6   10    3
  14    7   11    2
   4    9    5   16

Magična vsota kvadrata 1 je    34

Magični kvadrat 2 =
--------------------
   1   12   13   24   30   31
  35    8   17   23   26    2
  33   28   22   16    9    3
   4   27   21   15   10   34
  32   11   20   14   29    5
   6   25   18   19    7   36

Magična vsota kvadrata 2 je   111

Magični kvadrat 3 =
--------------------
   4    3    8
   9    5    1
   2    7    6

Magična vsota kvadrata 3 je    15
```

Lista 2. Ta rezultatna lista, ki se nadaljuje na naslednji strani, prikazuje prvih pet magičnih kvadratov, ko imamo n = 4, 6, 3, 5, 5. Četrti in peti kvadrat sta dobljena z različnima procedurama in rabita za primerjavo. Ob vsakem kvadratu se izpiše tudi magična vsota.

```
Magični kvadrat 4 =
--------------------
  11   10    4   23   17
  18   12    6    5   24
  25   19   13    7    1
   2   21   20   14    8
   9    3   22   16   15

Magična vsota kvadrata 4 je    65

Magični kvadrat 5 =
--------------------
  11   10    4   23   17
  18   12    6    5   24
  25   19   13    7    1
   2   21   20   14    8
   9    3   22   16   15

Magična vsota kvadrata 5 je    65
```

Magični kvadrat 6 =

1	32	209	64	177	96	145	144	128	97	176	65	208	33	240	241
255	18	47	194	79	162	111	114	143	159	82	191	50	223	226	2
3	238	35	62	179	94	147	142	126	99	174	67	206	211	19	254
253	20	221	52	77	164	109	116	141	157	84	189	196	36	237	4
5	236	37	204	69	92	149	140	124	101	172	181	53	220	21	252
251	22	219	54	187	86	107	118	139	155	166	70	203	38	235	6
7	234	39	202	71	170	103	138	122	151	87	186	55	218	23	250
249	232	217	200	185	168	153	120	136	105	88	73	56	41	24	9
248	25	216	57	184	89	152	121	137	104	169	72	201	40	233	8
10	231	42	199	74	167	106	135	119	154	90	183	58	215	26	247
246	27	214	59	182	91	150	123	134	102	171	75	198	43	230	11
12	229	44	197	76	165	108	133	117	156	85	188	60	213	28	245
244	29	212	61	180	93	148	125	132	100	173	68	205	45	228	13
14	227	46	195	78	163	110	131	115	158	83	190	51	222	30	243
242	31	210	63	178	95	146	127	130	98	175	66	207	34	239	15
16	225	48	193	80	161	112	129	113	160	81	192	49	224	17	256

Magična vsota kvadrata 6 je 2056

Magični kvadrat 7 =

137	136	118	100	82	64	46	28	10	281	263	245	227	209	191	173	155
156	138	120	119	101	83	65	47	29	11	282	264	246	228	210	192	174
175	157	139	121	103	102	84	66	48	30	12	283	265	247	229	211	193
194	176	158	140	122	104	86	85	67	49	31	13	284	266	248	230	212
213	195	177	159	141	123	105	87	69	68	50	32	14	285	267	249	231
232	214	196	178	160	142	124	106	88	70	52	51	33	15	286	268	250
251	233	215	197	179	161	143	125	107	89	71	53	35	34	16	287	269
270	252	234	216	198	180	162	144	126	108	90	72	54	36	18	17	288
289	271	253	235	217	199	181	163	145	127	109	91	73	55	37	19	1
2	273	272	254	236	218	200	182	164	146	128	110	92	74	56	38	20
21	3	274	256	255	237	219	201	183	165	147	129	111	93	75	57	39
40	22	4	275	257	239	238	220	202	184	166	148	130	112	94	76	58
59	41	23	5	276	258	240	222	221	203	185	167	149	131	113	95	77
78	60	42	24	6	277	259	241	223	205	204	186	168	150	132	114	96
97	79	61	43	25	7	278	260	242	224	206	188	187	169	151	133	115
116	98	80	62	44	26	8	279	261	243	225	207	189	171	170	152	134
135	117	99	81	63	45	27	9	280	262	244	226	208	190	172	154	153

Magična vsota kvadrata 7 je 2465

Lista 2. (Nadaljevanje s prejšnje strani) Ta del liste prikazuje še primera magičnih kvadratov za n = 16, 17. Iz teh kvadratov je razvidna metoda razmeščanja elementov kot zaporednih vrednosti od 1 do n X n. Ta razmeščevalna metoda je različna za sodi in lihi primer.

Jezik Lisp za mikroračunalnike

Podjetje Microsoft je napovedalo prvo izvedenko jezika Lisp za svoj operacijski sistem MS-DOS. To pa pomeni, da bo mogoče probleme umetne inteligence (področje izvedenskih sistemov in prevajanje naravnih jezikov) reševati tudi z mikroračunalniki, kot so IBM PC in njemu podobni. Novi paket bo imel oznako

muLisp-82
Artificial Intelligence Development System

in bo naslednik paketa muLisp-80 za osembitne mikroračunalnike.

Podobno kot so omejene obstoječe izvedenke za uporabo jezika Prolog na mikroračunalnikih v okviru problematike umetne inteligence, tako bo omejena tudi izvedenka muLisp-82 v svoji zmogljivosti. Tako bo muLisp-82 v glavnem zanimiv za univerze in raziskovalna podjetja. Vendar se bo z uporabo tega jezika lahko znatno razširila metodologija umetne inteligence med strokovnjaki in amaterji.

Podjetje Microsoft se je dokončno odločilo, da začne osvajati tržišče umetne inteligence s svojimi programskimi izdelki. Zato se dogovarja z drugimi podjetji za izdelavo in pravice na področju razpredelničnih pol pri razvoju izvedenskih sistemov (paket ExpertEase podjetja Export Software). Paket ExpertEase je programski generator za gradnjo malih izvedenskih sistemov na 16-bitnih mikroračunalnikih. Ta paket naj bi bil namenjen predvsem poslovnežem in manj akademikom, torej tistim, ki delajo na področju finančnega inženiringa in finančnega poslovanja.

Paket ExpertEasy je mogoče uporabiti na ibmovskih osebnih računalnikih, njegova cena pa je 1500 funtov. Podjetje Export Software bo kmalu izdalo demonstracijski paket s ceno 100 funtov. Pri vsem tem pa je bilo precej dvomov, da bi se tak paket lahko uporabljal na računalnikih tipa IBM PC.

A.P.Železnikar

Kaj pomeni ibmovski osebni računalnik ?

Kdaj smemo reči, da je osebni računalnik ibmovski, da je podoben osebnemu računalniku IBM PC, da ima združljivost z IBM PC? Katera so minimalna merila, ki to združljivost zagotavljajo (pravzaprav približujejo računalnik ibmovskemu)?

S pojavom računalnikov tipa IBM PC in IBM XT je nastal industrijski standard, ki se mu poskušajo prilagoditi številni proizvajalci (posnemovalci ibmovskih sistemov). V primerih približevanja standardom se vselej vprašujemo, kako in v kakšni meri so posnetki združljivi z IBM PC. Odgovor na to vprašanje je določena materialna in programska oprema, ki obstaja za določen ibmovski posnetek. Podjetje Award Software v ZDA je razvilo posebne programe, s katerimi se ugotavlja stopnja združljivosti z IBM PC.

Slika 1 prikazuje arhitektonski model za IBM PC in IBM XT. Ta model je sestavljen iz štirih ravnin:

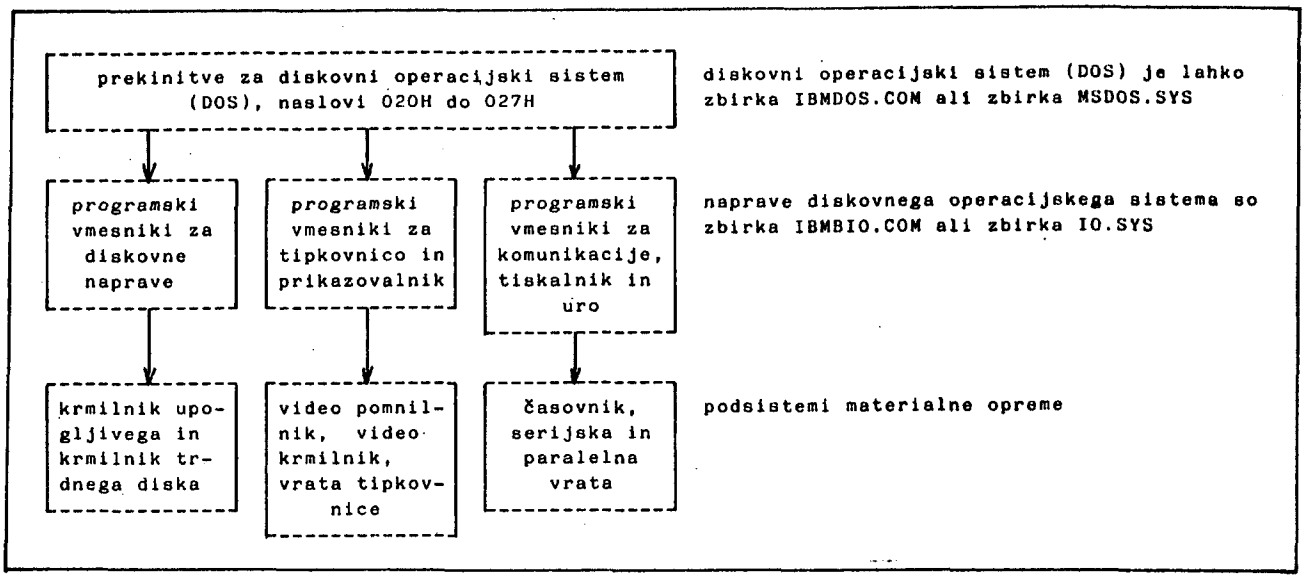
-- materialne opreme
-- nižjih podpornih rutin (ROM BIOS)
-- programskih vmesnikov za VI naprave sistema PC-DOS (ibmov BIO.COM)
-- operacijskega sistema PC-DOS (ibmov (DOS.COM)

Če je materialna (aparaturna) oprema popolnoma ibmovsko združljiva, je mogoče na računalniku uporabljati ibmove ROMe in ibmov DOS. Vendar je z avtorskimi pravicami podjetja IBM ta metoda prepovedana in posnemovalci ibmovskih osebnih računalnikov morajo imeti svoj ROM BIOS (osnovni VI sistem v ROMu), ki se mora dovolj razlikovati od ibmovega, hkrati pa mora zagotavljati identično povezavo kot ibmov ROM BIOS. Tako je mogoče implementirati modul IO.SYS za operacijski sistem MS-DOS, ki pa mora biti dovolj različen in mora zagotavljati povezave, ki so identične za PC-DOS.

Tri področja združljivosti so tako bistvena: materialna oprema, ROM BIOS in povezave za PC-DOS. Materialna oprema mora biti sestavljena iz določenih komponent (integrirana vezja 8088, 8259, 8255, NEC 765 itd.), VI vrat in povezav med integriranimi vezji. ROM BIOS združuje osnovne programske rutine za VI in organizira materialno opremo v ustrezno upravljano obliko. ROM BIOS za IBM PC in za IBM XT je enoopravilni in enouporabniški pripomoček za VI. Čeprav se uporablja prekinitveni mehanizem, se hkrati izvaja en sam proces (edina izjema se pojavi pri časovniški prekinitvi). PC-DOS je ibmova izvedenka Microsoftovega MS-DOSa.

Na trgu je vrsta mikroračunalnikov, ki imajo izredno dobro materialno in programsko združljivost z IBM PC. Ti proizvodi so:

Columbia
Compaq
Corona
Eagle
Mitsubishi
Stearns
Seequa



Slika 1. Arhitektonski model osebnega računalnika IBM PC in IBM XT

Tabela 1. Glavne materialne komponente osebnega računalnika IBM PC

Integrirano vezje	VI naslovi	Uporaba
Intel 8088, 8086		Osrednji procesor
NEC 765	3F4H, 3F5H	Krmilnik upogljivega diska
Vmesnik 74LS	3F2H	Sekundarno krmiljenje upogljivega diska
Intel 8255	60H do 63H	VI vrata za zvočnik, tipkovnico, za konfiguriranje računalnika, za usposobitev preizkusa parnosti RAMa
Intel 8253	40H do 43H	Ura realnega časa, časovnika zvočnika, zahteva za DMA pomnilniško osveževanje
Intel 8237	00H do 0FH	DMA krmilnik
Vmesniki 74LS	80H do 83H	Registri strani pri DMA
Intel 8259	20H, 21H	Prekinitveni krmilnik
Vmesnik 74LS	0A0H do 0AFH	Krmiljenje NMI
Trdni disk	320H do 32FH	Krmilnik trdnega diska (vinčestrski disk)
Intel 8255	378H do 37AH	Tiskalniški krmilniki
	3BCH do 3BEH	
	278H do 27AH	
National 8250	2F8H do 2FEH	Serijsko komunikacijsko vezje
	3F8H do 3FEH	
Vmesniki za Motorola 6845	3D0H do 3DFH	Barvnigrafični krmilnik
Vmesniki za Motorola 6845	3B0H do 3BFH	Krmilnik za monokromatično prikazovanje

Preizkus materialne združljivosti

Pri ugotavljanju združljivosti moramo preizkusiti, kakšna so uporabljena integrirana vezja in uporabljeni naslovi VI vrat. Tabela 1 prikazuje podatke za IBM PC in IBM XT.

Materialna združljivost z IBM PC zahteva, da obstajata monokromatični in grafični pomnilnik na ustreznih naslovih in da imata znakovni in atributski (pixel) pomnilnik identično organizacijo. Pri tem je potrebno preveriti operacije iz integriranih vezij, tako da je ustrezno integrirano vezje prisotno na ustreznem VI naslovu. Prikazovalni pomnilnik se preizkusi tako, da se ustrezni vzorec vstavi neposredno v video pomnilnik in se opazuje.

Kadar želimo uporabljati prekinitveno delujoče operacijske sisteme za IBM PC, kot so npr. ONX, Concurrent CPM-86 itd., mora biti gornji preizkus veljaven, ker je sicer potrebno dobiti posebne izvedbe teh operacijskih sistemov.

VI (sistemsko) vodilo in časovne oblike na tem vodilu se lahko preizkusijo z uporabo komercialno dobavljivih vtičnih plošč, ko se ugotavlja njihovo delovanje. Razen tega bi lahko opazovali časovne signalne slike tudi z logičnim analizatorjem ter jih primerjali s signalnimi slikami na IBM PC ali IBM XT.

Združljivost ROM BIOSa

V ROM BIOSu imamo dva tipa povezav. Najbolj pogost način povezave (in tudi najboljši) je z uporabo programskih prekinitev. Drugi tip povezave je z neposrednim naslavljanjem tistega dela RAMa, ki ga uporablja ROM BIOS (to pa ni priporočljivo).

Z izključno uporabo programskih prekinitev dobimo računalni sistem, ki bo pravilno deloval za zelo širok spekter programske opreme. Veliko število posnemovalnih računalnikov za IBM PC deluje na ta način uspešno celo pri drugačni VI organizaciji in z drugačnimi integriranimi vezji, kot jih ima IBM PC.

Tabela 2 vsebuje seznam programskih prekinitev, ki so predvidene za ROM BIOS v IBM PC. Te programske prekinitve so v registre posredovani parametri ali pa se uporabijo prekinitveni vektorji za druge parametre.

Glavne funkcije v ROM BIOSu, ki jih uporabljajo programski paketi, so prekinitve za video, disk, tiskalnik, komunikacije, tipkovnico, časovnik in za druge naprave. Te povezave so opisane v dokumentaciji IBM Personal Computer in v XT Technical Reference Manual (tam je lista za ROM BIOS v dodatku A).

Video prekinitev (heksadecimalno 10) razpolaga s 15 funkcijami, ki so namenjene zaslonskemu prikazovanju, naslavljanju kurzorja, povratni povezavi svetlobnega peresa, pomikanju, teleprinterskemu izhodu, znakovnemu in atributivnemu branjupisanju, grafičnemu branjupisanju in paletni in barvni izbiri. Večina proizvajalcev posnetkov IBM PC je te probleme zadovoljivo razrešila.

Diskovna prekinitev (heksadecimalno 13) razpolaga s 5 funkcijami za upogljive diske in s 14 funkcijami za trdne diske. Funkcije od 9 do 14 za trdne diske uporablja IBM v proizvodnji in pri diagnostičnemu preizkušanju, tako da te funkcije niso vključene v splošne programske pakete. Prekinitve za upogljive in trdne diske uporabljajo parametrične tabele pri prekinitvenih vektorjih 1EH in 41H. To so kazalci v tabele, ki se nahajajo v ROMu ali v RAMu in kjer se nahajajo podatki za pogon motorjev, o številu sektorjev na stezi, številu stez na disku itd.

Bistvena nezdružljivost računalnika z IBM PC lahko izvira iz povečanih pomnilnih obsegov upogljivih diskov (v primerjavi z IBM PC). Taki sistemi imajo svoje časovne oblike in odgovarjajo drugače kot IBM PC; zato se npr. lahko zgodi, da preneha delovati ibmov zaščitni mehanizem za kopiranje zbirk, ki velja za IBM PC. Drugo bistveno področje nezdružljivosti je sporočanje napak. Kopirno zaščitni mehanizmi povzročajo kode napak, ki se sporočajo na IBM PC, na drugih sistemih pa so ta sporočila drugačna.

Tiskalniška prekinitev (heksadecimalno 17) razpolaga s 3 funkcijami za tri paralelna vrata. Komunikacijska prekinitev (14H) ima 4 funkcije za dvoje serijskih komunikacijskih vrat.

Tabela 2. Uporaba prekinitev za ROM BIOS pri računalniku IBM PC

prekinitev	uporaba
0	Deljenje z ničlo iz 80888086
1	Enojni korak (pri DEBUG)
2	Nemaskirna parnostna napaka
3	Točka prekinitve (pri DEBUG)
4	Prestop
5	Pisanje na zaslon
6-7	Rezervirano
8-0FH	Prekinitev iz 8259
10H	Video prekinitev, 15 funkcij
11H	Preslikava za naprave
12H	Obseg pomnilnika
13H	Diskovna prekinitev, 16 funkcij za upogljivi in 14 funkcij za trdni disk
14H	Komunikacijska prekinitev, 4 funkcije
15H	Kaseta
16H	Tipkovnična prekinitev, 3 funkcije
17H	Tiskalniška prekinitev, 3 funkcije
18H	Osnovni vstop v ROM
19H	Navezovalni nalagalnik
1AH	Prekinitev časa dneva, 2 funkciji
1BH	Prekinitev zaradi prekinjene tipkovnice
1CH	Časovniška prekinitev (bitje)
1DH	Kazalec v parametričen seznam za video inicializacijo
1EH	Kazalec v seznam diskovnih parametrov
1FH	Kazalec za grafični znakovni vzorec
20H-3FH	Prekinitve DOS
40H	Preusmeritev diskovne prekinitve za sisteme s trdnimi diski
41H	Kazalec v seznam parametrov za trdni disk
42H-FFH	Rezervirano ali uporabljeno za prekinitve DOSa, Basica ali za uporabniške prekinitve

Področje z največjim odstopanjem mikroračunalnikov od IBM PC je prekinitev za obdelavo tipkovnice (16H). Tu so pomembna notranja pomikalna stanja in vrnjeni kodi. Prekinitve naprav (11H in 12H) sporočajo tip naprave in obseg razpoložljivega pomnilnika.

Časovniške prekinitve (1CH) nastajajo zaradi bitja ure (tiktakanje). Drugi prekinitveni vektorji se uporabljajo v povezavi s krmilnikom 8259, potem so tu še prekinitve zaradi pasti, za enojni korak (po ukazih), pri deljenju z ničlo in za zaslonski zapis.

Preizkus združljivosti s PC-DOS

Večina programskih paketov, ki ne naslavlja neposredno zaslona ali ne uporablja ANSI izogibovalnih (escape) zaporedij za naslavljanje zaslona, uporablja standardne funkcijske klice za

MS-DOS. Če paket ne uporablja funkcijskih tipk, se verjetno lahko izvaja na vsakem MS-DOS stroju (računalniku). Vendar večina paketov za IBM PC potrebuje zaslonsko naslavljanje in kode funkcijskih tipk. Ti paketi uporabljajo video programsko prekinitev kot najnižjo obliko povezave (s preizkusom naprave) in uporabljajo standardne MS-DOS klice. Če posnemovalni računalnik podpira video povezavo, znakovno zaslonsko abecedo in kode funkcijskih tipk, potem bo lahko izvajano na posnemovalnem računalniku veliko število programskih paketov za IBM PC.

Potencialno področje nezdružljivosti je lahko nesposobnost branja in zapisovanja na upogljive diske z ibmovim formatom. Ti formati imajo 48 stez na colo (mera je tpi in pomeni tracks per inch), so eno- in dvostranski in z osmimi in devetimi sektorji na stezi. Ta nezdružljivost se lahko pojavi zaradi diskov s 96 tpi, ko ni mogoče zapisovati na diske z 48 tpi ali zaradi nesposobnosti branja 8-sektorskega formata.

Nadaljna nezdružljivost se lahko pojavi pri trdnih diskih, ki imajo drugačno delitev diska kot je pri IBM XT. IBM XT dovoljuje največ 4 različne navezljive diskovne delitve (particije). Vsak operacijski sistem lahko uporablja eno samo delitev. Tako lahko trdni disk vsebuje particije za DOS, CPM-86 inali Concurrent CPM-86.

Preizkus zaslona temelji na znakih, ki jih najdemo v IBM Technical Reference Manual, dodatek C.

A.P.Železnikar

Kako si zgradite ibmovski PC?

Podjetje Handwell Corp, 4962 El Camino Real, Los Altos, CA 94022 nudi vrsto modulov za samogradnjo ibmovskega PC. Matična tiskana plošča ima tele značilnosti:

-- procesorja 8088 in 8087
-- prekinitveni krmilnik 8259A
-- DMA krmilnik 8237
-- dva 28-nožična ROMa
-- osem razširitvenih vtičnih plošč
-- na matični plošči ni RAMa

Cena gole (prazne) plošče je $ 69, cena plošče s pricinjenimi podnožji za integrirana vezja, upori, kondenzatorji, priključnicami, kristali in tranzistorji pa $ 199. Cena pripadajočih integriranih vezij je $ 199, preizkušena tiskana plošča z integriranimi vezji brez ROMov pa ima ceno $ 399.

K matični plošči je mogoče dokupiti še vtične plošče. Za osnovno konfiguracijo je potreben RAM, VI vrata, krmilnik upogljivega diska, napajalnik in še kaj. Vse te in druge module je mogoče dokupiti po relativno zmernih cenah, in sicer: do 256k RAM + 2 serijskih vrat + 1 paralelna vrata + ura realnega časa (cena $ 249); krmilnik za upogljive diske (cena $169); krmilnik za trdni disk ($ 299); modul za barvno grafiko ($ 219); monokromatični, barvni in grafični modul ($ 329); 100W napajalnik ($ 169); barvni RGB monitor ($ 449).

A.P.Železnikar

Ali je peta računalniška generacija že pred vrati ?

Da bo stvar jasnejša že na samem začetku:

Peta računalniška generacija
je nova tehnološka generacija

Današnje aplikacije umetne inteligence na računalnikih četrte generacije nimajo bistvenega vpliva na razvoj osnovne tehnologije pete generacije. Splošna zabloda je, da bo sama metodologija umetne inteligence razrešila bistvene probleme nove tehnološke generacije. Če je pri razvoju nove generacije potrebna nova človekova inteligenca, to še ne pomeni, da je to prav umetna inteligenca in samo te vrste inteligenca. Kaj pa je z novimi fizikalnimi in tehnološkimi proizvodnimi procesi, ki bodo potrebni za izdelavo novih integriranih komponent ? Kaj je pri tem bistveno, zakaj in kako ?

Američani se jezijo, da so jim Japonci vzeli idejo pete generacije in da jo jim zdaj spremenjeno prodajajo nazaj. Američanom izgleda, da se zgodovina ponavlja: to kar so doživeli na področju televizijske, stereo in avtomobilske tehnike, se sedaj ponavlja z računalniško tehniko. Pojem računalniških generacij je star in enostavno razumljiv. Prvi računalniki so bili narejeni z elektromehaničnimi releji in elektronkami. Konec petdesetih let so se pojavili tranzistorski računalniki in računalniki z elektronkami so postali zastareli in nesodobni. Ti stari računalniki so tako postali prva računalniška generacija, tranzistorski računalniki pa druga generacija.

Sredi šestdesetih let so se tranzistorji začeli umikati prvim integriranim vezjem in ibmovi sistemi 360 so bili glasniki tretje računalniške generacije. Znatno povečana integracija vezij je imela za posledico pojav četrte računalniške generacije (npr. sistemi IBM 370, mikroprocesorji, mikrokrmilniki, pomnilna vezja). Po četrti računalniški generaciji pa je bila najavljena na dokaj umeten način peta računalniška generacija, torej nekaj, kar se še ni pojavilo in za kar se niti natančno še ni vedelo, kaj naj bi to bilo.

Ko so Japonci začeli akcijo za razvoj pete generacije, je bilo jasno, da je bila tehnološka generacijska veriga na določen način prekinjena. V deklaracijo pete generacije je vstopila kot najbolj bistvena komponenta umetna inteligenca, vprašanje nove tehnologije pa je ostalo nekako obrobno in samo po sebi razumljivo: tako

je prevladala ideologija inteligence in prihajajoča generacija je dejansko postala neke vrste ideološka generacija.

Prehodi iz ene računalniške generacije v drugo naj bi bili na določen način revolucionarni, torej taki, da se z njimi de facto prekine dotedanja evolucija. Prehod med prvo in drugo generacijo je bil revolucionaren predvsem zaradi pojava tranzistorja (volumen, energija, operacijska hitrost, zanesljivost). Prehod iz tranzistorja na integrirano vezje je bil vsekakor manj revolucionaren (volumen, zanesljivost, reproducibilnost) kot pri prvem prehodu. Prehod med tretjo in četrto generacijo pa je bil izrazito evolucionaren in je omogočal še dolgo življenje tretje generacije ob četrti. Evolucija četrte generacije traja še danes in peta generacija se še ni pojavila. Ali pa so morda vendarle že vidne nekatere spremembe, s katerimi bi bilo mogoče najaviti prihod pete generacije?

Američani zlobno namigujejo, da so bili Japonci predvsem pozorni opazovalci, ki so preprosto znali šteti do pet in so tako opazili, da obstaja oznaka (kot programska labela) z imenom +peta generacija+. To ime so tako zapisali na svojo zastavo in hkrati razglasili, da pomeni peta generacija umetno inteligenco, govoreče, poslušajoče, videče, sklepajoče, robotične in še kake računalnike. Ta pristop pa je dejansko bližji ideološki kot tehnološki metodi in kaže prej na obnašanje nesposobnega kot sposobnega. Tako se je rodila in nastala današnja terminologija pete računalniške generacije.

Vse dosedanje računalniške generacije so temeljile na tehnologiji vezij in ne na aplikacijah. Bolj učinkovite in inteligentne aplikacije so možne le z zmogljivejšo tehnologijo.

Realnost pete generacije

Peta generacija pomeni nove m i k r o p r o c e s o r j e ! Do nedavnega je ta izjava zbujala smeh pri uporabnikih velikih (kabinetnih) računalnikov in prevladovalo je mnenje, da mikroračunalniki sploh niso pravi računalniki in da se iz njih ne more razviti nekaj, kar bi bilo podobno peti generaciji. To bi vsakakor lahko bilo res, če bi se razvoj ustavil pri procesorjih 6502 ali 8088. Vendar je npr. IBM PC XT370 (osebni računalnik) pravi računalnik, z močnim operacijskim sistemom, virtualnim pomnilnikom in zapletenimi programi.

Mikroprocesorji so integrirana vezja z visoko stopnjo integracije (LSI), ki so dejansko spremenili prej veljavni koncept računalnika. Toda s kakšno utemeljitvijo bi lahko trdili, da je XT370 že primerek iz pete računalniške generacije? XT370 postavlja mikroprocesorje neposredno v svet velikih računalnikov; z njimi vdira v ta svet in dokazuje, kako je to mogoče. Ta sistem ni več nekakšna igrača z močno sistemsko programsko opremo, s hitrimi komunikacijami in z možnostjo, da je integralni del velikega računalnika. Kmalu bodo XT370 povezani v mreže medoperacijskih sistemov z izredno hitrimi komunikacijami, kjer bodo zbirke drugih sistemov lahko dostopane na enak način in z enako hitrostjo kot lokalne zbirke. Pri tem bodo veljali standardni zaščitni mehanizmi in lastnosti večkratnega dostopa kot v virtualnih sistemih (VM). To pa so že lastnosti, ki vsaj delno zadovoljujejo zahteve po računalniški zmogljivosti pete generacije.

Argument proti gornjemu utemeljevanju je, da XT370 še ni mikroprocesor (še ni izdelan v enem samem inegriranem vezju). XT370 uporablja tri mikroprocesorje in še en koprocesor. Ti mikroprocesorji delujejo kot dva sodelujoča računalnika in oblikujejo združen sistem. Tu je procesor 8088 na matični plošči le še računalniški pomočnik in je namenjen VI obdelavam. Njegova funkcija je nadzor tipkovnice, zaslona, diskov in komunikacijske opreme. Glavni dodatek je nova emulacijska plošča za 3277, ki omogoča hitre komunikacije z ibmovskimi gostitelji.

Dva druga mikroprocesorja in koprocesor (dva 68000 in en 8087) oblikujejo centralni procesor sistema XT370. Ukazne množice namreč ni mogoče realizirati z enim samim mikroprocesorjem zaradi že nekoliko zastarele tehnologije (procesor 68000 pripada letniku 1982). V XT370 je ukazna množica razdeljena na tri dele in vsak od treh procesorjev nosi svoj delež. Prvi del ukazne množice se izvaja na modificiranem mikroprocesorju 68000. S spremembo mikrokoda v tem procesorju se dobijo ukazni kodi sistema 370.

Modificirani Intelov procesor 8087 izvaja matematične ukaze. Modificirano vezje uporablja ibmov standard in ne standard IEEE (kot pravi procesor 8087). Tako proizvaja XT370 numerične rezultate z enako natančnostjo kot veliki ibmovi računalniki.

Zadnji od treh procesorjev (68000) se uporablja za dokaj redko uporabljane ukaze in je počasnejši.

Teorija in praksa

Trije procesorji sestavljajo osrednjo procesno enoto namiznega računalnika XT370 in na njemu se lahko izvajajo praktično vsi programi velikega računalnika 370. Glavne omejitve izvirajo iz pomnilniškega naslavljanja, saj znaša obseg pomnilnika v XT370 512k zlogov. Ko v glavnem pomnilniku ni več prostora, poskrbi procesor 8088 za prostor na disku; tako je mogoče nasloviti 4M zloge. Veliki sistem 370 pa ima 16M zlogov virtualnega pomnilnika in zato večji programi od 4M zlogov na XT370 niso izvedljivi.

IBM PC XT370 je prvi kabinetni sistem, ki ga lahko postavimo na mizo (namizni računalnik). Tudi DEC pripravlja namizno izvedbo VAXa. IBM pripravlja novo družino vezij sistema 370 z najnovejšo tehnologijo (370 v integriranih vezjih). Ti novi računalniki pa bodo (gledano tehnološko) prvi predstavniki pete računalniške generacije.

Teorija pete generacije predvideva znatno povečanje paralelnih procesov v sistemu. To povečanje naj bi znašalo več velikostnih razredov v primerjavi z današnjimi računalniki in mikrora-

čunalniki. Le na ta način bi bilo mogoče reševati kompleksne probleme dovolj hitro, s hitrostmi, ki imajo praktično vrednost. V tem pa se skrivajo novi tehnološki problemi, za katere ne obstajajo danes niti zadovoljivi algoritmi (razbijanje procesov v dovolj veliko število paralelnih procesov in njihovo sestavljanje).

A.P.Železnikar

Ibmov PC AT

Ko bo na razpolago več podatkov, bomo v eni od naslednjih številk časopisa Informatica podrobneje opisali osebni računalnik PC AT. Zaenkrat si oglejmo le njegove osnovne specifikacije.

Tri leta po uvedbi računalnika IBM PC se je pojavila njegova izboljšava z oznako AT (Advanced Technology). Ta računalnik vsebuje mikroprocesor 80286 (s taktom 6 MHz in z 2- do 3-krat višjo zmogljivostjo kot PC), diskovno enoto z upogljivim diskom za 1,2 M zloga, vinčestrski disk za 20 M zlogov (kot opcijo), izboljšano in razširjeno vodilo, ki je združljivo s starim vodilom (dve priključnici za vsako vtično tiskano ploščo: ena za staro in dve za ploščo novega tipa), bolj uporabno novo tipkovnico itd. IBM je najavil tudi novo izvedenko operacijskega sistema PC DOS 3.0 z manjšimi izboljšavami in z mrežno podporo, vendar brez lastnosti večopravilnosti (multitasking). V prvi četrtini leta 1985 bo IBM uvedel TopView, tj. operacijsko okolje, ki omogoča večopravilnost za ceno $ 149 in posebno izvedenko operacijskega sistema Xenix (Unix System 3) za procesor 80286 z zmogljivostjo večopravilnosti in večuporabnosti (multiuser).

Cene teh novih izdelkov so izredno konkurančne. Osnovna cena za IBM PC AT brez vinčestrskega diska je $ 3995. Delovni sistem v tkim. Byte konfiguraciji (dve diskovni enoti, 256 k zlogov RAMa, prikazovalnik, barvna podpora, vmesniki in DOS) ima ceno okoli $ 5700. Za standardno Byte konfiguracijo z vinčestrskim diskom za 20 M zlogov pa je cena okoli $ 6700.

Sistemska enota

Sistemska enota je za kakšno colo (2,54 cm) debelejša od PCja. Diskovne enote imajo polovično višino in so sivkasto rjave barve. Na sprednji strani je tudi vključitveni ključavnica (kot pri avtomobilu).

Bistvene spremembe so v notranjosti sistema. Najprej je tu mikroprocesor 80286 s taktno frekvenco 6 MHz. Ta procesor je pravi 16-bitni procesor in ima napredne zmogljivosti za upravljanje pomnilnika do 1 G zloga (G za giga).

Procesor 80286 lako deluje v dveh načinih. Z operacijskim sistemom PC-DOS deluje v "realnem naslovnem" načinu, v katerem emulira tako procesor 8086 kot 8088 ibmovega PC. V tem načinu ima 80286 le neposreden dostop do sistemskega pomnilnika z obsegom 640 k zlogov. V drugem načinu z operacijskim sistemom Xenix pa ima 80286 dostop do 16 M zlogov fizičnega pomnilnika.

IBM zagotavlja, da je AT 2- do 3-krat hitrejši od PCja. Glede na taktno frekvenco je ta sistem dejansko hitrejši za 25%, nadaljno 40%-no povečanje pa nastane zaradi 16-bitnega podatkovnega vodila. Konkreten preizkus je pokazal pohitritev za 150%, kar je 2,5-krat hitreje kot PC.

Minimalni obseg pomnilnika je 256 k zlogov, maksimalni pa 3 M zloge (pri uporabi petih razširitvenih vtičnig plošč s po 512 k zlogi).

Diskovne enote

AT uporablja nov tip diskovnih enot s po 1,2 M zloga. Te polvisoke enote dosežejo to gostoto s posebnimi diski pri 96 tpi (tpi je track per inch, tj. število stez na colo) v primerjavi z 48 tpi pri PCju in s 15 sektorji na stezo namesto z 9 sektorji na stezo pri PC-DOS 2.0. Te enote lahko preberejo standardne 360 k zlogovne diske.

V ATju je dovolj prostora za dve takšni enoti. Zaradi združljivosti s PC je mogoče instalirati standardno 360 k zlogovno diskovno enoto namesto druge diskovne enote.

K dvema diskovnima enotama za upogljive diske je mogoče dodati vinčestrski disk obsega 20 M zlogov. Ker je višina te enote polovična, jo je mogoče instalirati namesto enote za upogljivi disk v sistemsko ohišje.

Druga materialna oprema

Razširitveno vodilo v AT omogoča določeno združljivost z vodilom PCja, hkrati pa se lahko uporabijo tudi razširitvene vtične plošče, ki uporabljajo dodatne podatkovne in naslovne linije procesorja 80286. Novo vodilo je povezano na dve ločeni priključnici, in sicer na 62-nožično (IBM PC) in dodatno 36-nožično za 6 vtičnih plošč. Dve vtični plošči pa lahko imata le po eno 62-nožično priključnico.

AT je mogoče na sprednji strani zakleniti (izključiti in vključiti) s ključem. Tako se zaklene tudi tipkovnica, med tem pa se lahko izvaja dolg program ali pa je sistem vključen v mrežo. AT ima serijske in paralelno vmesnike na vtičnih ploščah. Ima tudi uro in koledar (v CMOS vezjih). Posebno stikalo na matični plošči pove sistemu, kakšen prikazovalnik je priključen.

Operacijski sistem PC-DOS 3.0

PC AT deluje z izboljšanim operacijskim sistemom PC-DOS, ki nosi oznako 3.0 in upošteva arhitekturo ATja. Pri prvi uporabi sistema s PC-DOS 3.0 se mora uporabiti začetna (zagonska) rutina, prek katere se definira število diskovnih enot, obseg pomnilnika itd. in ta informacija se shrani v posebnem CMOS pomnilniku (ki je trajno vključen podobno kot ura). Ti podatki se upoštevaju pri vsaki vključitvi računalnika.

PC DOS 3.0 podpira visoko zmogljive enote za upogljivi disk (1,2 M zlogov na enoto) za standardne eno in dvostranske diskete ter enega ali dva vinčestrska diska. Ti diski se lahko uporabljajo v različnih kombinacijah. Visoko zmogljive disketne enote shranijo 600 k zlogov na eno stran diskete, in sicer na 80 stezah s 15 sektorji na stezo. Štirje novi ukazi (Format, Backup, Diskcomp in Diskcopy) podpirajo te visokozmogljive enote. Ukaz Backup omogoča selektivno shranjevanje v treh kategorijah:

-- shranjevanje zbirk, modificiranih po zadnjem shranjevanju
-- dodajanje novih zbirk na shranjevalni disk in
-- shranjevanje zbirk, modificiranih po določenem datumu

Eden najprivlačnejših novih ukazov je Device, ki omogoča instalacijo virtualnih diskov. Takšnih 24 virtualnih diskov, ki se imenujejo tudi

gorenje procesna oprema

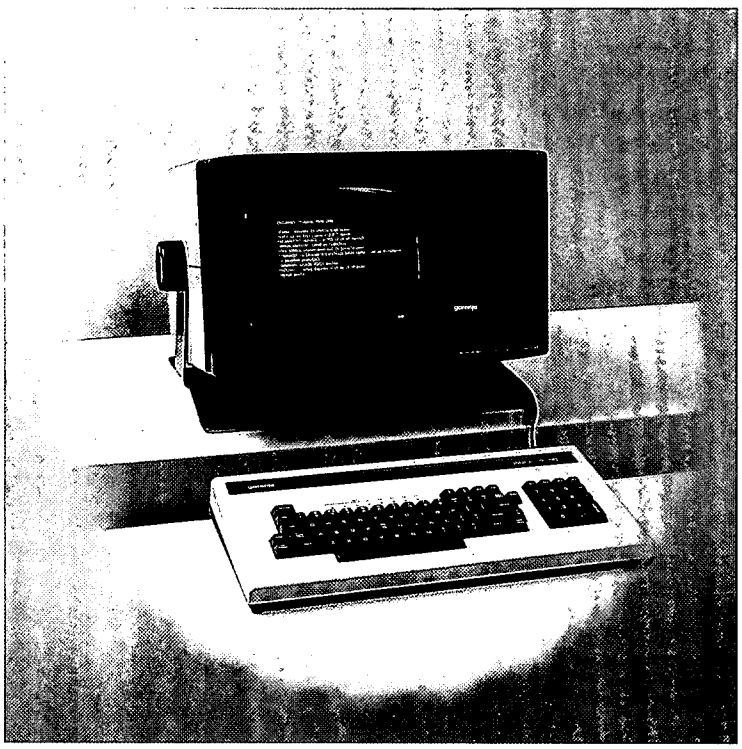
Zaslonski terminal PAKA 3000 je računalniška vhodno/izhodna enota. Terminal je zasnovan na mikroprocesorski tehnologiji in ga lahko izpopolnimo in usposobimo za opravljanje zahtevnejših nalog. Je enostaven, vendar z lastnostmi, ki olajšajo delo in izboljšajo komunikacijski odnos računalnik - človek.

Tipkovnica

Podobna je tipkovnici pisalnega stroja in je ločena od ohišja monitorja. Z njim jo povezuje kabel, dolg 1,50 m, ki dovoljuje postavitev monitorja in tipkovnice v različne položaje. S tem dosežemo zorni in delovni kot. Na tipkovnici so posebne funkcijske tipke za prenos kontrolnih znakov, ki krmilijo delovanje terminala. Skupina številčnih in funkcijskih tipk, oblikovana podobno kot pri kalkulatorjih, služi za vnašanje numeričnih podatkov in uporabo programskih operacij na terminalu. Na tipkovnici je 8 indikatorjev, ki dajejo operaterju informacijo o delovanju terminala in služijo za odkrivanje napak.

Zaslon

Ena od prednosti zaslonskega terminala PAKA 3000 je, da lahko prikazuje poročila v dveh formatih: po 80 in 132 znakov v vrstici. 132 znakov v vrstici omogoča zapis poročil, ki so standardno generirana v formatu za tiskalnik in neposreden prenos iz zaslona na tiskalnik brez preoblikovanja. Pri drsečem pomiku (SMOOTHSCROLL) lahko operater kontrolira podatke pri visokih hitrostih prenosa. S tipko NO-SCROLL pa lahko izpis kjerkoli ustavi in ga s pritiskom ponovno sproži. Zaslon lahko logično razdelimo tako, da se del 24-vrstičnega zaslona odvija ločeno. Podatke lahko vpisujemo na enem in izpisujemo na drugem delu zaslona, kar je ugodno za programerje in operaterje. S pritiskom na tipko SET-UP operater nadomesti vsebino zaslona s standardno sliko (SET-UP). S pomočjo te slike lahko izbere število znakov v vrstici, določi svetlost zaslona, nastavi tabulatorje, izbere prenosne hitrosti in druge lastnosti terminala (npr. svetlo ozadje na zaslonu, obliko zaslonskega kazalca, mejni signal).



Ekranski terminal PAKA 3000 je računarska ulazno/izlazna jedinica. Terminal je koncipiran na tehnologiji mikroprocesora te ga možemo usavršiti i osposobiti za složenije zadatke. Iako je jednostavan ima karakteristike koje olakšavaju rad i poboljšavaju komunikacijski odnos čovjeka i računara.

Tastatura

Tastatura je nalik na tastaturu pisaćeg stroja i odvojena je od kućišta monitora s kojim je povezana kablom dužine 1,5 m koji dopušta postavljanje monitora i tastature na različite načine. Time postižemo najbolji kut gledanja i rada. Tastatura ima i posebne funkcijske tipke za prijenos kontrolnih znakova koji upravljaju radom terminala. Skupina numeričkih i funkcijskih tipki oblikovana je slično kao kod kalkulatora te služi za unošenje numeričkih podataka i programske operacije na terminalu. Na tastaturi postoji i 8 indikatora koji informiraju operatera o djelovanju terminala i služe za otkrivanje grešaka.

Ekran

Jedna od prednosti ekranskog terminala PAKA 3000 sastoji se u tome što može prikazivati poruke u dva formata: sa po 80 ili 132 znaka z retku; 132 znaka u retku omogućuju bilježenje poruka kad su standardno generirane u formatu za štampač (printer) i direktan prijenos s ekrana na štampač bez preoblikovanja. Pri kliznom pomaku (SMOOTHSCROLL) operater može kontrolirati podatke prilikom velikih brzina prijenosa. Tipkom NO-SCROLL izvod se može bilo gdje zaustaviti i pritiskom tipke ponovno pokrenuti. Ekran možemo logično podijeliti na taj način da se dio ekrana od 24 retka odvija posebno. Podatke možemo upisivati na jednom i ispisivati na drugom dijelu ekrana, što je prikladno i za programere i za operatere. Pritiskom na tipku SET-UP operater zamjenjuje sadržaj ekrana standardnom slikom (SET-UP) pomoću koje može izabrati broj znakova u retku, odrediti svijetlost ekrana, namjestiti tabulatore, izabrati prijenosne brzine i druga svojstva terminala (npr. svijetla pozadina na ekranu, oblik ekranske kazaljke, granični signali).

RAM diski, podpira PC-DOS 3.0. Lokacije virtualnih diskov se nahajajo nad naslovom 1 M zlog.

Izvedenka 3.0 PC DOSa je opremljena s tremi priročniki in z referenčno kartico. Uporabniški priročnik je uvod v PC DOS za začetnike z osnovami operacijskega sistema v enostavnem tekstu in z barvno grafiko.

Programski paketi Basic, TopView in Xenix

Izpeljanka 3.0 za Basic vsebuje več razširitev. Shell stavek naloži in izvrši programsko zbirko (imenovano otroški proces), ki se potem lahko uporablja na sistemski ravnini v okviru jezika Basic. Če izvrševana zbirka tipa .BAT končuje z Exit stavkom, se programsko krmiljenje vrne sistemu Basic. Obstaja več pogojev, ki pri uporabi stavka Shell lahko povzročijo sistemski zlom, npr. pri dostopanju v DMA krmilnik, prekinitveni krmilnik, V/I vmesnik ali časovniški števnik iz otroškega procesa ali pri spreminjanju zbirke, ki je bila odprta v sistemu Basic iz otroškega procesa.

Funkcija Environ$ vrne informacijo v okoliško tabelo Basica (ta je locirana v imeniku DOS). Environ stavek pa modificira okoliško tabelo Basica.

TopView je poseben programski paket, ki vsebuje vse to, kar je bilo pričakovano v PC-DOS 3.0. Ta paket se bo pojavil v prvem četrtletju leta 1985. TopView je zanimivo večopravilno okolje z zmogljivostmi oknenja (windowing). TopView je mogoče uporabljati tudi z miško za nastavljanje oken, ki izvajajo aplikacije istočasno. Cena tega paketa bo $ 149 in z njo naj bi IBM konkuriral operacijskemu sistemu

Concurrent PC-DOS
(Concurrent CP/M-86 Version 3.1 z emulacijo sistema PC-DOS)

podjetja Digital Research, ki ima podobne zmogljivosti in ceno. Seveda pa se lahko na Concurrent PC-DOSu izvajajo tudi vse aplikacije sistema CP/M-86.

Napovedani operacijski sistem Xenix za AT se bo podobno kot TopView pojavil šele v prvem četrtletju leta 1985. Ta sistem je omejen na tri uporabnike, saj ima AT le dvoje serijskih vrat. Xenix ima prijazno uporabniško lupino, ki je podobna Multiplanu, trenutno edinemu aplikacijskemu paketu, ki ga ima IBM za Xenix.

PCjevska nezdružljivost

IBM opozarja, da naslednji materialni moduli PCja ne delujejo v povezavi z ATjem:

- standardna PC tipkovnica
- paralelnotiskalniška vtična plošča
- pomnilniška plošča za 64 do 256 k zlogov
- plošča za asinhrono komunikacijo (serijska vrata) in
- razširitvena enota

Zelo verjetno je, da tudi vrsta proizvodov dosedanjih proizvajalcev vtičnih plošč ne bo delovala v PC ATju.

IBM opozarja tudi na programske pakete, ki na AT niso uporabni:

- BPI System Accounting 1.0
- Peachtree Accounting 1.0
- EasyWriter 1.0
- pfs: File and Report 1.0
- VisiCalc 1.0 in 1.1
- Basic 1.X in 2.X
- Flight Simulator (Microsoft)

Uspešno pa delujejo na ATju Wordstar, Lotus 1-2-3, Multiplan in Peachtext.

IBM opozarja, da ne bo mogoče uporabljati PC programov, ki vsebujejo Poke stavke (pomnilniško specifični ukazi), programske stavke s časovnim krmiljenjem, povratne V/I ukaze na ista vrata (zaradi prekratkega obnovitvenega časa), PUSH SP ukaze itd.

Sklep

Danes deluje AT le v načinu realnih naslovov s sitemom DOS 3.0. V tem načinu je operacijska hitrost ATja 2,5-krat večja od hitrosti PCja. AT je zanimiv za obdelave, kot je zbiranje podatkov, saj ima 16 prekinitvenih ravnin in 7 DMA kanalov. DMA krmilnik uporablja takt 3 MHz, kar je dokaj hitro. DMA kanali 0, 1, 2, 3 podpirajo 8-bitni prenos v okviru 16 M-zložnega fizičnega naslovnega prostora v 64 k-zložnih blokih. Kanali 5, 6, 7 podpirajo 16-bitni prenos v 128 k-zložnih blokih. Kanal 4 se uporablja kot kaskada kanalov 0, 1, 2, 3 k procesorju 80286. IBM je dokumentiral DMA krmilnik za AT v priročniku.

PC AT je sicer zanimiv računalnik, vendar njegova glavna uporaba šele prihaja z učinkovitim večuporabniškim operacijskim sistemom, kot je Unix. Bolj izčrpne ocenitev ATja bo možna šele, ko bo zanj dobavljiv tudi Xenix.

Zbirni podatki za IBM PC AT

Izdelovalec: IBM Entry System Division
POB 1328
Boca Raton, FL 33432, USA

Procesor: 16/16-bitni Intel 80286, takt 6 MHz

Pomnilnik: 256 k do 3 M zlogov

Diskovne enote:
enota za upogljivi disk premera 5,25 cole, 1,2 M zlogov
trdni disk za 20 M zlogov kot izbirna možnost
enota za upogljivi disk premera 5,25 cole, 360 k zlogov kot izbirna možnost

Operacijski sistem:
IBM PC-DOS 3.0
V prihodnosti dobavljiva:
DOS 3.1 in Xenix

Programska oprema: IBM PC Basic 3.0

Cena: Z diskovno enoto 1,2 M zlogov ... $ 3995
Kot zgoraj in z monokromatičnim prikazom, barvno grafiko, drugim diskom (360 k) in DOSom ... $ 5469
Z diskom 1,2 M, trdnim diskom 20 M zlogov, 512 k pomnilnikom ... $ 5795
Kot zgoraj in z monokromatičnim prikazom, barvno grafiko in DOSom ... $ 6694

A. P. Železnikar

DOMAČA GRAFIČNA OPREMA SNOVANJE, PREDSTAVITEV IN IZRISOVANJE ČRNOBELIH SLIK

V Odseku za računalništvo in informatiko Instituta Jožef Stefan v Ljubljani ob podpori Raziskovalne skupnosti Slovenije razvijamo, implementiramo in prototipno izdelujemo grafično aparaturno in programsko opremo za programiranje, predstavitev in izrisovanje črnobelih slik na družini računalnikov Iskra-Delta ter DEC pod operacijskimi sistemi RT-11, RSX-11, VMS ter njihovimi domačimi izvedbami. Na sedanji stopnji razvoja lahko ponudimo končnim uporabnikom ter računalniškim proizvajalcem paket grafične aparaturne in programske opreme, ki obsega:

- standardni grafični programski paket GKS za računalnike pod operacijskim sistemom VMS;
- grafični procesor kot dodatek za videoterminal KOPA 1000 oziroma DEC VT100;
- grafični dodatek za risanje na matričnem pisalniku DEC LA-120;
- grafični vmesnik za risanje na matričnem pisalniku FACIT 4540;

V bližnji prihodnosti pa bo dokončan razvoj naslednje grafične opreme:

- digitalizacijska tablica;
- grafični procesor za videoterminal Gorenje;
- programska knjižnica programiranje grafike na miniračunalnikih tipa DEC PDP-11 in LSI-11 in podobnih računalnikih Iskra-Delta ter na podobnem računalniku IJS PMP-11;

GRAF-100
GRAFIČNI PROCESOR ZA VIDEOTERMINAL KOPA 1000 (VT100)

Institut Jožef Stefan je razvil in izdeluje grafični procesor GRAF-100 za vgradnjo v videoterminal Kopa 1000 oziroma VT100. S tem dodatkom pridobi videoterminal zmožnosti grafičnega terminala z ločljivostjo 650 x 240 svetlobnih točk ter pri tem ohrani vse lastne zmožnosti alfanumeričnega terminala. Bistvena prednost tega grafičnega procesorja pred uvoženimi procesorji tega tipa je v velikem številu (16) nivojev svetlobne intenzivnosti posamezne točke. To zmožnost procesor GRAF-100 izrablja za navidezno dvakratno povečanje ločljivosti s pomočno operacij za odpravo stopničenja (anti-aliasing) - zmožnost, ki so jo doslej omogočali le grafični procesorji najvišjega cenovnega razreda. Zmožnost risanja z velikim številom poltonov med črno in belo barvo omogoča uporabo tega grafičnega terminala za upodabljanje prostorskih objektov v strojništvu, gradbeništvu, lesarstvu, elektroniki in drugod.

LAGRAF-120
GRAFIČNI DODATEK ZA RISANJE NA MATRIČNEM PISALNIKU DEC LA-120

Grafični dodatek LAGRAF-120 omogoča uporabo matričnega pisalnika DEC LA-120 za rastrsko risanje z visoko ločljivostjo. Pri tem tiskalnik ohrani vse svoje zmožnosti za alfanumerično tiskanje. Dodatek LAGRAF-120 omogoča risanje z ustreznimi ukaznimi nabori, ki so kompatibilni z DECwriter IV-RA. Velikost in poraba električne energije sta manjši v primerjavi s podobnim dodatkom Selanar SG-120. Vgradnja plošče je zelo enostavna, tako da jo lahko izvede vsak brez posebnega orodja v nekaj minutah.

institut "jožef stefan" ljubljana, jugoslavija

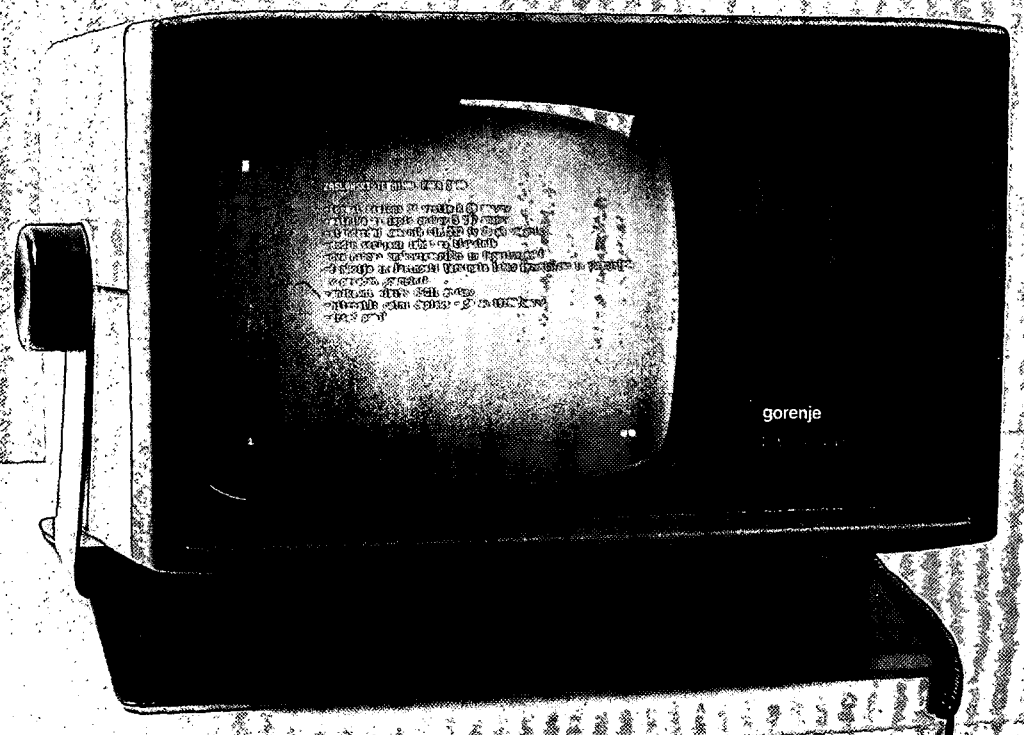
gorenje procesna oprema

Gorenje Procesna oprema, *n. sol. o.*
Celjska 5 a
63320 Titovo Velenje
Jugoslavija
Telefon: (063) 853 321
Telex: 33547 yu tgove

gorenje Leto kakovosti 1984

Zaslonski terminal PAKA 3000
Model TP 103

Ekranski terminal PAKA 3000
Model TP 103



gorenjeprocesna oprema

Tehnične specifikacije

Dimenzije:	
Monitor	
brez podstavka	dolžina 46 cm širina 43 cm višina 28 cm
s podstavkom	dolžina 52 cm širina 43 cm višina 36 cm
Tipkovnica	dolžina 46 cm širina 24 cm višina 6 cm
Masa:	15,6 kg
Pogoji delovanja:	temperatura od 10-40°C relativna vlaga 10-90%
Napajanje:	220 V ± 10%, 50 Hz, 50 W
Zaslon	
Katodna cev Format	diagonala meri 31 cm, fosfor GR (P31) 24 vrstic po 80 znakov ali 24 vrstic po 132 znakov (po izbiri)
Znaki	matrika s 7 × 9 točkami
Aktivna površina zaslona	205 mm × 165 mm ± 2,5 mm
Znakovni niz	96 ASCII in 106 semigrafičnih znakov
Tipkovnica	
Tipke	65 tipk je izdelanih in razporejenih podobno kot pri pisalnem stroju
Pomožna tipkovnica	18 numeričnih tipk s piko, vejico, minusom, tipko ENTER in štirimi programsko-funkcijskimi tipkami, zvočna potrditev vtipkanega znaka in mejni signal za napako
Povezave:	
Tip	EIA (RS-232-C)
Hitrosti:	polni dupleks 50, 75, 110 (dva stop bita), 134, 150, 200, 300, 600, 1200, 1800, 2000, 2400, 3600, 4800, 9600, 19200
Format znakov	asinhronski
Dolžina znakov	5, 6, 7 ali 8 bitov; (če izberemo 8 bitov za znak, osmi bit ne nosi informacije)
Kode	USASCII, JUS A.F0.101
Parnost	soda, liha ali je ni
Sinhronizacija	s tiskalnikom: CTS ali XON/XOFF s sistemom: kontrolne kode DC 1, DC 3 (XON/XOFF)

Tehničke specifikacije

Dimenzije:	
Monitor	
bez podloška	dužina 46 cm širina 43 cm visina 28 cm
s podloškom	dužina 52 cm širina 43 cm visina 36 cm
Tastatura	dužina 46 cm širina 24 cm visina 6 cm
Težina:	15,6 kg
Uvjeti djelovanja:	temperatura od 10-40°C relativna vlaga 10-90%
Napajanje:	220 V ± 10%, 50 Hz, 50 W
Ekran	
Katodna cijev Format	dijagonala 31 cm fosfor GR (P 31) 24 retka po 80 znakova ili 24 retka po 132 znaka (po izboru)
Znaci	matrica sa 7 × 9 točaka
Aktivna površina ekrana	205 mm × 165 mm ± 2,5 mm
Niz znakova	96 ASCII i 106 semigrafskih znakova
Tastatura	
Tipke	65 tipki izvedenih i raspoređenih slično kao kod pisaćeg stroja
Pomoćna tastatura	18 numeričkih tipki s tačkom, zarezom, minusom i tipkom ENTER te sa četiri programsko-funkcijske tipke, zvučno potvrđivanje otipkanog znaka i granični signal za grešku
Povezivanje	
Tip	EIA (RS-232-C)
Brzine:	puni dupleks 50, 75, 110 (dva stop bita), 134, 150, 200, 300, 600, 1200, 1800, 2000, 2400, 3600, 4800, 9600, 19200
Format znakova	asinhroni
Dužine znakova	5, 6, 7 ili 8 bita (ako odaberemo 8 bita za znak osmi bit nije nosilac informacije)
Kodovi	USASCII, JUS A. F0. 101
Parnost	parni, neparni ili je nema
Sinhronizacija	s printerom: CTS ili XON/XOFF sa sistemom: kontrolni kodovi DC 1, DC 3 (XON/XOFF)

gorenjeprocesna oprema

Zaslonski terminal PAKA 3000 je računalniška vhodno/izhodna enota. Terminal je zasnovan na mikroprocesorski tehnologiji in ga lahko izpopolnimo in usposobimo za opravljanje zahtevnejših nalog. Je enostaven, vendar z lastnostmi, ki olajšajo delo in izboljšajo komunikacijski odnos računalnik - človek.

Tipkovnica

Podobna je tipkovnici pisalnega stroja in je ločena od ohišja monitorja. Z njim jo povezuje kabel, dolg 1,50 m, ki dovoljuje postavitev monitorja in tipkovnice v različne položaje. S tem dosežemo zorni in delovni kot. Na tipkovnici so posebne funkcijske tipke za prenos kontrolnih znakov, ki krmilijo delovanje terminala. Skupina številčnih in funkcijskih tipk, oblikovana podobno kot pri kalkulatorjih, služi za vnašanje numeričnih podatkov in uporabo programskih operacij na terminalu. Na tipkovnici je 8 indikatorjev, ki dajejo operaterju informacijo o delovanju terminala in služijo za odkrivanje napak.

Zaslon

Ena od prednosti zaslonskega terminala PAKA 3000 je, da lahko prikazuje poročila v dveh formatih: po 80 in 132 znakov v vrstici. 132 znakov v vrstici omogoča zapis poročil, ki so standardno generirana v formatu za tiskalnik in neposreden prenos iz zaslona na tiskalnik brez preoblikovanja. Pri drsečem pomiku (SMOOTHSCROLL) lahko operater kontrolira podatke pri visokih hitrostih prenosa. S tipko NO-SCROLL pa lahko izpis kjerkoli ustavi in ga s pritiskom ponovno sproži. Zaslon lahko logično razdelimo tako, da se del 24-vrstičnega zaslona odvija ločeno. Podatke lahko vpisujemo na enem in izpisujemo na drugem delu zaslona, kar je ugodno za programerje in operaterje. S pritiskom na tipko SET-UP operater nadomesti vsebino zaslona s standardno sliko (SET-UP). S pomočjo te slike lahko izbere število znakov v vrstici, določi svetlost zaslona, nastavi tabulatorje, izbere prenosne hitrosti in druge lastnosti terminala (npr. svetlo ozadje na zaslonu, obliko zaslonskega kazalca, mejni signal).

Ekranski terminal PAKA 3000 je računarska ulazno/izlazna jedinica. Terminal je koncipiran na tehnologiji mikroprocesora te ga možemo usavršiti i osposobiti za složenije zadatke. Iako je jednostavan ima karakteristike koje olakšavaju rad i poboljšavaju komunikacijski odnos čovjeka i računara.

Tastatura

Tastatura je nalik na tastaturu pisaćeg stroja i odvojena je od kućišta monitora s kojim je povezana kablom dužine 1,5 m koji dopušta postavljanje monitora i tastature na različite načine. Time postižemo najbolji kut gledanja i rada. Tastatura ima i posebne funkcijske tipke za prijenos kontrolnih znakova koji upravljaju radom terminala. Skupina numeričkih i funkcijskih tipki oblikovana je slično kao kod kalkulatora te služi za unošenje numeričkih podataka i programske operacije na terminalu. Na tastaturi postoji i 8 indikatora koji informiraju operatera o djelovanju terminala i služe za otkrivanje grešaka.

Ekran

Jedna od prednosti ekranskog terminala PAKA 3000 sastoji se u tome što može prikazivati poruke u dva formata: sa po 80 ili 132 znaka z retku; 132 znaka u retku omogućuju bilježenje poruka kad su standardno generirane u formatu za štampač (printer) i direktan prijenos s ekrana na štampač bez preoblikovanja. Pri kliznom pomaku (SMOOTHSCROLL) operater može kontrolirati podatke prilikom velikih brzina prijenosa. Tipkom NO-SCROLL izvod se može bilo gdje zaustaviti i pritiskom tipke ponovno pokrenuti. Ekran možemo logično podijeliti na taj način da se dio ekrana od 24 retka odvija posebno. Podatke možemo upisivati na jednom i ispisivati na drugom dijelu ekrana, što je prikladno i za programere i za operatere. Pritiskom na tipku SET-UP operater zamjenjuje sadržaj ekrana standardnom slikom (SET-UP) pomoću koje može izabrati broj znakova u retku, odrediti svijetlost ekrana, namjestiti tabulatore, izabrati prijenosne brzine i druga svojstva terminala (npr. svijetla pozadina na ekranu, oblik ekranske kazaljke, granični signali).

gorenjeprocesna oprema

Gorenje Commerce, *p. o.*
Celjska 5a
63320 Titovo Velenje
Jugoslavija
Telefon: (063) 853321
Telex: 33616 yu sogor

Znaki

Matrika za izpis znakov obsega 7 × 9 točk in se razprostira na prostoru 9 × 12 točk, kar omogoča spuščanje ležečih znakov za dve točki. Operater lahko izbere svetle znake na temni podlagi ali temne znake na svetli podlagi, in sicer za vsak znak posebej ali za cel zaslon. Ta lastnost poudarja določene dele teksta, temni znaki na svetlem ozadju pa dajejo videz tiskanega teksta na papirju. Znaki so lahko še utripajoči in poudarjeni. Uporabniku je na voljo dvojna višina in dvojna širina znakov, s čimer dosežemo preglednost teksta in čitanje na večjo razdaljo. Možne so tudi vse kombinacije različnih slik znakov. Osnovni niz znakov vsebuje poleg črk, številk in ločil še 106 grafičnih znakov za prikaz grafičnih informacij na zaslonu. Izberemo lahko dva različna nabora znakov: ameriški in jugoslovanski.

Splošni podatki

Zaslonski terminal TP 103 ima mehansko stikalo za vklop terminala. Vse druge funkcije terminala, kot so prenosna hitrost, tabulatorji, pariteta, itd. so shranjene v posebnem pomnilniku in jih spreminjamo preko tipkovnice. Nastavljive lastnosti terminala se ohranijo tudi če terminal izključimo in ga ponovno vključimo. Odstranitev mehanskih stikal olajša uporabo testnih diagnostičnih programov in omogoča enostavno prilagajanje terminala. Vgrajeni testni diagnostični programi poenostavijo vzdrževanje in zmanjšajo čas osamitve in popravila napak.
Univerzalni močnostni del je prilagojen za napajanje terminala in vseh dodatkov in omogoča njihovo vgrajevanje na terenu.
Terminal TP 103 deluje z dupleksno asinhrono komunikacijsko linijo in ima standardni vmesnik EIA 232 in 20 mA vmesnik.
Novost terminala TP 103 je, da poleg glavnega vhoda vsebuje serijski izhod za tiskalnik.

Znaci

Matrica za ispisivanje znakova obuhvaća 7 × 9 točaka i rasprostire se na prostoru od 9 × 12 točaka što omogućuje spuštanje nižih znakova za dvije točke. Operater može izabrati svijetle znake na tamnoj podlozi ili tamne znake na svijetloj podlozi i to za svaki znak posebno ili za cijeli ekran. Tim svojstvom ističu se određeni dijelovi teksta, a tamni znaci na svijetloj podlozi daju dojam teksta štampanog na papiru. Pored toga znaci mogu titrati ili biti još naglašeniji.
Korisnik može raspolagati sa dvije visine i dvostrukom širinom znakova čime se postiže preglednost teksta i omogućuje čitanje s veće udaljenosti. Isto tako moguće su i sve kombinacije različitih slika znakova. Osnovni niz znakova pored slova, brojeva i interpunkcije sadrži i 106 grafičkih znakova za prikaz grafičkih informacija na ekranu. Možemo birati između dva različita niza znakova: američkog i jugoslavenskog.

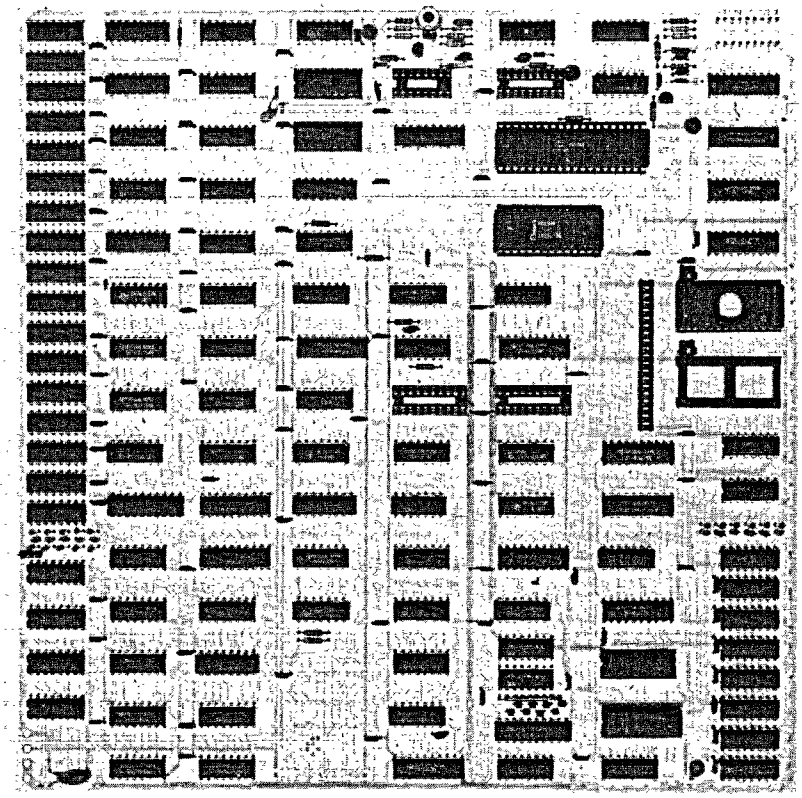
Opći podaci

Ekranski terminal TP 103 ima mehanički prekidač za uključivanje terminala dok su sve druge funkcije terminala kao što su brzina prijenosa, tabulatori, paritet itd. pohranjeni u posebnoj memoriji i pratimo ih preko tastature. Namjestiva svojstva terminala održavaju se i ako terminal isključimo te ga ponovo uključimo. Uklanjanje mehanskih prekidača olakšava primjenu testnih dijagnostičnih programa i omogućuje jednostavno prilagođavanje terminala. Ugrađeni testni dijagnostični programi pojednostavljaju održavanje i skraćuju vrijeme lociranja i popravku grešaka.

Univerzalni dio za jakost prilagođen je za napajanje terminala i svih dodataka te omogućuje njihovo ugrađivanje na terenu.
Terminal TP 103 djeluje pomoću dupleks asinhrone komunikacijske linije i ima standardni interface EIA 232 i interface 20 mA.
TP 103 ima i tu novost da pored glavnog ulaza sadrži i serijski izlaz za štampač.

Gorenje Commerce EP / Titovo Velenje / september '84 / Tisk: Tiskarna Ljubljana

LAGRAF–120
GRAFIČNI DODATEK ZA RISANJE NA MATRIČNEM PISALNIKU DEC LA–120



GRAF–100
GRAFIČNI PROCESOR ZA VIDEOTERMINAL KOPA 1000 (VT100)