METODOLOGIJA SNOVANJA IZVRŠILNEGA PROCESORJA ZA 32-BITNI RAČUNALNIŠKI SISTEM

MAKSIMILJAN GERKEŠ

VISOKA TEHNIŠKA ŠOLA, VTO ELEKTROTEHNIKA, MARIBOR

UDK: 681.326

Članek podaja pregled postopkov snovanja, uporabljenih pri izdelavi prototipa 32-bitnega izvršilnega procesorja, ki je sestavni del 32-bitnega računalniškega sistema, ki podpira kompleksno instrukcijsko množico.
Izvršilni procesor je po predlaganem postopku v začetku definiran kot univerzalni trioperandni operator, ki ga postopoma razgrajujemo v vedno večje detalje. Vrednosti parametrov hrani posebna pomnilna struktura, ki je dopolnjena z delovno pomnilno strukturo in omogoča hranjenje vmesnih rezultatov zaradi interpretacije operacij, definiranih z instrukcijsko množico. Zaradi interpretacije operacij je vpeljana mikroprogramirana krmilna struktura, ki je definirana na podlagi karakterističnih lastnosti mikroprogramov.

DESIGN METHODOLOGY FOR A 32-BIT EXECUTION PROCESSOR FOR A 32-BIT COMPUTER SYSTEM: Article describes design methods used for a 32-bit execution processor prototype design. 32-bit execution proccessor is relatively high speed universal operator support for a 32-bit computer system based on a complex instruction set.
At the beginning we define execution processor as a universal three operand operator. With stepwise refinement it is then decomposed into lower level parts. In and outcomming parameter values are written into I/O register set which is completed with working register set for holding partial results, generated with interpreted operations, defined by complex instruction set. Because of operations interpretation microprogrammed control structure is defined based on the characteristic features of microprogramms.

UVOD:

Porazdelitev operacij, pri implementaciji neke računalniške arhitekture, na primerno organizirane sisteme je osnovna naloga vsakega snovanja. V preprostejših primerih so sheme, ki definirajo povezavo takih sistemov v celoto enostavne, organizacija sistemov pa je preprosta. Kadar pa je potrebno za implementacijo izdelati modele kompleksnih arhitektur, pa je takšna naloga znatno težavnejša. Pogosto imajo dominanten vpliv na organizacijo takega modela zahteve po performansah.

V našem primeru smo izdelali model organizacije izvršilnega procesorja, ki izvaja operacije določene s kompleksno instrukcijsko množico VAX arhitekture. Globalni model sistema je namreč takšen, da predvideva delitev operacij na instrukcijski in izvršilni del. Pri tem ima instrukcijski del sistema nalogo strežbe izvršilnemu procesorju, ki izvaja zahtevane operacije. Osnovna naloga instrukcijskega procesorja (strežba) definira lastnost instrukcijskega dela, ki je vhodno-izhodno intenziven. Medtem pa je izvršilni del (izvajanje operacij) operacijsko intenziven.

V nadaljevanju podajamo potek snovanja organizacije izvršilnega procesorja.

1. IZHODIŠČA ZA DEFINIRANJE ORGANIZACIJE IZVRŠILNEGA PROCESORJA

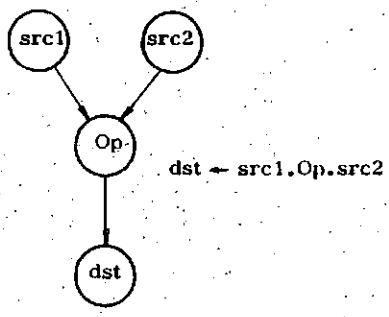
Organizacija izvršilnega procesorja je podrejena pospešenemu izvajanju operacij, ki so določene s kompleksno instrukcijsko množico. S stališča instrukcijskega procesorja, je izvršilni procesor samo razmeroma zmogljiv operator, ki mu instrukcijski procesor posreduje kodo zahtevane operacije in vrednosti parametrov (operande). izvršilni procesor pa vrne rezultat zahtevane operacije.

Analiza instrukcijske množice pokaže, da mora biti izvršilni procesor univerzalnega tipa, da prevladujejo tri operandne operacije in da je potrebna interpretacija večine operacij, ki so specificirane s kompleksno instrukcijsko množico.

Na podlagi opisanih lastnosti lahko definiramo začetne približke k organizaciji izvršilnega procesorja. Tako lahko prvi zahtevi po univerzalnosti priredimo operator univerzalnega tipa, trioperandni tip operacij pa ponazorimo po sliki 1.1.

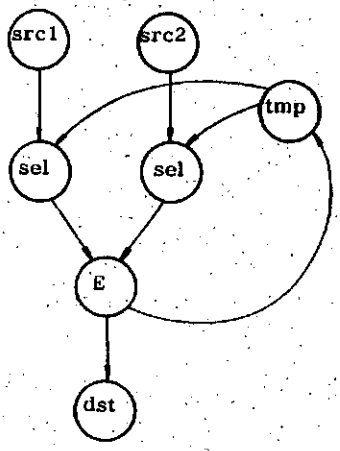
Oznake na sliki 1.1 pomenijo:

src 1 - prvi operand
src 2 - drugi operand
Op - univerzalni operator
dst - tretji operand (rezultat).



Slika 1.1: Začetni približek k organizaciji izvršilnega procesorja

Shema na sliki 1.1 izpolnjuje samo prvi dve zahtevi iz naše specifikacije. Zahteva po interpretaciji operacij $Op = \{Op_i \mid i = 1, 2, \ldots, n\}$ namreč predpostavlja, da lahko poljubno operacijo Op_i iz instrukcijske množice interpretiramo delomá ali v celoti z operacijami iz množice elementarnih operacij $E = \{e_j \mid j = 1, 2, \ldots, m\}$. Sedaj lahko shemo na sliki 1.1 dopolnimo tako, kot to podaja slika 1.2.



Slika 1.2: Dopolnjen model, ki omogoča interpretacijo operacij

Oznake na sliki 1.2 pomenijo:

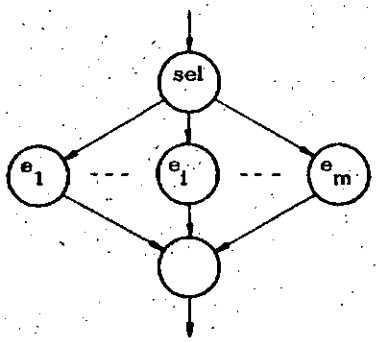
sel - selektorska operacija, ki izbere za operacijo operand iz src k ali tmp

tmp - nabor parametrov, ki omogočajo interpretacijo operacij iz Op. $tmp = \{tmp_e \mid e=1,2,\ldots,q\}$

S tem, da smo operacije iz Op nadomestili z operacijami iz E smo seveda predpostavili nek krmilni mehanizem, ki poskrbi, da se operacije iz Op interpretirajo z operacijami iz E. Po predpostavki lahko tedaj operacijo Op_i ponazorimo z eno ali več alternativnimi sekvencami, ki sestojijo iz operacij $\{e_j\}_{j=1,\ldots,m}$. Sekvencioniranje operacij iz E, tako da bodo ponazorile operacije iz Op, pa je naloga krmilnega mehanizma, ki ga bomo definirali kasneje.

Pri takšnem konceptu moramo tedaj za vsako operacijo Op_i, ki jo specificira ustrezna sekvenca sestavljena z elementi množice E, poskrbeti za selekcijo ustreznih operacij iz E. Če takšno sekvenco ponazorimo kot niz $\langle e_i \rangle$ moramo v vsakem koraku iz množice E izbrati ravno tisto operacijo, ki jo zahteva pripadajoči element operacijskega niza.

Začetni približek za takšno operacijo selekcije podaja slika 1.3.



Slika 1.3: Začetni približek k selekciji operacij iz E

Če posameznim elementom množice E priredimo selekcijske kode sel_i $i=1,2,\ldots,m$ lahko zapišemo tudi začetno krmilno enačbo selekcijske operacije

$$\text{izbrana operacija} = sel1 \wedge e_1 \vee sel_2 \wedge e_2 \vee \ldots sel_m \wedge e_m$$

kjer velja: $i \in \{1, 2, \ldots, m\}$,

$j \in \{1, 2, \ldots, m\}$,

$$(\forall i \neq j) \rightarrow sel_i \wedge sel_j \ .$$

Za izpolnjevanje performančnih zahtev takega operatorja, je potrebno grupiranje operacij iz E v tri osnovne skupine: osnovne aritmetične in logične operacije; operacije pomika, rotacije, maskiranja in ekstrakcije; operacije množenja. Šele tako lahko izpolnimo časovne zahteve, saj zahteva vsaka skupina operacij specifično organizacijo pripadajočega operatorja. Sedaj velja:

E = rot U mul U alu

kjer je:

rot - skupina operacij pomika, rotacije ...

mul - skupina operacij množenja

alu - skupina osnovnih aritmetičnih in logičnih operacij.

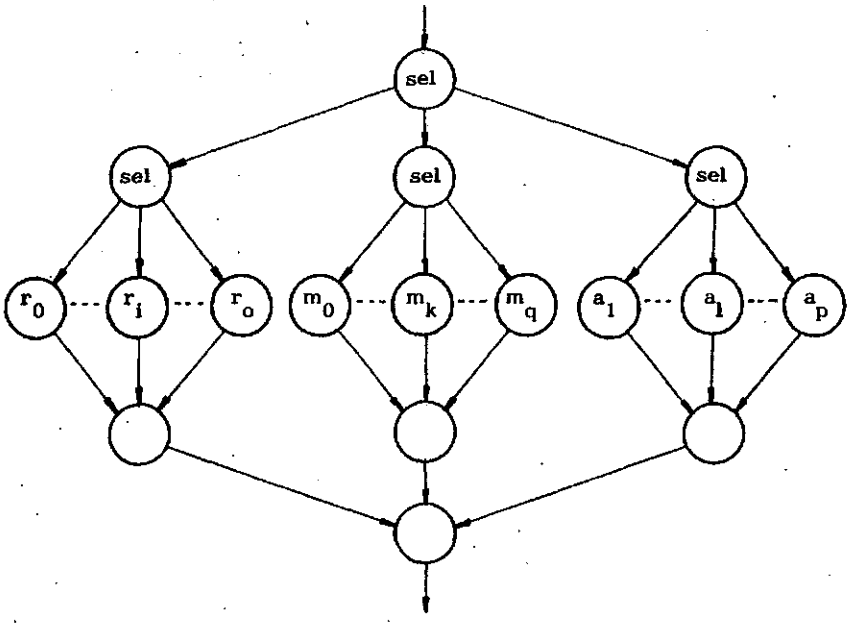
Ker se lahko v sistemski časovni enoti izvede naenkrat samo ena operacija iz izbrane skupine operacij lahko model krmiljenja že nekoliko dopolnimo po sliki 1.4.

Oznake na sliki pomenijo:

$\{r_i \mid i = 1, 2, \ldots, o\}$ = rot

$\{in_k \mid k = 1,2,\ldots,q\}$ = mul

$\{a_l \mid l = 1,2,\ldots,p\}$ = alu.



Slika 1.4: Podrobnejša krmilna shema za izbiro operacije

Glede na sliko 1.4 lahko prilagodimo tudi krmilno enačbo, saj smo vpeljali dva nivoja selekcije.

izbrana skupina operacij = sel 1. sk. $\wedge$ rot $\vee$

$\vee$sel 2. sk. $\wedge$mul $\vee$sel. 3. sk. $\wedge$alu

sel i. sk. - izbira skupine operacij

izbrana operacija 1. sk. = sel $1 \wedge r_1 \vee$ sel $2 \wedge r_2 \vee$ ....

$\ldots \vee$sel $o \wedge r_o$

izbrana operacija 2. sk. = sel $1 \wedge m_1 \vee$sel $2 \wedge m_2 \vee$ ...

$\ldots \vee$sel $q \wedge m_q$

izbrana operacija 3. sk. = sel $1 \wedge a_1 \vee$ sel $2 \wedge a_2 \vee$ ...

$\ldots \vee$sel $p \wedge a_p$

V celoti pa velja:

izbrana operacija = sel 1. sk. $\wedge$(sel $1 \wedge r_1 \vee$ ...

$\ldots \vee$sel $o \wedge r_o$)

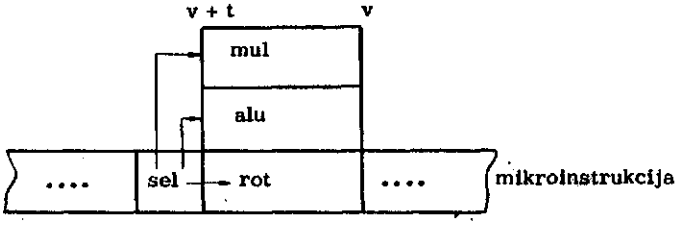
sel 2.sk. $\wedge$(sel i $m_1 \vee \ldots \vee$sel $q \wedge m_q$)

sel 3.sk. $\wedge$(sel $1 \wedge a_1 \vee \ldots \vee$sel $p \wedge a_p$)

Omenimo, da lahko takšno enačbo realiziramo z bralnimi pomnilniki. Tako definirana krmilna enačba ima tudi zanimivo lastnost s stališča mikroprogramske realizacije. V mikroinstrukcijo lahko namreč vpeljemo vertikalno funkcionalnost polj po sliki 1.5.

S tem smo tudi utemeljili enakost selekcijskih kod sel_i v operacijskih skupinah alu, mul, rot.

Čeprav razgradnja modela s tem še ni zaključena, smo



Slika 1.5: Zgradba polja mikroinstrukcije

osnovne karakteristike snovanja univerzalnega operatorja že zajeli.

2. DEFINICIJA INTERNE POMNILNE STRUKTURE IZVRŠILNEGA PROCESORJA

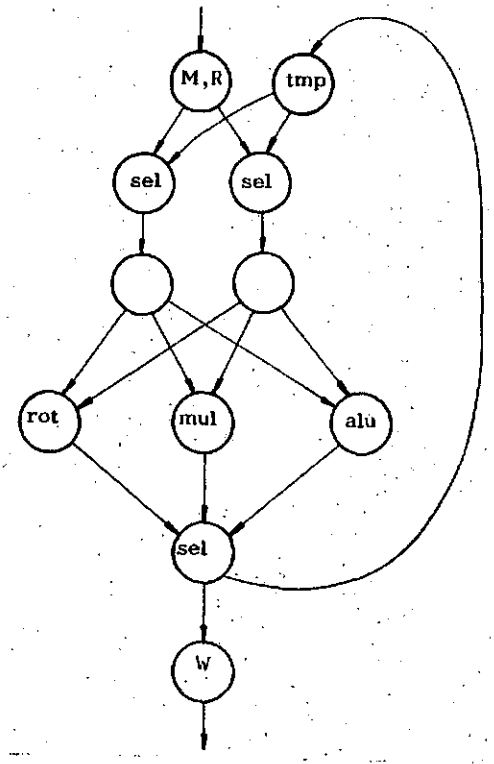
Za izpeljavo interne pomnilne strukture izvršilnega procesorja vzamemo model izvršilnega procesorja, ki ga podaja slika 2.1.

Pomen oznak je:

M, R - pomnilna struktura v katero vpisuje operande instrukcijski procesor

W - pomnilna struktura v katero vpisuje rezultate izvršilni procesor

Čeprav je shema pomnilne strukture na videz nepregledna, lahko krmilni sistem zelo jasno definiramo. Trioperandni tip operacij namreč zahteva v enem strojnem ciklu tri adrese operandov in sicer v začetku strojnega cikla čitanje do dveh izvornih operandov in ob koncu strojnega cikla destinacijsko adreso rezultata. Tako lahko pomnilni strukturi priredimo triadresni pomnilnik iz katerega lahko hkrati beremo do dva operanda, vanj pa vpišemo en operand. Operacije pomnilne strukture defi-

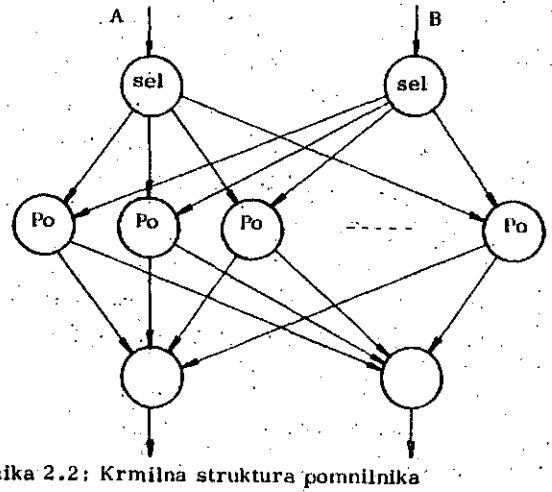


Slika 2.1: Organizacija izvršilnega procesorja

niramo takole:

read A, read B = R_{AB}
read A = R_A
read B = R_B
write B = W_B

Ker se čitanje in vpis časovno ne prekrivata, lahko preko adresne strukture B krmilimo čitanje in vpis operacij. kar nekoliko poenostavi strukturo pomnilnika. Pomnilno strukturo lahko tedaj ponazorimo kot n zaporednih lokacij. določenih z adresami. Nad vsako lokacijo lahko izvedemo operacijo čitanja ali vpisa. Zgoraj definiramo množico operacij nad lokacijami pomnilnika označimo s Po. Slika 2.2 podaja tedaj predlagano krmilno



Slika 2.2: Krmilna struktura pomnilnika

shemo interne pomnilne strukture.

Krmilni enačbi A in B strani se glasita:

Operacija A = sel 1$\wedge R_A \vee$ sel 2$\wedge R_A \vee \ldots \vee$ sel n$\wedge R_A$
$= R_A \wedge$ (sel 1$\vee$ sel 2$\vee \ldots \vee$ sel n)

Operacija B = sel 1$\wedge (R_B \vee W_B) \vee$ sel 2$\wedge (R_B \vee W_B) \vee \ldots$
$\ldots \vee$ sel n$\wedge (R_B \vee W_B)$
$= (R_B \vee W_B) \wedge$ (sel 1$\vee$ sel 2$\vee \ldots \vee$ seln).

Ker imamo opravka s tremi strukturami M, R, tmp, W definiramo nad vsako strukturo naslednje operacije:

M, R - R_A, R_B, R_{AB}
tmp - R_A, R_B, R_{AB}, W_B
W - W_B.

Da preprečimo hkraten vpis in čitanje postavimo:

WE - odobreno čitanje,
$\overline{WE}$ - odobren vpis.

Sedaj lahko krmilni enačbi zapišemo določneje:

operacija A = WE$\wedge$ sel M,$R_A \wedge$(sel1$\vee$ sel2$\vee \ldots \vee$ sel p)
$\vee$ WE$\wedge$ sel $tmp_A \wedge$(sel1$\vee$ sel2$\vee \ldots \vee$ sel q)

operacija B = WE$\wedge$ sel M, $R_B \wedge$(sel1$\vee$ sel2$\vee \ldots \vee$ sel p)
$\vee$ WE$\wedge$ sel $tmp_B \wedge$(sel1$\vee$ sel2$\vee \ldots \vee$ sel q)
$\vee \overline{WE} \wedge$ sel $tmp_B \wedge$(sel1$\vee$ sel2$\vee \ldots \vee$ sel q)
$\vee \overline{WE} \wedge$ sel $W_B \wedge$(sel1$\vee$ sel2$\vee \ldots \vee$ sel r)

Slika 2.3 podaja tako definirano krmilno strukturo pomnilnika izvršilnega procesorja.

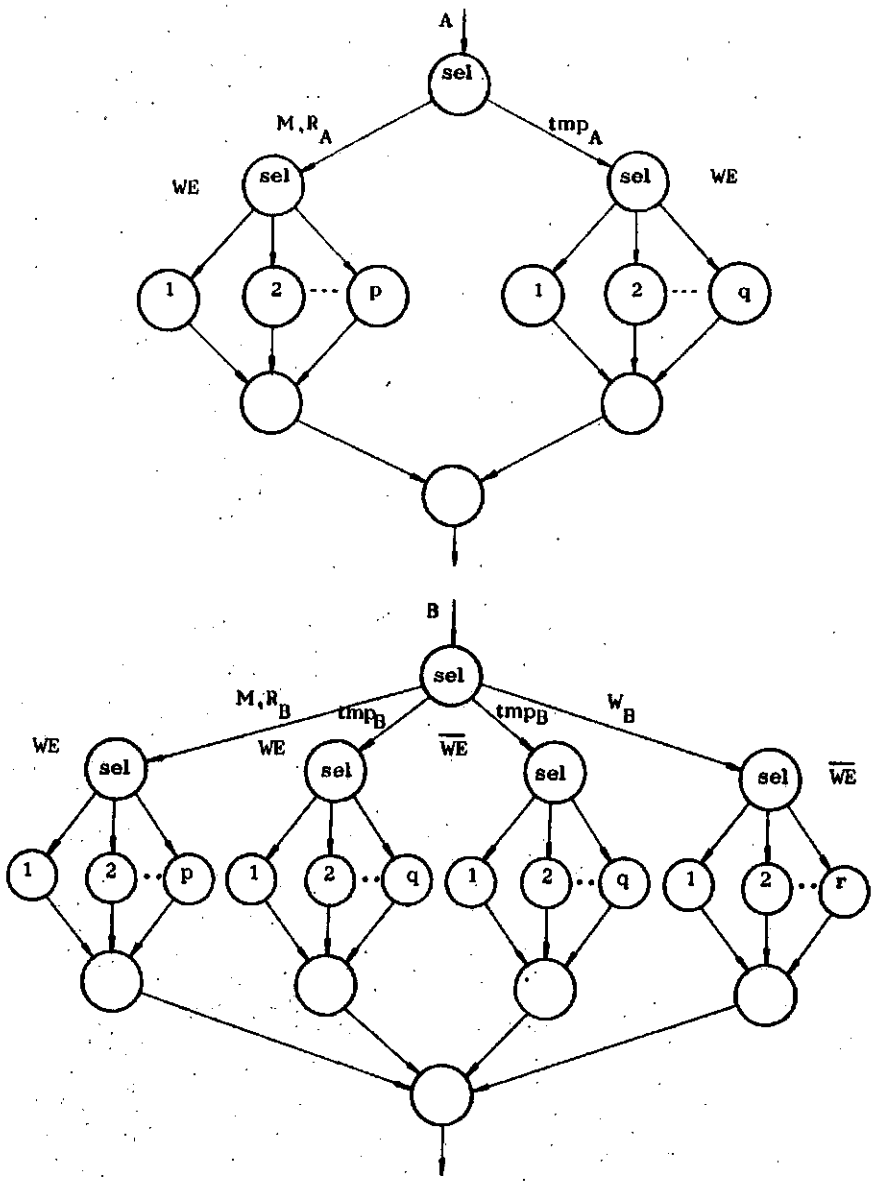
Organizacijo interne pomnilne strukture podaja slika 2.4, tako kot jo vidi izvršilni procesor.

Razgradnja interne pomnilne strukture izvršilnega procesorja s tem še ni zaključena, vendar pa že dosedanji postopek dovolj dobro ilustrira potek snovanja.

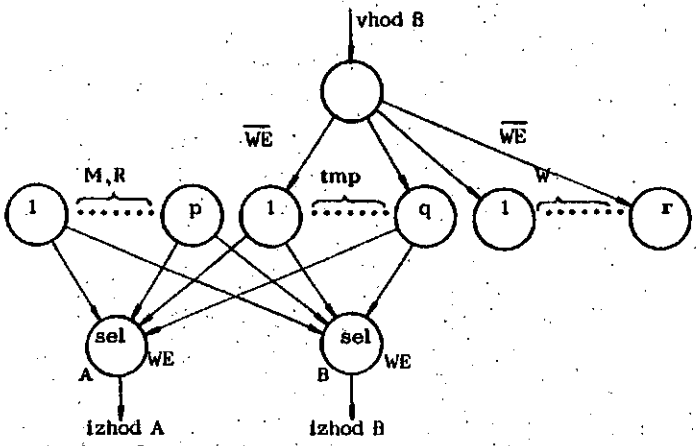
3. ORGANIZACIJA KRMILNE ENOTE IZVRŠILNEGA PROCESORJA

Organizacijo krmilne enote izvršilnega procesorja določimo s pomočjo analize lastnosti diagramov poteka, ki opisujejo operacije, specificirane z instrukcijsko množico. Diagrami poteka predstavljajo mikroprograme zapisane na registerskem nivoju v meta jeziku. Osnovna značilnost mikroprogramov je velika razvejanost. Zaradi časovnih zahtev je potrebno vejitve na eno izmed več možnih paralelnih vej izvesti v enem strojnem ciklu.

Naslednja značilnost je, da se identične skupine mikrooperacij pogosto ponavljajo v mikroprogramih sicer različnih instrukcij, kar omogoča definiranje primernih makroukazov ali podprogramov. Operacije v zankah so sicer v našem primeru manj pogoste, vendar se zaradi ča-



Slika 2.3: Krmilna struktura pomnilnika izvršilnega procesorja



Slika 2.4: Organizacija interne pomnilne strukture izvršilnega procesorja

sovnih zahtev izkaže smotrna realizacija zančnega mehanizma, ki omogoča tudi vgnezditev zanke.

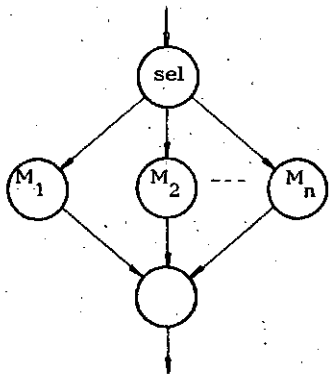
Zaradi razvejanosti mikroprogramov tedaj adresa naslednje mikroinstrukcije v splošnem ni enolično določena. Zato si prizadevamo mikroprogramski pomnilnik organizirati tako, da bi bilo možno vnaprej prebrati vse mikroinstrukcije, ki so možne naslednice mikroinstrukcije v izvajanju.

Prvi del naloge torej zahteva, da vsaki mikroinstrukciji v izvajanju določimo množico možnih naslednjih mikroinstrukcij. Drugi del naloge pa zahteva, da na podlagi statusov, ki se postavijo med izvajanjem tekoče mikroinstrukcije, izvedemo selekcijo izbrane mikroinstrukcije iz prebrane množice mikroinstrukcij. Takšno nalogo je možno rešiti s primernim asociativnim pomnilnikom.

V našem primeru smo ugotovili, da imajo vejitve na do štiri možne veje mnogo višjo frekvenco, kot vejitve na več kot štiri možne veje. Zato se odločimo, da vejitve na do štiri možne veje realiziramo s paralelnim branjem

štirih mikroinstrukcij, vejitve na več kot štiri veje pa klasično z modificiranjem naslednje adrese mikroinstrukcije z ALI funkcijo, vendar samo do štiri mikroinstrukcije natančno. Dokončna selekcija se opravi enako kot v primeru vejitve na eno izmed štirih možnih vej.

Izhodiščno krmilno shemo mikroprogramirane krmilne enote podaja slika 3.1.



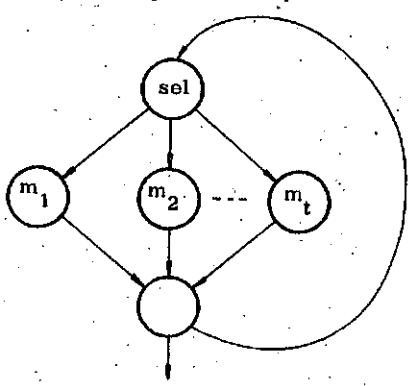
Slika 3.1: Izhodiščni model mikroprogramirane krmilne enote

Začetna predpostavka je, da je mikroprogramov toliko, kot je operacij v množici operacij Op. Na podlagi slike 3.1 lahko zapišemo enačbo za izbiro mikroprograma glede na zahtevano operacijo Op_i.

Izbira M_i = sel1 $\wedge M_1 \vee$ sel2 $\wedge M_2 \vee \ldots$
sel i $\wedge M_i \vee \ldots \vee$ sel n $\wedge M_n$

Pravila, po katerih teče izbira mikroprogramov za izvajanje, so seveda določena na nivoju instrukcijske množice. Model, s katerim lahko ponazorimo tako operacijo, je pomnilnik, razdeljen na podmnožice lokacij, kjer je na vsaki podmnožici lokacij zapisan mikroprogram.

Model krmiljenja izbranega mikroprograma M_i (selekcijo mikroinstrukcij) lahko ponazorimo po sliki 3.2.



Slika 3.2: Izbira mikroinstrukcij v mikroprogramu

Pravila, po katerih se izbirajo mikroinstrukcije, določimo sami.

Krmilna enačba za sliko 3.2 se glasi:

izbira m_i = sel 1 $\wedge m_1 \vee$ sel 2 $\wedge m_2 \vee \ldots$
sel i $\wedge m_i \vee \ldots \vee$ sel t $\wedge m_t$.

Doslej razvit krmilni model mikroprogramske krmilne enote podaja slika 3.3.

Za izbiro mikroinstrukcij znotraj mikroprograma izberemo princip "next" polja v mikroinstrukciji. Iz slike 3.3 lahko razberemo zanimivo lastnost takega načina izbire naslednje mikroinstrukcije. Z izbiro mikroprograma M_i omejimo gibanje kazalca, ki izbira mikroinstrukcije na razmeroma omejeno področje lokacij mikroprogramskega pomnilnika. Iz tega pa sledi, da je lahko velikost "next" polja znatno krajša od dolžine celotnega adresnega prostora mikroprogramskega pomnilnika. Pri tem je značilna naslednja lastnost takega pristopa: za izbiro mikroprograma lahko uporabimo identični mehanizem, kot velja za vejitveni krmilnik na nižjem abstraktnem nivoju (nivo mikroinstrukcij v izbranem mikroprogramu). Mehanizem razvijemo v poslošeni obliki.

Adreso ponazorimo kot binarni vektor:

$A = \langle a_{n-1}, a_{n-2}, \ldots, a_1, a_0 \rangle$

Bazno adreso, ki je osnova za določitev adrese A zapišemo:

$BA = \langle a_{n-1}, a_{n-2}, \ldots, a_i, \underbrace{0, 0, \ldots, 0}_{i} \rangle$

Pomik na bazno adreso ("next" polje, vejitev) izrazimo v obliki:

$PA = \langle 0, 0, \ldots, 0, a_{i-1}, a_{i-2}, \ldots, a_1, a_0 \rangle$

Adresa A je sedaj določena z relacijo:

$A = BA \,.\, ALI \,.\, PA$

$A = \langle a_{n-1}, a_{n-2}, \ldots a_i, 0, \ldots, 0 \rangle \,.\, ALI \,.$
$\langle 0, 0, \ldots, a_{i-1}, a_{i-2}, \ldots, a_1, a_0 \rangle$

Alternativni zapis zgornja relacije, ki je primeren za izvedbo v logiki s tremi stanji lahko zapišemo:

$A = 1BA' \, 1PA$, kjer je $'$ operacija konkatenacije nizov

$1BA = \langle a_{n-1}, a_{n-2}, \ldots, a_i \rangle$ in

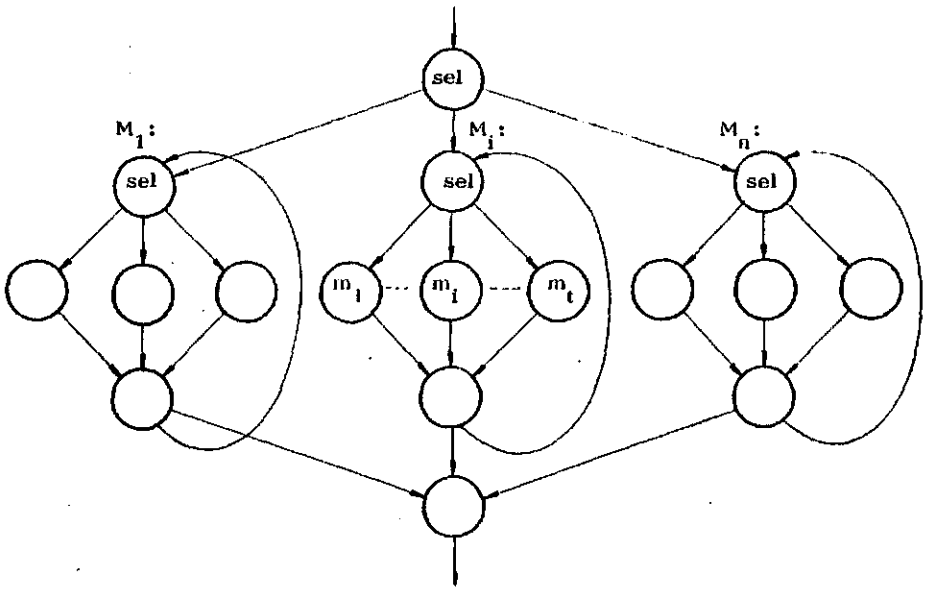
$1PA = \langle a_{i-1}, a_{i-2}, \ldots, a_1, a_0 \rangle$.

Glede na izhodiščne predpostavke tega razdelka opisani postopek tvorbe adrese za vejitveni krmilnik v našem primeru nekoliko dopolnimo. Bazna adresa ima v našem primeru naslednjo obliko:

$BA = \langle a_{15}, a_{14}, \ldots, a_6, 0, 0, \ldots 0 \rangle$.

Izbiro mikroinstrukcije, do štiri mikroinstrukcije natančno podaja prvi pomik:

$1PA = \langle 0, 0, \ldots, 0, a_5, a_4, a_3, a_2, 0, 0 \rangle$,

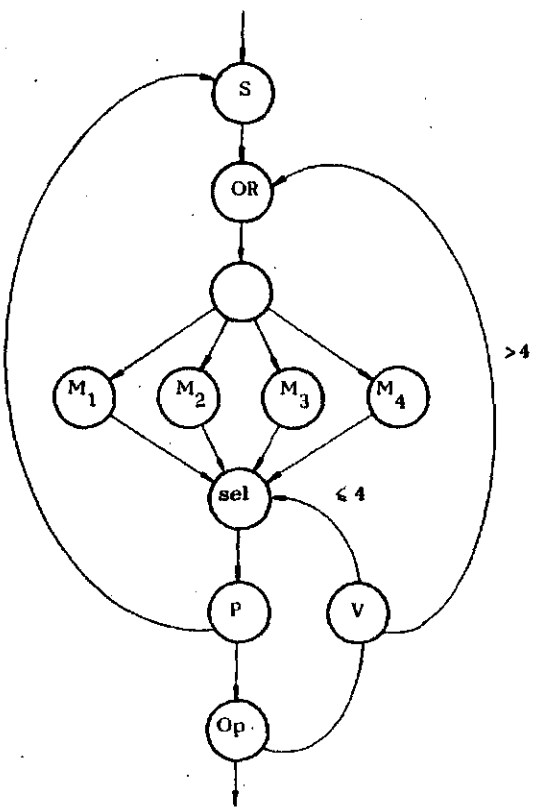


Slika 3.3: Model mikroprogramske krmilne enote izvršilnega procesorja

dokončno izbiro mikroinstrukcije pa določa drugi pomik (izbira ene izmed štirih pomnilnih matrik):

$2PA = \langle 0, 0, \ldots\ldots, 0, a_1, a_0 \rangle$.

Izvedbo tako zasnovane mikroprogramske krmilne enote podaja slika 3.4.



Slika 3.4: Organizacija mikroprogramske krmilne enote izvršilnega procesorja

Oznake na sliki 3.4 pomenijo:

S - sekvencer adres mikroinstrukcij

OR - ALI funkcija

M_i - mikroprogramski pomnilniki

sel - operacija selekcije

P - prekrivalni register

Op - univerzalni operator

V - krmilnik vejitev

Razgradnja mikroprogramske krmilne enote s tem še ni zaključena, podane pa so osnovne karakteristike za naš primer.

4. SELEKTORSKA OPERACIJA

Realizacija selekcijske operacije:

sel (sel . koda):

sel1 → operacija 1 (mikroprogram 1, mikroinstrukcija 1, ...)

sel2 → operacija 2 (.....)

.

.

.

seln → operacija n (.....)

izvedemo s pomočjo izraza:

izbira = $sel1 \wedge m_1 \vee sel2 \wedge m_2 \vee \ldots \vee seln \wedge m_n$.

V zgornjem izrazu je:

$m_i = \langle u_a, u_{a-1}, \ldots, u_1, u_o \rangle$

binarni vektor, kjer je:

$u_j \in \{0, 1\}$.

sel. koda je preslikava definirana takole:

sel. koda: $sel_i \longrightarrow \ldots$,

sel. koda: $\langle s_b, s_{b-1}, \ldots s_1, s_0 \rangle \rightarrow \wedge$,

kjer je:

$s_k \in \{0, 1\}$ in $\wedge \in \{0, 1\}$.

Selekcijsko kodo pogosto zapišemo kot logični produkt elementov s_k ob zahtevi, da je lahko pravilen en sam produkt v selektorski operaciji.

Zgled:
Izbiro mikroinstrukcije m_i v mikroprogramu M_j izvedemo takole:

sel. koda = 1 za mikroinstrukcijo m_i.

Za vse ostale mikroinstrukcije iz $M_j - \{m_i\}$ ima selekcijska koda vrednost 0. Tedaj lahko zapišemo:

izbrana $m_i = 0 \wedge m_1 \vee 0 \wedge m_2 \vee \ldots \vee 1 \wedge m_i \vee \ldots \vee 0 \wedge m_n$

izbrana $m_i = 1 \wedge m_i$

izbrana $m_i = m_i$

izbrana $m_i = \langle u_a, u_{a-1}, \ldots, u_1, u_0 \rangle_i$.

5. ZAKLJUČEK

Postopek navzdolnjega snovanja je ilustriran na primeru implementacije izvršilnega procesorja, ki izvaja operacije določene s kompleksno instrukcijsko množico. Za ponazoritev rezultatov so uporabljeni usmerjeni grafi in matematična logika.

Izhodiščni približek je bil, da lahko model procesorja specificiramo s selektorsko operacijo, ki ma toliko vej, kot je instrukcij. Vsaka veja tako realizira eno izmed operacij specificiranih s kompleksno instrukcijsko množico.

Zaradi interpretacije teh operacij z bolj elementarnimi operacijami, smo bili prisiljeni definirati lokalne parametre, katerih vrednosti hrani delovni pomnilnik. Posledica interpretacije je tudi mikroprogramska krmilna enota, ki je za programerja na nivoju kompleksne instrukcijske množice nevidna.

Ilustriran postopek snovanja je bil uporabljen pri izdelavi modela za celotni 32-bitni računalniški sistem, ki izvaja kompleksno instrukcijsko množico.

Osnovna prednost predlaganega postopka snovanja so mnogo krajši časi snovanja, pregledna struktura sistema in večja zanesljivost realizacije, saj je nevarnost neodkritih napak mnogo manjša kot pri klasičnih postopkih snovanja.

6. LITERATURA

1. M. Gerkeš, M. Družovec, M. Pernek: Aplikacija bipolarnega mikroprocesorja, poročilo o delu za leto 1983, raziskovalna naloga št. 03-2570/796-83, Visoka tehniška šola, VTO Elektrotehnika, Maribor 1983.
2. S. P. Kartashev, S. I. Kartashev, eds.: Designing and Programming Modern Computers and Systems, Vol. 1 LSI Modular Computer Systems, Prentice-Hall, Inc., Englewood Cliffs, N. J., 1982
3. J.K.Iliffe: Advanced Computer Design, Prentice-Hall International, Inc., London, 1982