LOGIČNI MODELI RAČUNALNIŠKIH STRUKTUR II

MAKSIMILJAN GERKEŠ
TEHNIŠKA FAKULTETA, MARIBOR
VTO ELEKTROTEHNIKA, RAČUNALNIŠTVO IN INFORMATIKA
METALNA, MARIBOR
TOZD TOVARNA INVESTICIJSKE OPREME

UDK : 681.3.517.11/12

POVZETEK: Poljuben označen usmerjen graf z enim začetnim in končnim vozliščem, pri katerem se vse poti sklenejo med tema vozliščema, je preoblikovan v vase zaključeno selektorsko operacijo, tudi če vsebuje zanke s končno mnogo ponovitvami. Za izgradnjo modelov sekvenčnih in mikroprogramiranih strojev so izdelani postopki dekompozicije s pomočjo karakterističnih lastnosti, kar lahko vodi do realizacije, ki ima več sekvenčnih oz. mikroprogramiranih strojev kot začetni model. Definiran je posplošen model operatorja s selektorsko operacijo in ga je možno razgraditi do poljubnih detaljev. Izdelani so modeli konkretnih računalniških struktur.

ABSTRACT: LOGICAL MODELS FOR COMPUTER STRUCTURES II. Any labelled directed graph with single input and single output node in which all directed paths are terminated at the output node, can be modelled with a select operation in the self loop, even if loops with a finite number of repetitions are presented in the initial graph. For a model design of a sequential or microprogrammed maschine procedures were defined, that can lead to realizations with some sequential or microprogrammed maschines, altough initial model was single maschine model. Abstract operator with the ability to decompose it into any required detail is defined. Models for well known computer structures were built.

UVOD:

Poljuben označen usmerjen graf, ki ima eno začetno in eno končno vozlišče in v katerem vse usmerjene poti vodijo od začetnega h končnemu vozlišču lahko preoblikujemo v graf vase zaključene selektorske operacije. Takšno preoblikovanje usmerjenega grafa je možno tudi, če graf vsebuje operacije v zankah, ki imajo končno število ponovitev. Iz tega razloga lahko poljuben program, mikroprogram, ... realiziramo v obliki sekvenčnega stroja oz. mikroprogramsko krmiljenega stroja. Kadar pa vase zaključena selektorska operacija rabi kot izhodiščni model za snovanje izbrane računalniške strukture lahko s postopki dekompozicije preoblikujemo to operacijo v sestavljeno vase zaključeno selektorsko operacijo, katere realizacija lahko v splošnem zahteva izgradnjo več povezanih sekvenčnih strojev ali mikroprogramsko krmiljenih strojev.

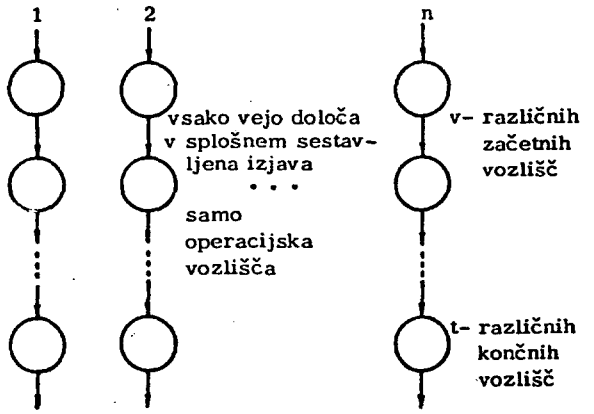
Pri dekompoziciji kompleksnih sekvenčnih strojev - kot so naprimer modeli procesorjev je smiselno uporabiti splošnejši pristop k dekompoziciji in sicer z uporabo karakterističnih lastnosti specifikacije, katerim podrejamo dekompozicijo izbrane strukture. Odstopanja od karakterističnih lastnosti pa obravnavamo kot izjeme, s katerimi korigiramo osnovno strukturo, dobljeno samo na podlagi upoštevanja karakterističnih lastnosti. Za snovanje operatorjev sekvenčnih oz. mikroprogramiranih strojev je uporabljen abstraktni pristop, ki omogoča definicijo posplošenega modela operatorja ter njegovo dekompozicijo do poljubnih detaljev. Predlagani so modeli konkretnih računalniških struktur, izgrajeni s pomočjo opisanih postopkov modeliranja od krmilnih enot preko pomnilnih struktur do operatorjev.

1. PRETVORBA GRAFA OPERACIJ V GRAF VASE ZAKLJUČENE SELEKTORSKE OPERACIJE

Kot iztočnico vzemimo označen usmerjen graf z enim začetnim in enim končnim vozliščem. Graf naj ne vsebuje operacij v zankah. Vse poti, ki izhajajo iz začetnega vozlišča morajo končati v končnem vozlišču. Vsem operacijskim vozliščem (Op_i, i = 1, 2, ..., n) pripišemo izjave Q_j, ki specificirajo naslednja operacijska vozlišča. Vsaki usmerjeni poti, ki povezuje začetno in končno vozlišče lahko tedaj priredimo graf samih sekvenčno povezanih operacij. Tak graf - sekvenčno operacijo pa lahko po [1] preoblikujemo v vase zaključeno selektorsko operacijo.

Vzemimo, da najdemo v označenem usmerjenem grafu G n-usmerjenih poti, ki vodijo od začetnega h končnemu vozlišču. Tedaj lahko narišemo n-sekvenčnih operacij, s katerimi ponazorimo prehode po grafu G v izbranih trenutkih. Začetna vozlišča teh grafov (sekvenčnih operacij) niso enaka, če je začetno vozlišče grafa selektorska operacija; enaka so, če je začetno vozlišče opera-

cijsko vozlišče. Podobno velja za končna vozlišča, da niso vsa enaka, če je graf zaključen s selektorsko operacijo, drugače so vsa vozlišča enaka. Na sliki 1.1 je podan splošen primer, če je število neenakih začetnih in končnih vozlišč različno. Poljubno operacijsko vozlišče Op_i je določeno z izjavo Q_i, ki jo pridružimo predhodnemu vozlišču Op_j in v bistvu določa usmerjeno vejo od Op_j k Op_i, ali s konjunkcijo $Q_i \wedge P_{ij} \wedge P_{kl} \wedge \ldots \wedge P_{rs}$, če vodi k vozlišču Op_i usmerjena pot preko več ali ene selektorskih operacij.

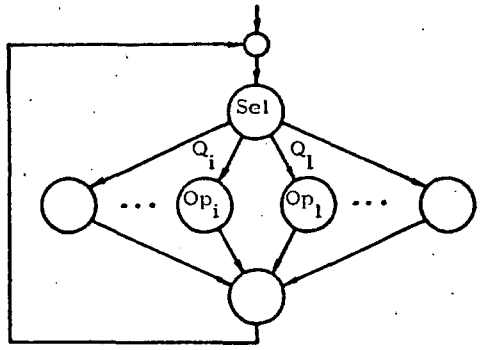


Slika 1.1: Ponazoritev usmerjenega grafa G z n-usmerjenimi grafi - sekvenčnimi operacijami

Po opisanem postopku pretvorbe se lahko nekatere operacije v vase zaključeni selektorski operaciji ponovijo tudi do n-krat, saj lahko isto vozlišče večkrat "prehodimo" po različnih usmerjenih poteh. Tedaj reduciramo veje, ki se v vase zaključeni selektorski operaciji ponavljajo takole:

$$Q_i \wedge Op_i \vee Q_i \wedge Op_i \vee \ldots \vee Q_i \wedge Op_i = Q_i \wedge Op_i \qquad (1.1)$$

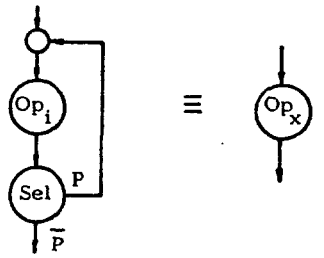
Operacije, ki se v vase zaključeni selektorski operaciji ponavljajo (1.1) nadomestimo z eno samo operacijo. Slika 1.2 podaja zgled tako preoblikovanega grafa G, ki ustreza modelu sekvenčnega stroja opisanega v [1] .



Slika 1.2: Usmerjen graf G preoblikovan v vase zaključeno selektorsko operacijo

1.1. Preoblikovanje operacij v zanki

Sedaj sprostimo pogoj, da usmerjen graf G ne sme vsebovati operacij v zanki. Obdržimo pa omejitev, da morajo imeti zanke končno mnogo ponovitev. Tvorimo nov graf, v katerem operacije v zanki ponazorimo kot vozlišča z enim vhodom in enim izhodom. Slika 1.3 podaja zgled takšne pretvorbe, kjer je lahko operacija Op_i tudi sestavljena operacija, ki vsebuje nove operacije v zanki.

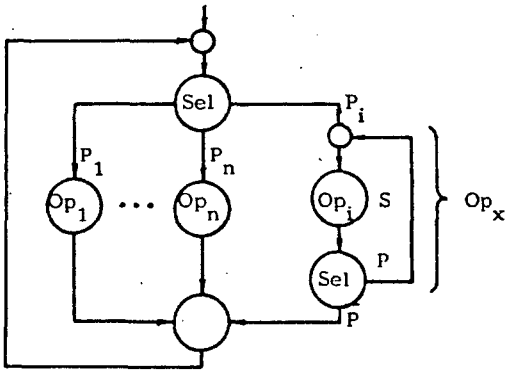


Slika 1.3: Pretvorba operacije v zanki v vozliščno operacijo

Tako dobljen graf sedaj ustreza pogojem, da ga lahko preoblikujemo v vase zaključeno selektorsko operacijo. Ugotovimo lahko, da vsaka operacija v zanki, če govorimo o njeni logični realizaciji v bistvu predstavlja avtonomen sekvenčni stroj. Pri logični realizaciji usmerjenega grafa, ki vsebuje več zank, bi tedaj morali izdelati še toliko sekvenčnih strojev, kolikor je operacij v zanki.

Takšen pristop pa je v splošnem preveč tog, zato skušamo operacije v zanki prevesti v koncept ene vase zaključene selektorske operacije oz. enega sekvenčnega stroja. Takoj pripomnimo, da končni cilj ni vedno ena sama vase zaključena selektorska operacija oz. en sekvenčni stroj, ampak da želimo imeti možnost, da sami odločamo o številu vase zaključenih selektorskih operacij oz. o organizaciji sistema.

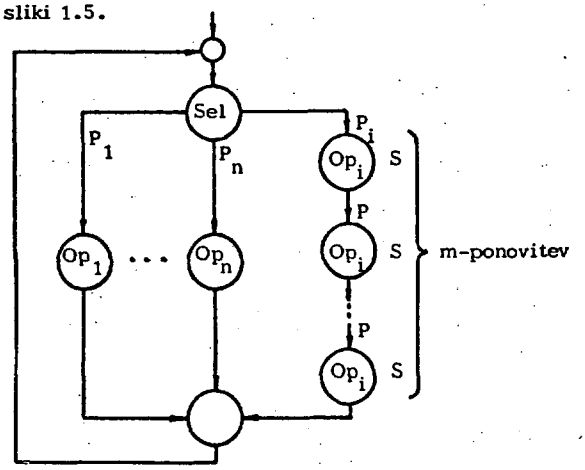
Vzemimo, da smo označen usmerjen graf G, ki vsebuje operacijo Op_i v zanki preoblikovali v vase zaključeno selektorsko operacijo, ki jo podaja slika 1.4.



Slika 1.4: Graf G preoblikovan v vase zaključeno selektorsko operacijo

Na sliki 1.4 so P_1, P_2, ..., P_n v splošnem sestavljene izjave oblike: Q_i, $Q_j \wedge P_1$, ...

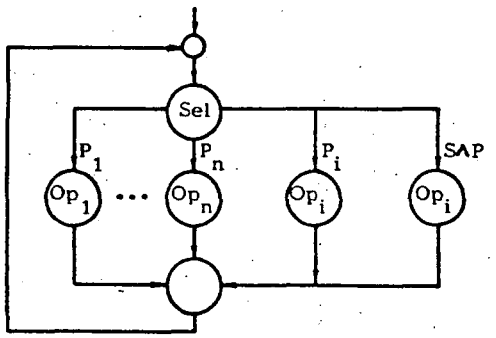
Sedaj pa opazujmo graf s slike 1.4 v trenutku, ko se operacija Op_i ponovi m-krat. Tedaj lahko narišemo graf na sliki 1.5.



Slika 1.5: Graf G v primeru, ko se operacija Op_i ponovi m-krat

Sekvenčno operacijo v grafu s slike 1.5 lahko po [1] preoblikujemo v vase zaključeno selektorsko operacijo. Graf na sliki 1.5 tedaj preide v graf na sliki 1.6. Pravkar opisano pretvorbo zančne operacije lahko verificiramo tudi s popolno indukcijo in s tem potrdimo pravilnost pretvorbe.

Za graf na sliki 1.6 izdelajmo še simbolični zapis:



Slika 1.6: Preoblikovana zančna operacija

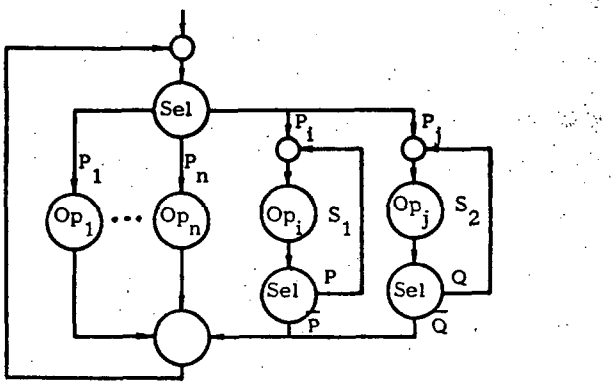
sel (S, P_j):

P_1 — Op_1
P_2 — Op_2
⋮
P_{i-1} — Op_{i-1}
P_{i+1} — Op_{i+1}
⋮
P_n — Op_n
P_i — Op_i
$S \wedge P$ — Op_i

kjer je za izjavo S (in tudi izjave Q_i, i=1, 2, ...) v praksi snovanja strojne opreme udomačen izraz stanje. Zančni operaciji Op_i smo izjavo S pripisali zato, da zagotovimo enoličnost selektorske operacije - samo ena izmed izjav P_1, P_2, ..., P_i, ... , P_n, P je lahko naenkrat pravilna.

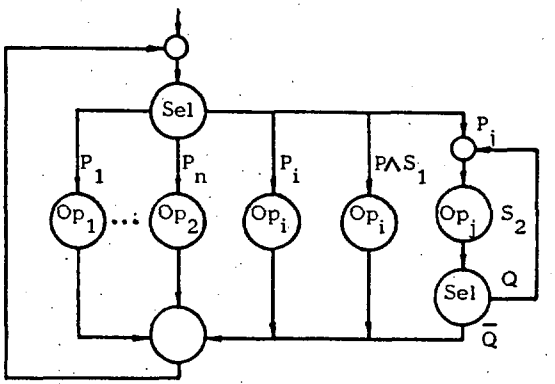
V ta namen vzemimo, da so izjave P_1, P_2,..., P_n izbrane kot konjunktivne kombinacije niza izjav p_{m-1}, p_{m-2},... ..., p_1, p_0 in velja $2^m - n \geq 1$.
Tedaj lahko za S izberemo iz preostalih konjunktivnih kombinacij iz niza p_{m-1}, p_{m-2},..., p_0.
Z zgornjim primerom smo pokazali, kako je mogoče z dodatno izjavo S ohraniti enoličnost selektorske operacije.

Če sta v usmerjenem grafu dve ali več zančnih operacij jih reduciramo postopoma - z nekaj izkušnjami pa lahko reduciramo tudi vse hkrati. Vzemimo primer, da sta nam pri preoblikovanju grafa G v vase zaključeno selektorsko operacijo preostali dve operaciji v zanki. Slika 1.7 podaja tak primer.



Slika 1.7: Preoblikovan graf G z dvema zančnima operacijama

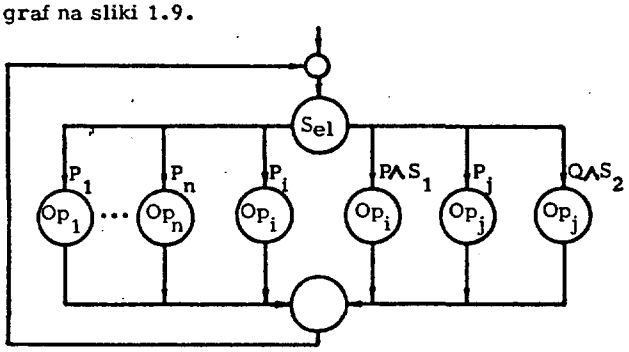
Najprej preoblikujemo zančno operacijo Op_i in pustimo zančno operacijo Op_j nedotaknjeno. Graf s slike 1.7 preide tedaj v graf na sliki 1.8.



Slika 1.8: Prvi korak preoblikovanja

Postopek ponovimo nad zančno operacijo Op_j in dobimo

graf na sliki 1.9.



Slika 1.9: Preoblikovan graf s slike 1.7

Zapišimo še selektorsko operacijo v simboličnem zapisu:

sel:

$P_1 \rightarrow Op_1$
$\vdots$
$P_{i-1} \rightarrow Op_{i-1}$
$P_{i+1} \rightarrow Op_{i+1}$
$\vdots$
$P_{j-1} \rightarrow Op_{j-1}$
$P_{j+1} \rightarrow Op_{j+1}$
$\vdots$
$P_n \rightarrow Op_n$
$P_i \rightarrow Op_i$
$S_1 \wedge P \rightarrow Op_i$
$P_j \rightarrow Op_j$
$S_2 \wedge Q \rightarrow Op_j$

Vzemimo, da so izjave $P_1, P_2, \ldots, P_i, \ldots, P_j, \ldots, P_n$ iz konjunktivnih kombinacij niza izjav:

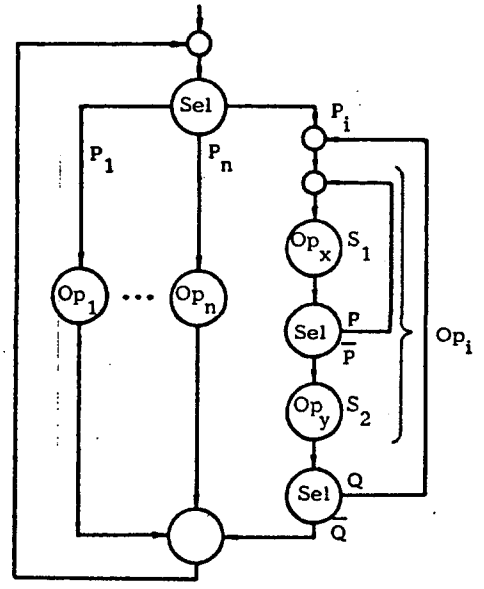
$P_{m-1}, P_{m-2}, \ldots, P_1, P_0$ in $2^m - n \geq 2$.

Tedaj lahko za S_1 in S_2 izberemo iz preostalih konjunktivnih kombinacij zgornjega niza.

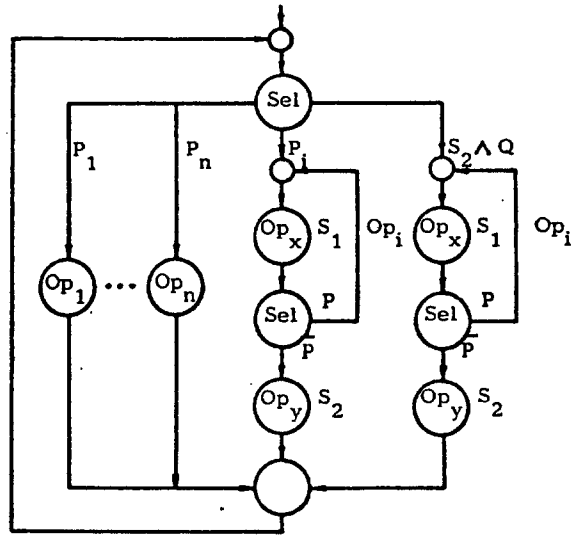
Obdelajmo še primer, ko zančna operacija v vase zaključeni selektorski operaciji tudi sama vsebuje zančno operacijo. Tako oblikovan graf G podaja slika 1.10.

Preoblikovanje grafa s slike 1.10 pričnemo tako, da si zanko določeno z izjavo Q zamislimo kot sestavljeno operacijo (Op_i). Tedaj preide graf v obliko na sliki 1.4, ki jo že znamo preoblikovati. Če opravimo postopek preoblikovanja, preide graf s slike 1.10 v obliko na sliki 1.11.

Ponovno si zamislimo zančni operaciji Op_x kot operaciji Op_i in dobimo graf, v katerem sta po dve sekvenčno povezani operaciji, ki ju lahko preoblikujemo po [1] v vase zaključeno selektorsko operacijo. S tem dobimo graf na sliki 1.12.

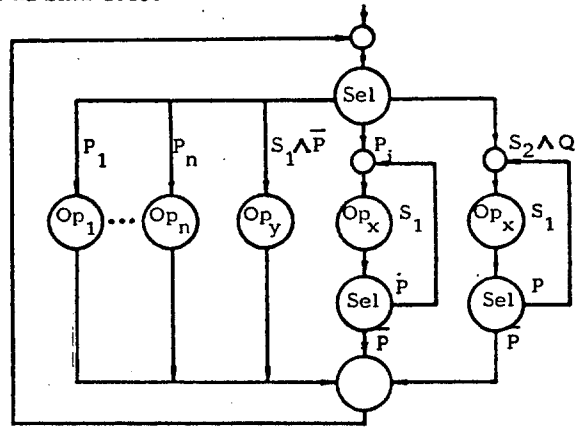


Slika 1.10: Graf G z vgnezdeno zančno operacijo

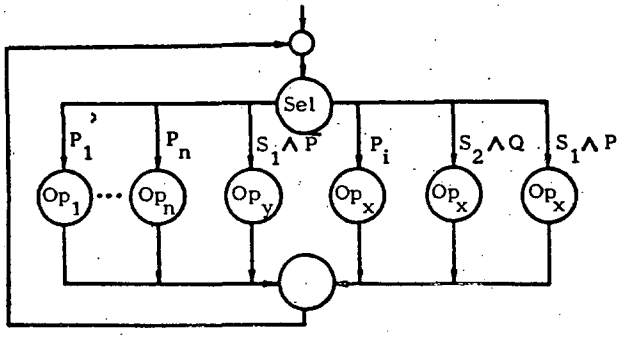


Slika 1.11: Prvi korak pri preoblikovanju vgnezdene zanke

Graf na sliki 1.12 pa je s stališča preoblikovanja zančnih operacij enak grafu na sliki 1.7, zato lahko narišemo kar končno obliko grafa vase zaključene selektorske operacije na sliki 1.13.



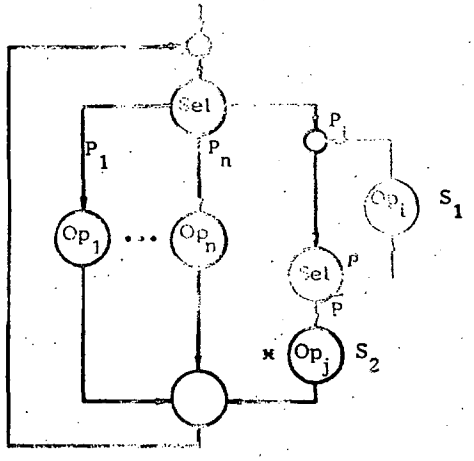
Slika 1.12: Drugi korak preoblikovanja vgnezdene zanke



Slika 1.13: Preoblikovan graf vgnezdene zanke

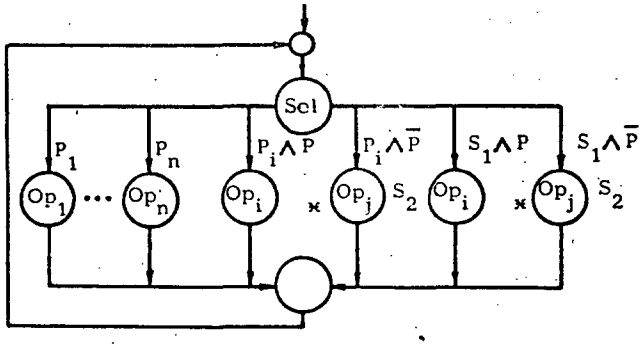
Posplošitev opisane transformacije na n-vgnezdenih zank je enostavna in lahko shajamo z doslej definiranimi transformacijami.

Opravimo pa še preoblikovanje grafa s slike 1.14.



Slika 1.14: Graf G z operacijo v povratni veji zanke

Grafu G smo kos s smiselno uporabo doslej opisanih transformacij. Zato lahko narišemo kar graf vase zaključene selektorske operacije na sliki 1.15.



Slika 1.15: Graf vase zaključene selektorske operacije za graf s slike 1.14

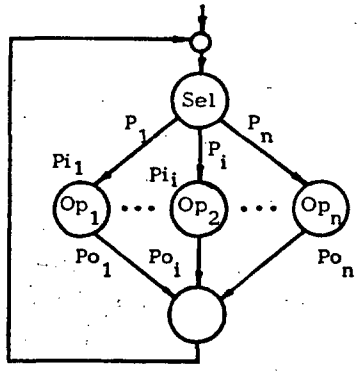
Z zvezdico označeno operacijo Op_j smo vrisali zato, ker lahko pogosto izvorni graf G preoblikujemo tako kot na sliki 1.14. Sicer bi morali vpeljati namesto vozlišča Op_j na sliki 1.15 vozlišče O_n, ki izvaja "prazno" operacijo.

Pri fizičnem snovanju moramo zagotoviti tudi inicializacijo operacij v naših grafih. To dosežemo s tem, da v vhodno vozlišče "pripeljemo" izjavo, ki izbere začetno vejo v vase zaključeni selektorski operaciji - sistem postavimo v začetno stanje.

2. DEKOMPOZICIJA VASE ZAKLJUČENE SELEKTORSKE OPERACIJE - SEKVENČNEGA STROJA; NAVZDOLNJE SNOVANJE

V tem razdelku se omejimo na dekompozicijo kompleksnih vase zaključenih selektorskih operacij - sekvenčnih strojev. Pri enostavnih sekvenčnih strojih lahko neposredno z uporabo transformacij definiranih v [1] in v 1. razdelku preoblikujemo graf sekvenčnega stroja v želeno obliko.

Izhajajmo iz vase zaključene selektorske operacije na sliki 2.1.



Slika 2.1: Izhodiščni graf za dekompozicijo

Dekompozicijo takšnega grafa lahko pričnemo, če imamo na voljo specifikacijo o operacijah, stanjih in pogojih, ki morajo biti izpolnjeni, da se operacije lahko izvedejo.

V splošnem lahko isti graf ob upoštevanju njegovega opisa (specifikacije) razgradimo na množico različnih načinov. Kako bo potekala razgradnja in kdaj bo postopek končan je odvisno od lastnosti operacij in zunanjih zahtev, ki smo jih postavili kot cilje snovanja.

Omejimo se na en sam primer dekompozicije sekvenčnega stroja, ki pa bo pretežno ilustriral karakteristike dekompozicije sekvenčnih strojev. Pri tem bomo uporabili poenostavljen zapis, saj bi bil formaliziran zapis preobsežen.

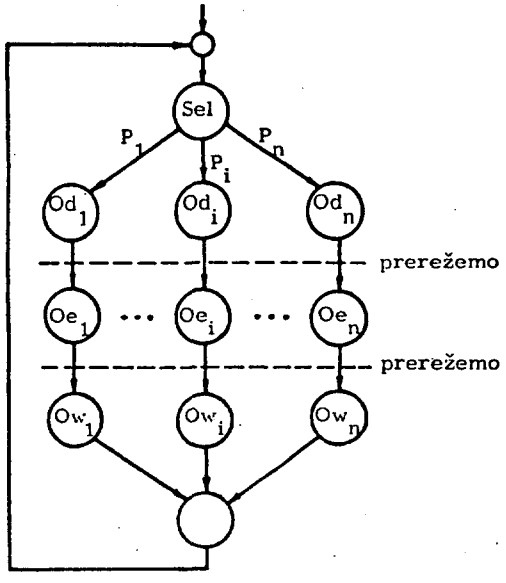
Izhajajmo iz naloge, da je potrebno izdelati procesor, ki izvaja instrukcije kompleksne računalniške arhitekture (CISC), katere opis obsega nekaj sto strani.
Že sam zapis selektorske operacije za tako instrukcijsko množico v smislu slike 2.1 bi bil toliko nepregleden, da iz njega ne bi kaj dosti razbrali. Zato uporabimo nekoli-

ko splošnejši pristop. Iz specifikacije naše instrukcijske množice ugotovimo, da je splošna zgradba instrukcije npr. takšna:

code src 1. rx src 2. rx ... src m.rx dst.wx (2.1)

V 2.1 je code ime operacije, iz katerega razberemo tudi tipe operandov, src j . rx so določila operandov ali kar operandi in dst . wx določilo destinacijskega operanda. Naša specifikacija dovoljuje, da lahko oblika 2.1 degenerira v code, v code z izvornimi operandi brez destinacijskega operanda ali code s samo destinacijskim operandom. Pri izvajanju vsake instrukcije se postavi še niz izjav, ki v splošnem lahko vplivajo na izbiro naslednje instrukcije za izvajanje. Značilna lastnost instrukcijske množice naj bo, da instrukcije i+1 ni možno pričeti dekodirati, dokler ni v celoti dekodirana instrukcija i, ali dokler ni v celoti dekodirana instrukcija i in izvedena zahtevana operacija, ki postavi niz izjav te instrukcije. To lastnost lahko ugotovimo iz imena operacije v instrukciji.

Ta opis nam zaenkrat zadošča, da lahko pričnemo z dekompozicijo. Izhajajmo iz slike 2.1 in predpostavimo, da imajo vse instrukcije splošno zgradbo (2.1). Tedaj lahko obdelavo poljubne instrukcije ponazorimo s tremi sekvenčno povezanimi operacijami in sicer z operacijo dekodiranja instrukcije Od_i, operacijo izvajanja Oe_i in operacijo vpisa destinacijskega operanda Ow_i. Eksplicitna definicija operacij nas ne zanima in jo lahko za konkretno instrukcijsko množico izpeljemo s pomočjo [1] in [2] . Model iz katerega smo izhajali na sliki 2.1 lahko sedaj preoblikujemo tako kot podaja slika 2.2.



Slika 2.2: Začetni korak dekompozicije

Odstopanja od idealizirane zgradbe instrukcij (2.1) bomo upoštevali kasneje. Takšen pristop nam omogoča, da se v določenih fazah snovanja osredotočimo na karakteristične lastnosti instrukcijske množice in prilagodimo izgradnjo modela tistim lastnostim arhitekture, ki so na določenem nivoju snovanja dominantne. Na ta način imamo ves čas pred seboj bistvo problema in se ne izgubljamo v detaljih. Seveda pa mora končni model naše arhitekture verno posnemati vse njene lastnosti. Lahko se zgodi, da pri takšnem pristopu "vidimo" tudi lastnosti, ki jih "ni" oz. tiste ki niso dovolj karakteristične, da bi jih upoštevali že na tekočem nivoju snovanja, ampak šele kasneje. Skrajna možna posledica je lahko, da je rezultat snovanja nezadovoljiv in je potrebno postopek snovanja ponoviti.

Da minimiziramo subjektivno komponento pri ugotavljanju karakterističnih lastnosti lahko koristimo razmeroma obsežen matematični aparat za analizo sistemov.

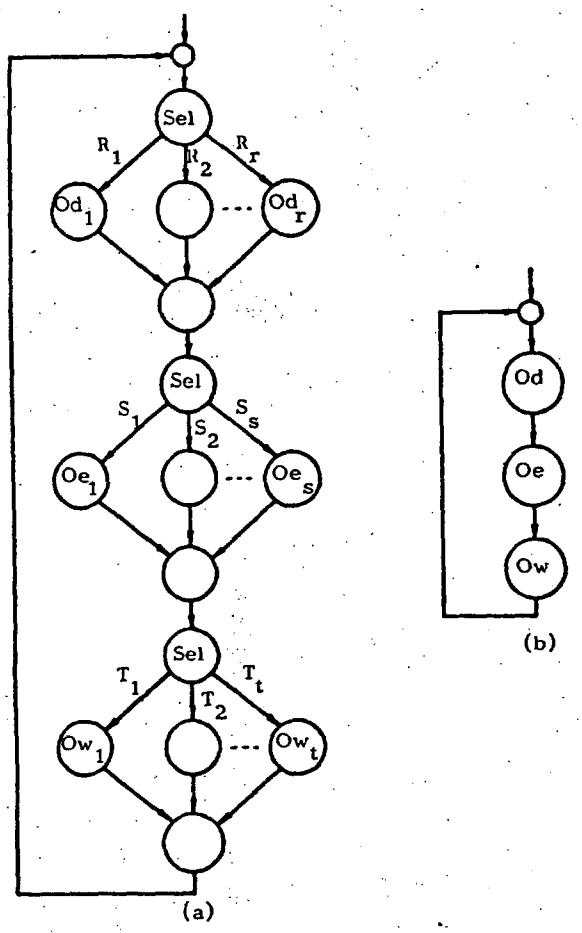
Graf s slike 2.2 lahko ponazorimo kot tri sekvenčno povezane vase zaključene selektorske operacije. Upoštevamo še, da lahko instrukcije, ki imajo enako operacijo Od_i, "enako" operacijo Oe_j (odvisno od organizacije operacijske enote) in enako operacijo Ow_l nadomestimo z eno samo operacijo, če ustrezno prilagodimo izjave $P_1,\ldots,P_n$ v vseh treh novih selektorskih operacijah. Tako dobimo krmilni model na sliki 2.3.

Na sliki 2.3 (b) je podan zgoščen graf za sliko 2.3 (a). 2.3 (b) nam bo omogočal dovolj zgoščen zapis, zato nadaljujemo snovanje s tako oblikovanim grafom. Poiskusimo sedaj združiti operaciji Od in Ow v eno samo sestavljeno operacijo. V ta namen definirajmo izjavo W-past, ki trdi, da je rezultat operacije Oe_j, j = 1,2,..., s določen. Ko postane izjava W-past pravilna se lahko sproži izvajanje operacije Ow.

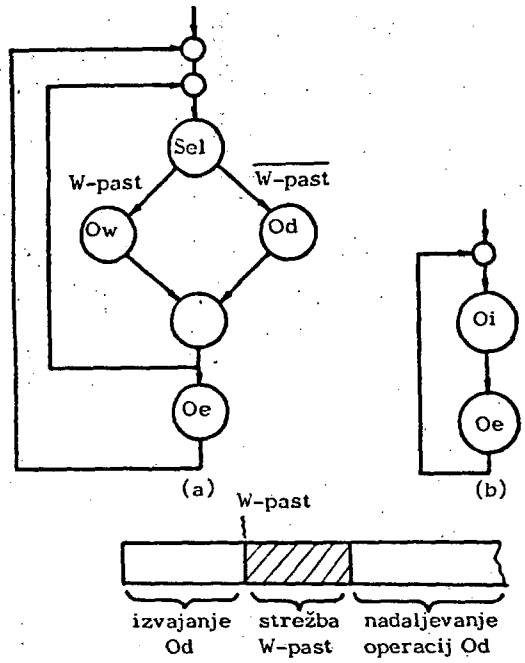
Graf s slike 2.3 (b) preide tako v obliko na sliki 2.4.

S tem smo dobili dve sekvenčno povezani sestavljeni operaciji Oi in Oe. Pri tem je Oi že v svoji zanki, kar pomeni, da jo bomo realizirali kot sekvenčni stroj. Tudi operacijo Oe bomo realizirali kot sekvenčni stroj, saj so operacije, definirane s kompleksno instrukcijsko množico toliko kompleksne, da zahtevajo pri logični realizaciji dodatno interpretacijo z enostavnejšimi operacijami.

Graf na sliki 2.4 (b) lahko preoblikujemo v paralelno operacijo, če dovolimo, da se ob izvajanju operacije Oe instrukcije i paralelno dekodira instrukcija i+1, torej izva-



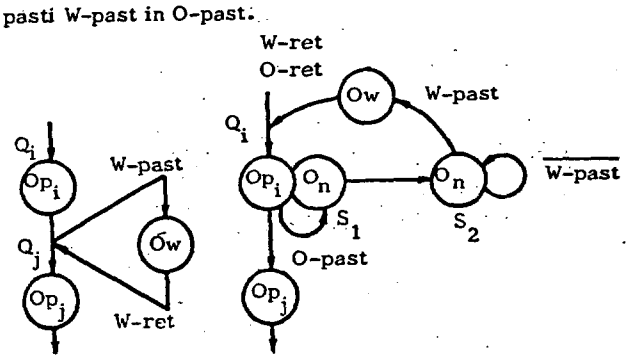
Slika 2.3: Ponazoritev s tremi sekvenčno povezanimi selektorskimi operacijami



Slika 2.4: Združitev operacij Ow in Od v sestavljeno operacijo Oi

janje operacije Oi. Takšna paralelna operacija je možna, če lahko po končanem dekodiranju instrukcije i pričnemo z dekodiranjem instrukcije i+1. Dekodiranje instrukcije i+1 se lahko izvede v celoti, če ni odvisnosti med izvornimi operandi instrukcije i+1 in destinacijskim operandom instrukcije i. Sicer je potrebno ob ugotovljeni odvisnosti med operandi, prekiniti izvajanje dekodiranja instrukcije i+1 ter počakati, da nastopi W-past, ustreči W-past, ter nadaljevati pri prekinjenem dekodiranju. Izjavo s katero ugotovimo odvisnost poimenujmo z O-past in jo definirajmo takole: O-past postane pravilna, če nastopi odvisnost med izvornimi operandi instrukcije i+1 in destinacijskim operandom instrukcije i, za katero se izvaja operacija Oe.

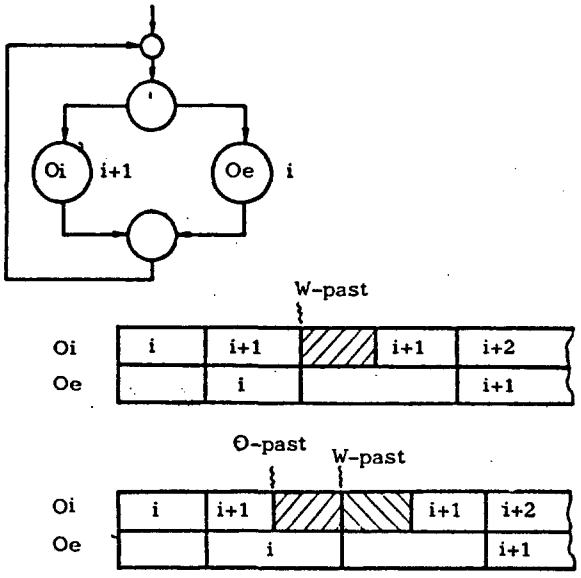
Na sliki 2.5 sta podana grafa, ki ponazarjata strežbo pasti W-past in O-past.



Sl. 2.5: Strežba W-pasti in O-pasti

Iz slike 2.5 razberemo, da se strežba W-pasti vrine med izvajanje operacij Op_i in Op_j (Od) in da se po končani strežbi izvaja operacija Op_j, ki je določena z izjavo Q_j, ki jo mora stroj pomniti, da lahko nadaljuje izvajanje. Pri strežbi O-pasti pa se operacija Op_i med izvajanjem spremeni v operacijo On, ki izvede "prazno" operacijo, nato nastopi čakanje na W-past in po končani strežbi W-pasti se ponovi izvajanje operacije Op_i (Od). Slika 2.6 podaja paralelno vase zaključeno operacijo Oi ' Oe. Uvedimo še termina za Oi in Oe za fizično izvedbo in sicer instrukcijski procesor za Oi ter operacijski procesor za Oe.

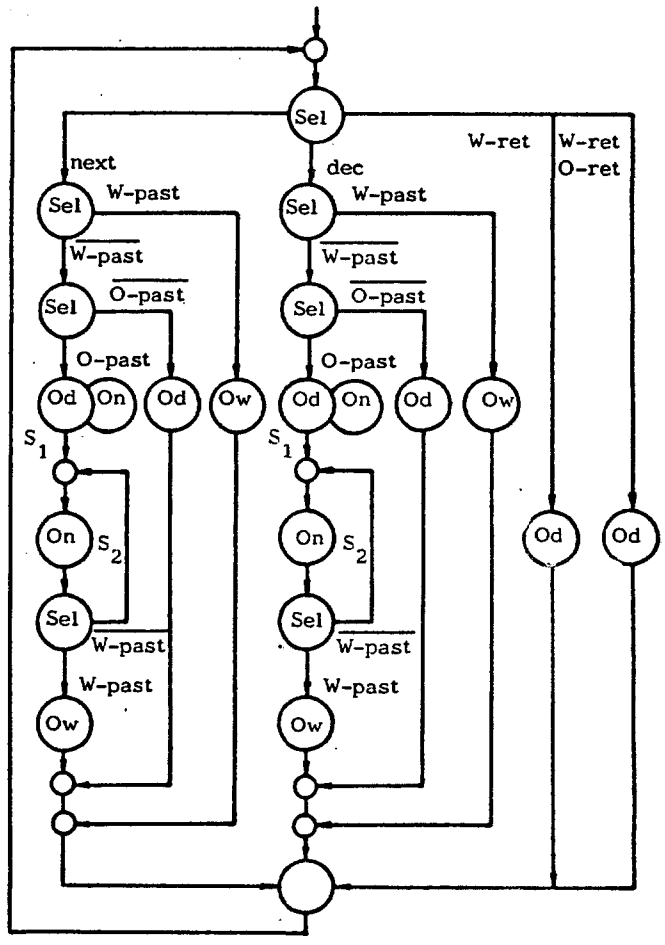
Sedaj imamo na voljo parametre za pričetek definicije krmilnih grafov za instrukcijski in operacijski procesor. Na sliki 2.7 je podan krmilni graf za instrukcijski procesor oz. za sestavljeno operacijo Oi s slike 2.6. Pri tem smo vpeljali še izjavi next in dec, ki omogočata - prva interpretacijo operacij dekodiranja in druga dekodiranje instrukcijskega niza. Nabor potrebnih krmilnih izjav s tem sicer še ni zaključen, vendar se bomo s takim krmilnim modelom zadovoljili in ga bomo prevedli na izhodiščno organizacijsko shemo instrukcijskega procesorja. Zato najprej preoblikujemo graf s slike 2.7 v graf vase zaključene selektorske operacije, tako kot jo podaja slika 2.8. Za preoblikovanje smo uporabili postopke



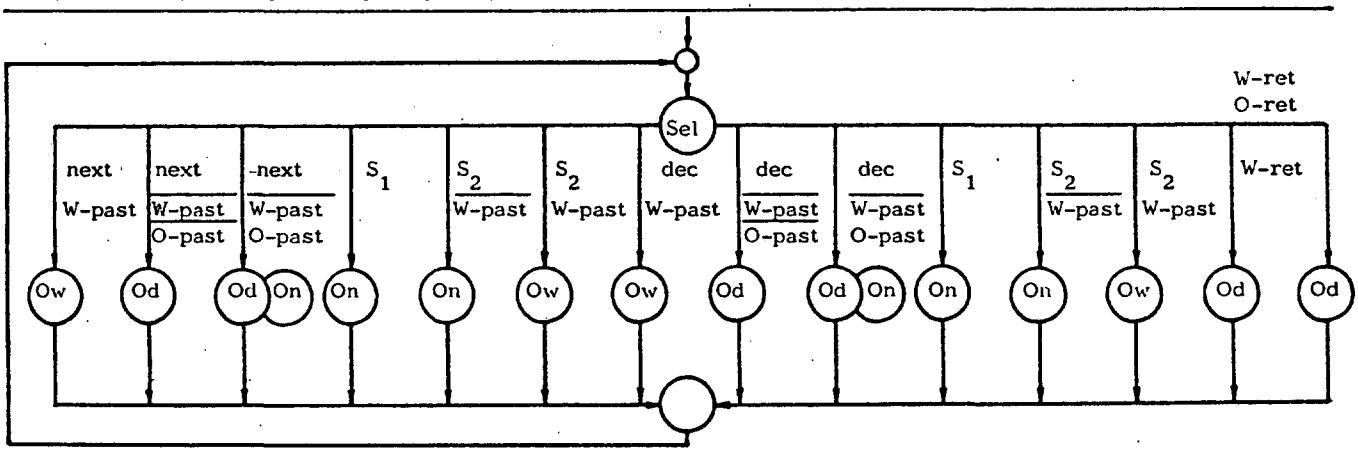
Sl. 2.6: Paralelni model izvajanja operacij

opisane v [1] in v 1. razdelku. V naslednjem koraku pa sliko 2.8 prevedemo v mikroprogramsko krmiljen model sekvenčnega stroja po sliki 2.9. Tako dobljeni model še ne izpolnjuje specifikacije naše kompleksne instrukcijske množice.

Zato specificiramo, da se instrukcije, katerih pričetek dekodiranja je odvisen od postavljenih izjav instrukcije v operacijskem procesorju, v celoti dekodirajo in izvedejo v instrukcijskem procesorju. Pri instrukcijah, ki nimajo destinacijskega operanda pa W-past sproži v in-

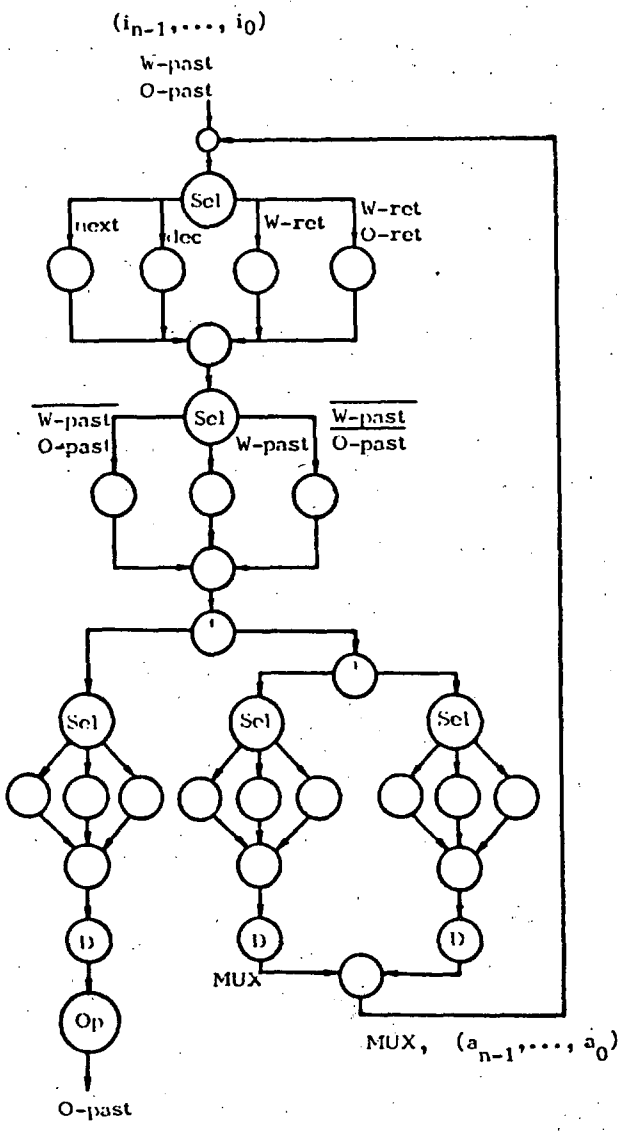


Slika 2.7: Začetni približek k realnemu krmilnemu modelu instrukcijskega procesorja

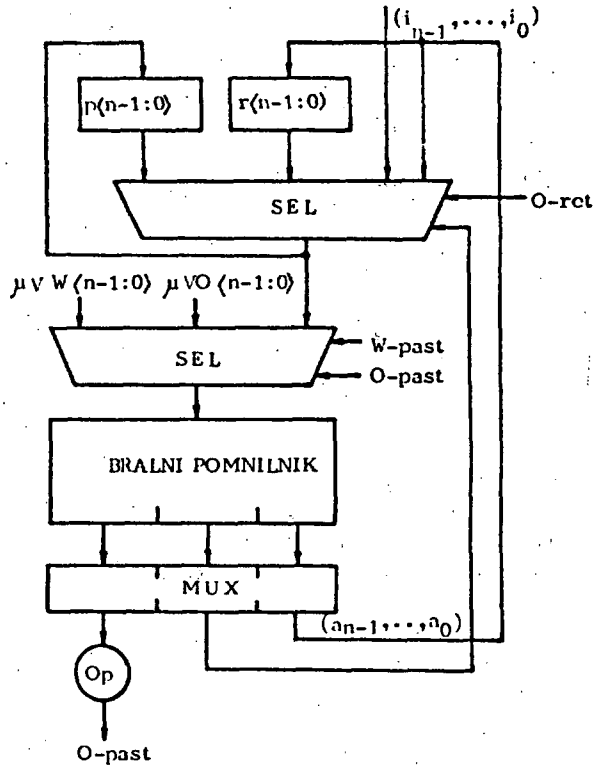


S_1 in S_2 lahko definiramo kot izjavi poljubno izbrani iz niza možnih konjunktivnih kombinacij izjav $a_{n-1}, \ldots, a_0$. next izjava določa niz $a_{n-1}, \ldots, a_0$ kot vir, ki določa naslednjo operacijo. Izjava dec določa niz $i_{n-1}, \ldots, i_0$ kot vir, ki določa operacije pri dekodiranju nove instrukcije. Izjava W-ret določa niz $r_{n-1}, \ldots, r_0$, kot vir, ki določa naslednjo operacijo ter W-ret ob aktivni izjavi O-ret niz $p_{n-1}, \ldots, p_o$ kot vir, ki določa novo operacijo. Izjava W-past izbira niz $\mu VW\langle n-1:0\rangle$ kot vir, ki določa naslenjo operacijo, ter izjava O-past niz $\mu V O\langle n-1:0\rangle$ kot vir, ki določa naslednjo operacijo. Definiramo še selektorsko operacijo MUX, ki specificira eno izmed izjav next, dec, W-ret.

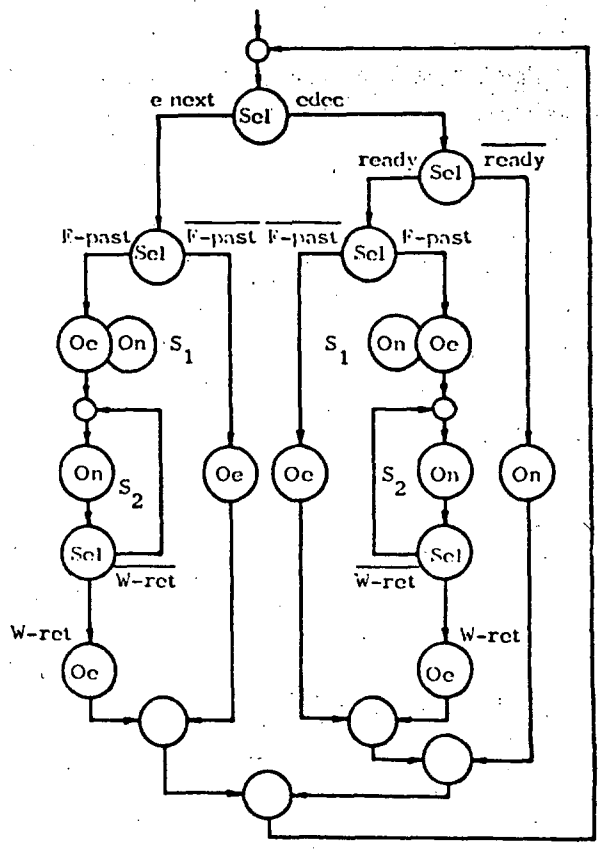
Slika 2.8: Vase zaključena selektorska operacija za 2.7



Slika 2.9: Mikroprogramsko krmiljen model instrukcijskega procesorja - začetni približek



Slika 2.10: Začetni približek k blokovni shemi instrukcijskega procesorja



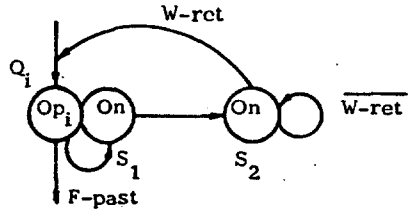
Slika 2.11: Začetni krmilni model operacijskega procesorja

strukcijskem procesorju izvajanje operacije On.

S tem zaključimo snovanje modela instrukcijskega procesorja; doslej dobljeno blokovno shemo podaja slika 2.10.

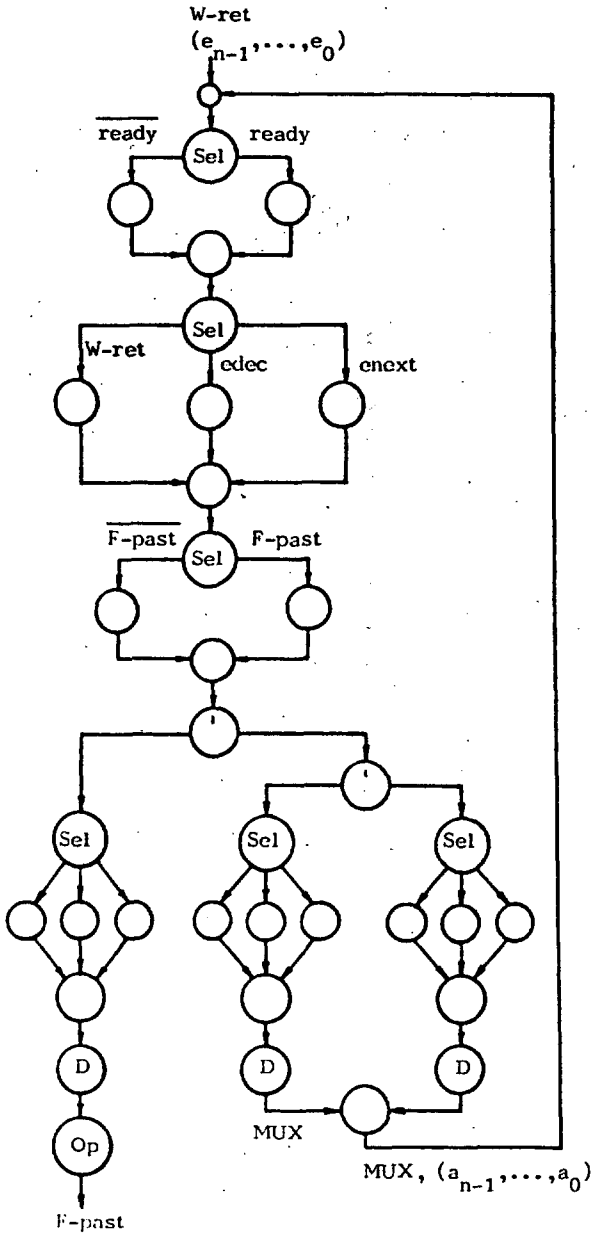
Podoben postopek opravimo pri snovanju začetnega modela operacijskega procesorja. Operacija Oe s slike 2.5 tako preide v sestavljeno operacijo, katere graf podaja slika 2.11. Pri tem je F-past izjava, ki pove, da se v operacijskem procesorju izvaja operacija, ki bo zahtevala strežbo W-pasti v instrukcijskem procesorju, hkrati pa v le-tem še teče strežba prejšnje W-pasti. Zato je potrebno počakati na zaključek strežbe W-pasti in nato ponoviti mikrooperacijo, ki je bila prekinjena ob strežbi W-pasti. Graf na sliki 2.12 podaja strežbo F-pasti. Izjava enext specificira izvajanje naslednje mikrooperacije, izjava edec pa pričetek izvajanja

novega mikroprograma, ki interpretira operacijo Oe_j, katero je določil instrukcijski procesor, če je tudi ready izjava, ki trdi da so vsi parametri za strežbo na voljo, pravilna.



Slika 2.12: Graf strežbe F-pasti

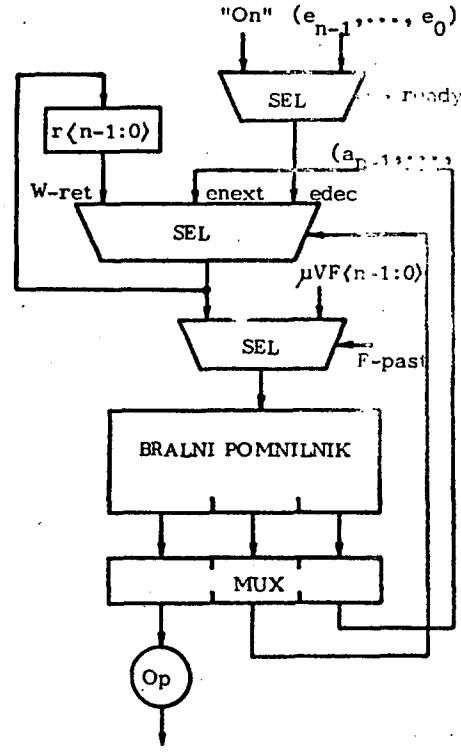
Graf s slike 2.11 po opisanih postopkih preoblikujemo v mikroprogramski model sekvenčnega stroja, katerega graf podaja slika 2.13. Začetno blokovno shemo operacijskega procesorja, dobljeno iz tega grafa, pa podaja slika 2.14.



Slika 2.13: Mikroprogramski model operacijskega procesorja - začetni približek

Doslej smo podali le pričetek dekompozicije sekvenčnega stroja (slika 2.1), s katerim smo pričeli razgradnjo. S smiselno uporabo doslej opisanih postopkov lahko krmilno strukturo obeh procesorjev razvijemo do potrebnih detaljev.

Čeprav smo se omejili na en sam primer dekompozicije, so v splošnem postopki razgradnje kompleksnih sekvenčnih strojev podobni doslej opisanim. Preostane nam še izgradnja modelov operacijskih enot za takšne stroje.

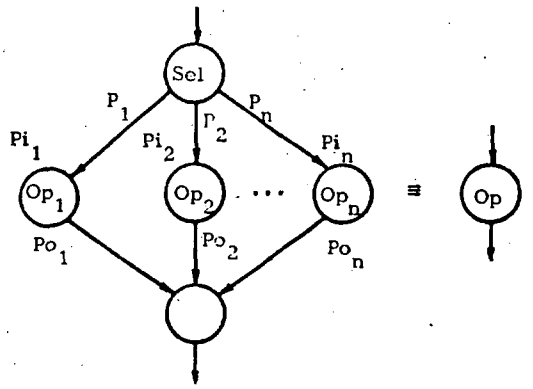


Slika 2.14: Blokovna shema operacijskega procesorja - začetni približek

3. MODELI OPERACIJSKIH ENOT

Predpostavimo, da smo za izbrano instrukcijsko množico razvili po postopkih [2] algoritme v simboličnem zapisu ali v poljubnem formalnem jeziku, ki opisujejo operacije določene z instrukcijsko množico na abstraktnem nivoju, primernem za logično realizacijo. Iz tako pripravljenih algoritmov lahko razberemo, katere operacije se izvajajo, katere komponente stanj moramo pomniti, katere komponente stanj dobimo iz instrukcijskega niza oz. zunanjega pomnilnika, katere izjave povzročijo vejitve itd. Skratka na voljo imamo vse podatke, da lahko v celoti realiziramo tako krmilno kot operacijsko strukturo sekvenčnega stroja oz. krmilne in operacijske strukture sekvenčnih strojev, odvisno od tega kako smo defi-

nirali organizacijo sistema. Načeloma lahko pričnemo z izgradnjo modela operacijske enote iz različnih izhodišč. V našem primeru pa definirajmo operacijsko enoto kot selektorsko operacijo, ki odvisno od pravilnosti izjav P_1, P_2,...,P_n izvede pripadajočo operacijo nad začetnim stanjem in producira končno stanje za to operacijo. Model operacijske enote lahko tedaj izgradimo iz selektorske operacije, katere graf podaja slika 3.1.
Izpolnjenost začetnih pogojev Pi_j, j = 1, 2, ..., n lahko ugotavljamo med izvajanjem operacij in sprožimo pasti, če ti pogoji niso izpolnjeni, ali pa smo ugotavljanje izpolnjenosti teh pogojev že vgradili v krmilne algoritme in jih ugotavljamo s pomočjo vejitev.

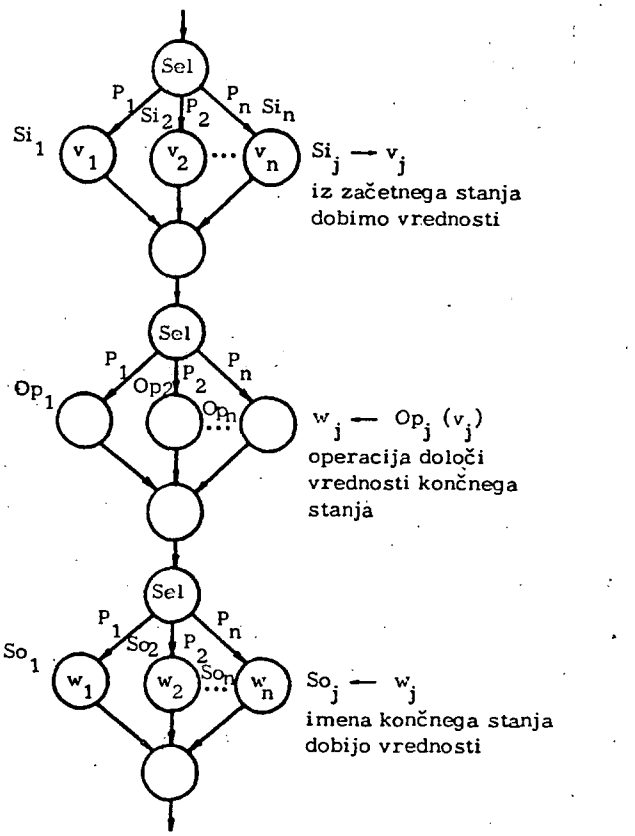


Slika 3.1: Začetni približek k modelu operacijske enote.

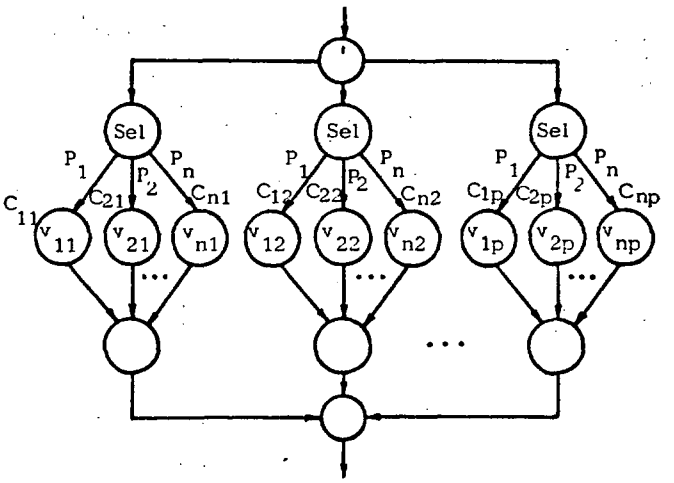
V praksi pogosto uporabljamo oba načina. V našem primeru bomo smatrali $Pi_j = 1$, j = 1, 2,..., n. V splošnem lahko rečemo, da glede na pravilno P_j, j = 1, 2,... ..., n najprej izberemo tej izjavi pripadajoče začetno stanje Si_j, nato aktiviramo izbrano operacijsko enoto, ki izvede operacijo Op_j, dobljene vrednosti pa priredimo imenom končnega stanja So_j. Graf s slike 3.1 preide tedaj v obliko na sliki 3.2.
V praksi so pogosto končna stanja nekaterih operacij ali njihove komponente hkrati tudi začetna stanja ali komponente operacij, ki se bodo šele izvršile. V ta namen predpostavimo, da imajo začetna stanja do p komponent in ustrezno preoblikujmo prvo selektorsko operacijo s slike 3.2 v paralelno selektorsko operacijo nad komponentami stanj po sliki 3.3.

Pri tem dovolimo, da so lahko komponente stanj prazne - $C_{jk} = 0$, $j \in \{1, 2,...,n\}$, $k \in \{1, 2,..., p\}$.

Za končna stanja So_j, j = 1, 2, ..., n predpostavimo, da imajo do r komponent. B_{jk}, $j \in \{1, 2, ..., n\}$, $k \in \{1, 2, ..., r\}$. Slika 3.4 podaja graf paralelne selektorske operacije, ki vrednosti komponent priredi imenom končnega stanja So_j. Tudi tukaj dovolimo, da se lahko izvedejo prazne prireditve $B_{jk} = 0$.

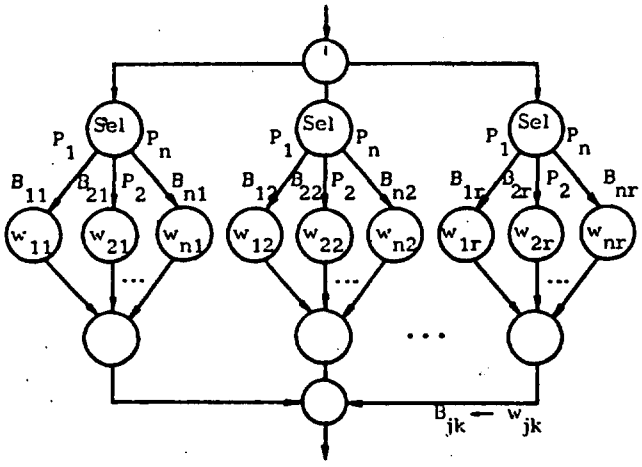


Slika 3.2: Model delovanja operacijske enote



Slika 3.3: Odjem vrednosti stanj Si_j po komponentah

Sedaj pa upoštevajmo, da so lahko končne komponente tudi začetne komponente novih operacij. V ta namen definirajmo tabelarično prireditev imen komponent C_{jk} in B_{jk} novim imenom C_1, C_2,..., C_m, tako da bo v tabeli vsaka komponenta, ki je hkrati začetna in končna imela eno samo ime. Nad preimenovanimi komponentami definirajmo novo paralelno selektorsko operacijo, pri kateri načeloma dovolimo, da lahko vsako vrednost komponente



Slika 3.4: Pripis vrednosti stanju So_j po komponentah stanja odjemamo ali priredimo (beremo oz. vpišemo).

Izjave R_i, i = 1, 2,..., p bodo določale odjem, izjave $\overline{R}_j$, j = 1, 2,..., r pa prireditev vrednosti. Izjave Q_{jk}, j = 1, 2,..., p oz r in k = 1, 2, ..., m bomo v konjunkciji z izjavami R_i oz. $\overline{R}_j$ uporabili za izbiro vozlišč katerim bomo odjemali vrednosti v_z, z = 1, 2,..., m oz. priredili vrednosti w_l, l = 1, 2,...,r. Za opis tako definirane sestavljene paralelne selektorske operacije uporabimo simbolični zapis.

$$
\begin{array}{lll}
\text{Sel:} & \text{Sel:} & \text{Sel:} \\
R_1 \text{ — sel:} & R_2 \text{ — sel:} & R_p \text{ — sel:} \\
\quad Q_{11} \text{ — } v_1 & \quad Q_{21} \text{ — } v_1 & \quad Q_{p1} \text{ — } v_1 \\
\quad Q_{12} \text{ — } v_2 & \quad Q_{22} \text{ — } v_2 & \quad Q_{p2} \text{ — } v_2 \\
\quad \vdots & \quad \vdots \quad \cdots & \quad \vdots \\
\quad Q_{1m} \text{ — } v_m & \quad Q_{2m} \text{ — } v_m & \quad Q_{pm} \text{ — } v_m
\end{array}
$$

$$
\begin{array}{ll}
\overline{R}_1 \text{ — sel:} & \overline{R}_r \text{ — sel:} \\
\quad Q_{11} \text{ — } (C_1) \text{ — } w_1 & \quad Q_{r1} \text{ — } (C_1) \text{ — } w_r \\
\quad Q_{12} \text{ — } (C_2) \text{ — } w_1 & \quad Q_{r2} \text{ — } (C_2) \text{ — } w_r \\
\quad \vdots \quad \cdots & \quad \vdots \\
\quad Q_{1m} \text{ — } (C_m) \text{ — } w_1 & \quad Q_{rm} \text{ — } (C_m) \text{ — } w_r
\end{array}
\qquad (3.1)
$$

Sestavljena paralelna selektorska operacija je definirana tako, da lahko "preberemo" poljubno permutacijo p vrednosti in "vpišemo" poljubno permutacijo do r vrednosti. Na ta način lahko dobimo vrednosti poljubnega začetnega stanja Si_j in priredimo vrednosti poljubnemu končnemu stanju So_j.

V praksi običajno ne potrebujemo tako splošne organizacije "pomnilnega prostora", zato lahko pri realizaciji konkretne naloge sestavljeno operacijo (3.1) primerno poenostavimo.

3.1. Model pomnilne celice

Izhajajmo iz selektorske operacije:

Sel (V):

$$
\begin{aligned}
V &\text{ — } P(Q, Q) \\
\overline{V} &\text{ — } P(Q, D),
\end{aligned}
\qquad (3.2)
$$

kjer je P relacija zamenjave vrednosti Q z vrednostjo Q ali vrednosti Q z vrednostjo D. (3.2) preoblikujemo v disjunktivno obliko:

$$V \wedge P(Q,Q) \vee \overline{V} \wedge P(Q, D) \qquad (3.3)$$

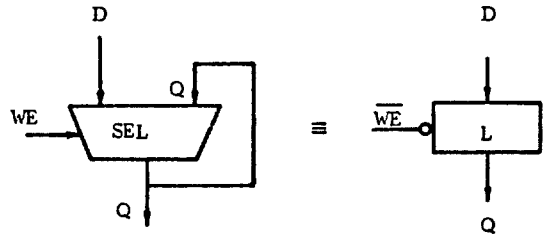
Modelirajmo V, Q, D z izjavami po [1] takole:
WE izjava, ki bo ekvivalentna izjavi V, $Q_{n-1},\ldots,Q_o$ niz izjav, s katerimi modeliramo Q, ter $D_{n-1},\ldots,D_o$ niz izjav, s katerimi modeliramo D. Relacijo zamenjave pa ponazorimo z logično ekvivalenco.

$$
\begin{aligned}
&WE \wedge (Q_{n-1} = Q_{n-1}) \vee \overline{WE} \wedge (Q_{n-1} = D_{n-1}) \\
&WE \wedge (Q_{n-2} = Q_{n-2}) \vee \overline{WE} \wedge (Q_{n-2} = D_{n-2}) \\
&\qquad\qquad \vdots \\
&WE \wedge (Q_o = Q_o) \quad \vee \overline{WE} \wedge (Q_o = D_o)
\end{aligned}
\qquad (3.4)
$$

Izraz (3.4) poenostavimo in dobimo:

$$
\begin{aligned}
Q_{n-1} &= Q_{n-1} \wedge WE \vee D_{n-1} \wedge \overline{WE} \\
Q_{n-2} &= Q_{n-2} \wedge WE \vee D_{n-2} \wedge \overline{WE} \\
&\vdots \\
Q_o &= Q_o \wedge WE \vee D_o \wedge \overline{WE}
\end{aligned}
\qquad (3.5)
$$

(3.5) imenujmo relacije pomnjenja,za pomnilno celico ki jo te relacije opisujejo pa uporabimo simbol na sliki 3.5.



Slika 3.5: Simbol pomnilne celice tipa zapah

3.2. Model pomnilnika s serijskim odjemom in serijsko prireditvijo

Da ne bi ponovno v celoti izvajali vseh relacij tako kot pri pomnilni celici, uporabimo nekoliko poenostavljen zapis. Z D označimo vrednost, ki je na vhodu pomnilnika, z D_n, $D_{n-1},\ldots, D_1$ označimo vrednosti v pomnilniku (pomnilnih celicah) z Q pa označimo vrednost na izhodu pomnilnika. Z A_i, i = 1, 2,..., n označimo poljubne

konjunktivne kombinacije niza izjav (a_{m-1}, a_{m-2},... ..., a_o), kjer velja $2^m \geq n$.

Z izjavo $\bar{R}$ pa bomo zahtevali prireditve vrednosti, z izjavo R pa odjem vrednosti.

Sedaj lahko definiramo selektorsko operacijo:

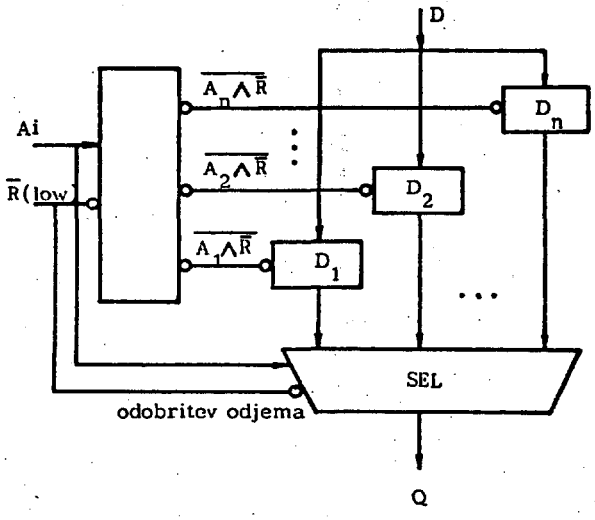
Sel:

A_1 — sel:
R — P (Q, D_1)
$\bar{R}$ — P (D_1, D)

A_2 — sel:
R — P (Q, D_2)
$\bar{R}$ — P (D_2, D) (3.6)
⋮

A_n — sel:
R — P (Q, D_n)
$\bar{R}$ — P (D_n, D)

Če upoštevamo model pomnilne celice in simbolični zapis (3.6), lahko kar narišemo eno izmed možnih blokovnih shem takega pomnilnika na sliki 3.6.



Slika 3.6: Blokovna shema serijsko organiziranega pomnilnika

3.3. Model paralelnega pomnilnika

Model serijskega pomnilnika posplošimo tako, da bo možno paralelno branje p vrednosti in paralelni vpis do r vrednosti ter s tem logično realizacijo sestavljene paralelne selektorske operacije (3.1). Predpostavimo $p \geq r$ in definirajmo ADR_{1i}, i = 1, 2,..., m,..., ADR_{ri},..., ADR_{pi}, ter so ADR_{1i},..., ADR_{pj}, i in j = 1, 2, ..., m, poljubne konjunkcije izjav a_{n-1},..., a_o ob pogoju da, če je i = j imamo opraviti z eno in isto konjunktivno kombinacijo izjav a_{n-1},..., a_o. D_r, D_{r-1},..., D_1 so vrednosti na vhodu paralelnega pomnilnika, Q_p, Q_{p-1},..., Q_1 so vrednosti na izhodu ter V_1, V_2,...,V_m vrednosti v pomnilnih celicah. R_1, R_2,..., R_p specificirajo branje iz paralelnega pomnilnika, izjave $\bar{R}_1$, $\bar{R}_2$,..., $\bar{R}_r$ pa vpis v paralelni pomnilnik.

Sestavljena paralelna operacija se tedaj glasi:

Sel:

R_1 — sel:
ADR_{11} — P(Q_1,V_1)
ADR_{12} — P(Q_1,V_2)
⋮
ADR_{1m} — P(Q_1,V_m)

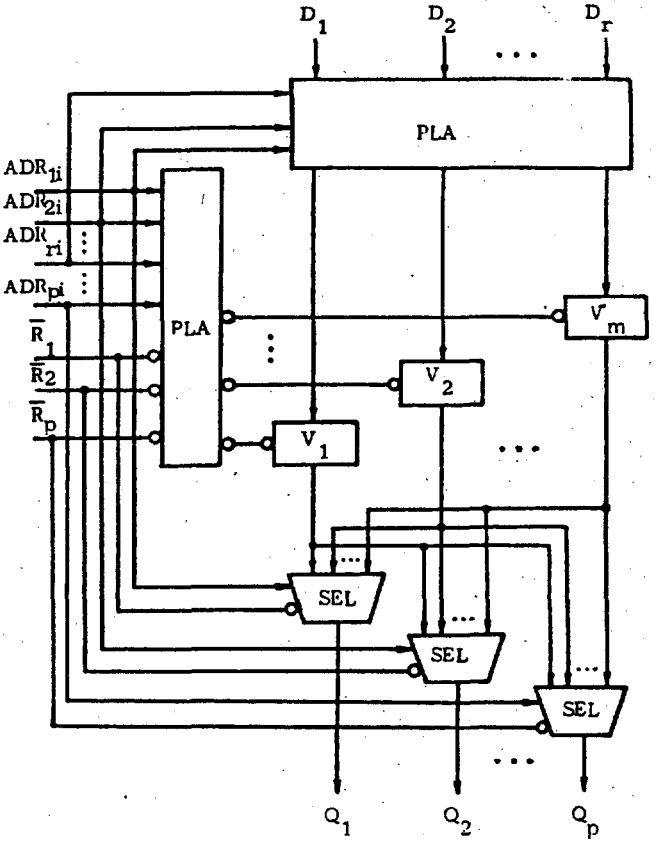
R_p — sel:
ADR_{p1} — P(Q_p,V_1)
ADR_{p2} — P(Q_p,V_2)
⋮
ADR_{pm} — P(Q_p, V_m)

$\bar{R}_1$ — sel:
ADR_{11} — P(V_1,D_1)
ADR_{12} — P(V_2,D_1)
⋮
ADR_{1m} — P(V_m,D_1)

$\bar{R}_r$ — sel:
ADR_{r1} — P(V_1,D_r)
ADR_{r2} — P(V_2,D_r)
⋮
ADR_{rm} — P(V_m,D_r)

Paralelna selektorska operacija, ki odloča o vpisu v pomnilne celice sicer ni enolična, vendar ni običajno, da bi istemu imenu komponente stanja skušali hkrati prirediti dve ali več vrednosti.

Eno izmed možnih blokovnih shem takega pomnilnika podaja slika 3.7.

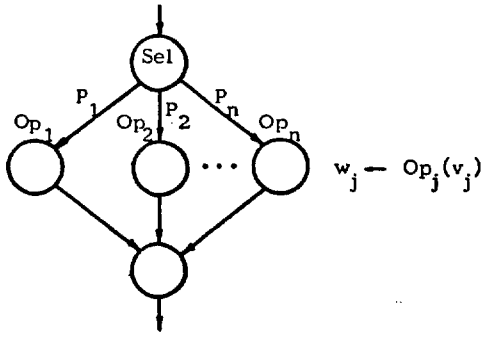


Slika 3.7: Blokovna shema paralelnega pomnilnika

Tak paralelni pomnilnik v praksi omogoča precej več kot realno potrebujemo. Zato ga lahko pri reševanju konkretne naloge primerno poenostavimo.

3.4. Modeli operatorjev

V tem razdelku bomo pod operatorjem razumeli predvsem enote, ki izvajajo operacije določene s selektorsko operacijo kot začetnim krmilnim modelom, ki ga lahko po potrebi razgradimo do primernega abstraktnega nivoja. V ta sklop sodi tudi selektorska operacija s slike 3.2, ki izvaja operacije $Op_j(v_j)$. Na sliki 3.8 je ponovno narisan graf selektorske operacije, iz katerega bomo razvili modele operatorjev.



Slika 3.8: Začetni približek k modelu operatorja

Selektorsko operacijo zapišimo v disjunktivni obliki:

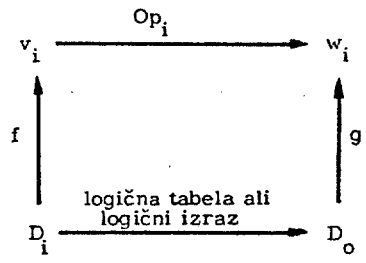
$$P_1 \wedge Op_1(v_1) \vee P_2 \wedge Op_2(v_2) \vee \ldots \vee P_n \wedge Op_n(v_n). \qquad (3.8)$$

$Op_j(v_j)$ pa pomeni vrednost w_j, ki jo dobimo kot rezultat operacije. (3.8) lahko tedaj zapišemo:

$$P_1 \wedge w_1 \vee P_2 \wedge w_2 \vee \ldots \vee P_n \wedge w_n \qquad (3.9)$$

Vendar (3.9) opisuje samo odjem vrednosti w_j iz izhoda operatorja, ne pa tudi izvajanje operacije. Zakaj sedaj nenadoma težave s selektorsko operacijo? Selektorska operacija, kot je definirana, je logični (krmilni) model kamor lahko zapišemo "karkoli" . Če sledimo pot i skozi selektorsko operacijo, lahko rečemo da, če je pravilna P_i potem se izvede operacija Op_i nad vrednostmi stanja Si_i in kot rezultat določi vrednosti končnega stanja So_i. Ostale poti v selektorski operaciji lahko v tem trenutku ignoriramo. Pri fizični izvedbi modela selektorske operacije pa je tako, da moramo definirati podatkovne poti, definirati logično izvedbo operatorjev, poskrbeti za krmiljenje celotnega operatorja in na koncu odvzeti pravilen rezultat.

V ta namen najprej definirajmo realizacijo operacije Op_i, i = 1, 2, ..., n, z operatorjem OP_i. V splošnem lahko operatorje realiziramo na dva načina; s pomočjo logičnih tabel ali s pomočjo logičnih izrazov. Slika 3.9



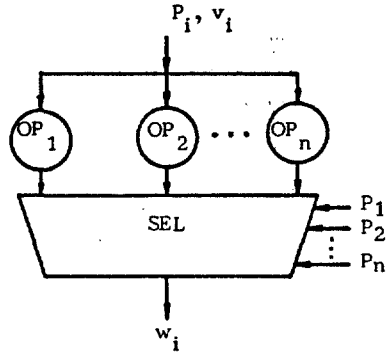
Slika 3.9: Definicija logičnega modela operatorja

podaja koncept po katerem operacijo Op_i prevedemo v njen logični model.

f in g sta surjektivni preslikavi, f^{-1} in g^{-1} pa sta v splošnem relaciji. D_i in D_o so nizi pravilnostnih vrednosti izjav, s katerimi smo modelirali vrednosti v_i in w_i glede na [1].

V splošnem lahko tedaj tudi operatorje ponazarjamo s selektorsko operacijo, saj lahko poljubno logično tabelo prevedemo v disjunktivno obliko prav tako pa tudi poljuben logični izraz.

Sedaj pa ponovno izhajajmo iz (3.8). Tedaj lahko glede na zgornja izvajanja narišemo naslednjo blokovno shemo operatorja za sliko 3.8 na sliki 3.10.



Slika 3.10: Začetni model operatorja

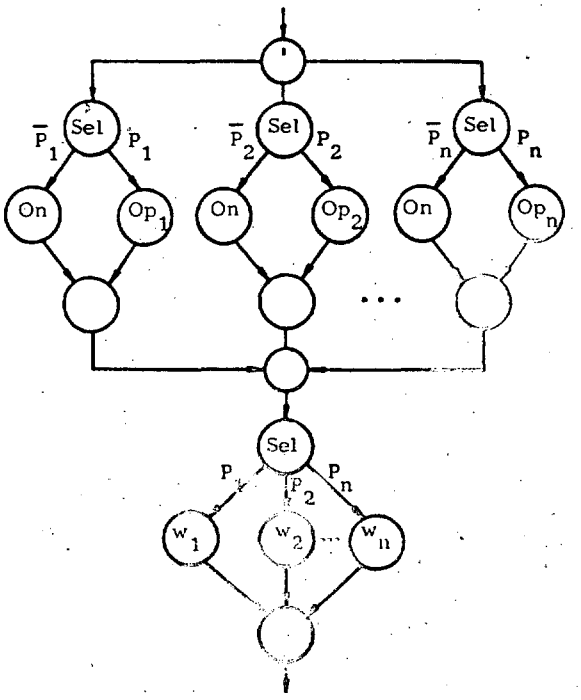
Kjer so $Op_1, Op_2, \ldots, Op_n$ v splošnem zopet selektorske operacije, ki jih lahko definiramo takole:

Sel:

$$\left.\begin{array}{l} P_i \wedge v_i^1 \rightarrow w_i^1 \\ P_i \wedge v_i^2 \rightarrow w_i^2 \\ \vdots \\ P_i \wedge v_i^x \rightarrow w_i^x \end{array}\right\} Op_i \qquad (3.10)$$

Z v_i^j smo označili različne vrednosti začetnih stanj definiranih z začetnim pogojem Pi_i (Si_i) in z w_i^j vrednosti končnih stanj So_i. V splošnem desna stran (3.10) ni enolična.

Sedaj pa definirajmo še nekatere enakovredne blokovne sheme operatorjev. Izhajajmo iz grafa na sliki 3.11.



Slika 3.11: Preoblikovana selektorska operacija

Ugotovimo lahko, da graf s slike 3.11 ustreza grafu s slike 3.8. Velja namreč:

$$OV\ldots VOVP_i \wedge Op_i(v_i) VOV\ldots VO \qquad (3.11)$$

in

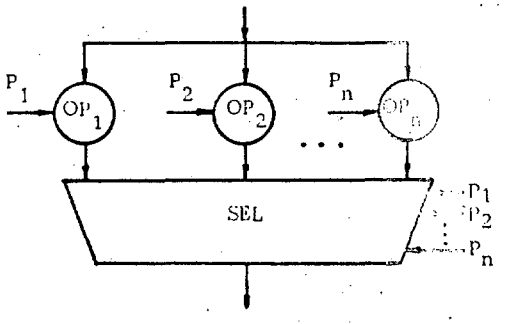
$$\begin{aligned}
&\overline{P}_1 \wedge O_n(v_i) \vee P_1 \wedge Op_1(v_1)\\
&\vdots\\
&\overline{P}_i \wedge O_n(v_i) \vee P_i \wedge Op_i(v_i) \qquad (3.12)\\
&\vdots\\
&\overline{P}_n \wedge O_n(v_n) \vee P_n \wedge Op_n(v_n)
\end{aligned}$$

Če je P_i pravilna tedaj zgornje relacije preidejo v obliko:

$$P_i \wedge Op_i(v_i)$$

kar je enako (3.11). Pri tem je oprecija O_n definirana tako, da velja $O_n(v_i) = 0$, $i = 1, 2, \ldots, n$.

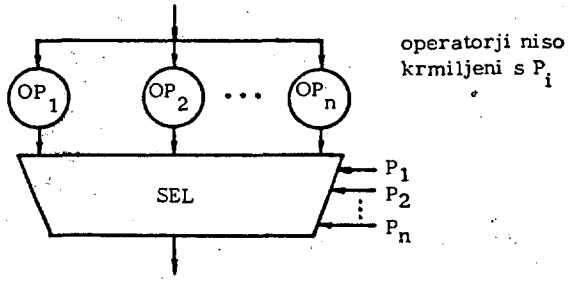
Blokovno shemo za ta primer podaja slika 3.12.



Slika 3.12: Enakovreden začetni model operatorja

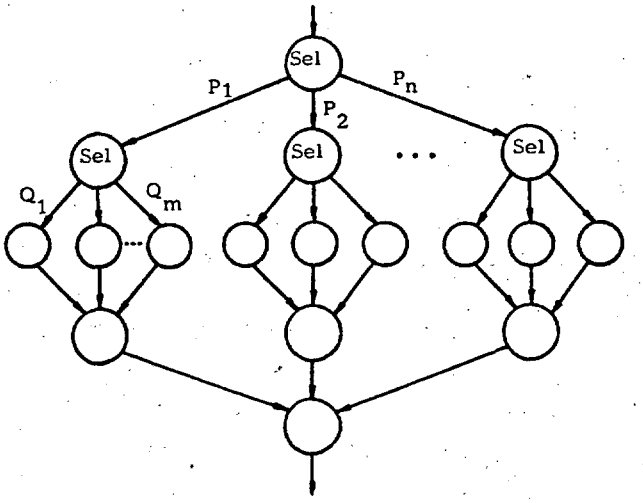
Iz razlogov, ki so snovalcem dobro znani lahko SEL operacijo s slike 3.12 pogosto nadomestimo z ALI operacijo ali s skupnim vodilom s tremi stanji.

V praksi se pogosto uporablja tudi poenostavljena blokovna shema operatorja s slike 3.12, ki jo podaja slika 3.13.



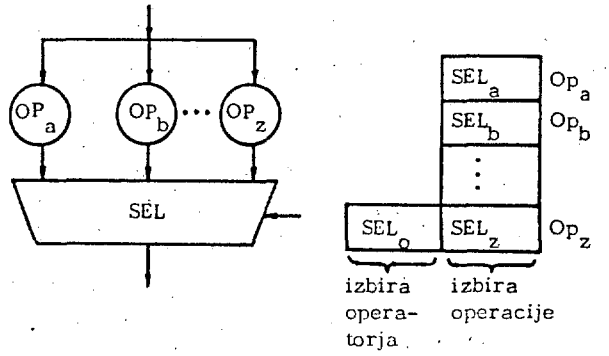
Slika 3.13: Poenostavljen model operatorja

Poimenujmo operator s slike 3.10 univerzalni operator. Sedaj pa izhajajmo iz krmilnega grafa na sliki 3.14.



Slika 3.14: Sestavljena selektorska operacija kot krmilni model

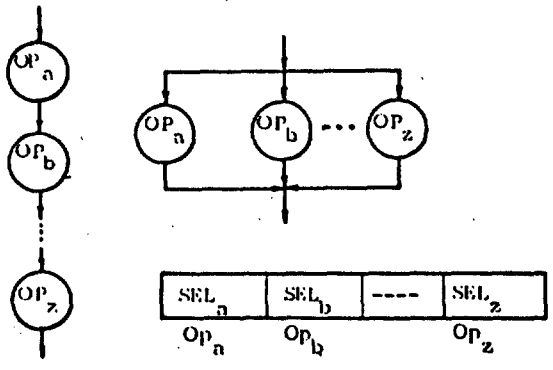
Vzemimo, da lahko notranje selektorske operacije realiziramo z operatorji OP_a, OP_b,..., OP_z. Tedaj lahko takoj narišemo blokovno shemo za celotno krmilno shemo s slike 3.14 na sliki 3.15.



Slika 3.15: Model operatorja in krmilna struktura za krmilni graf s slike 3.14

Na sliki 3.15 so SEL_i krmilni ukazi, do katerih pridemo s preoblikovanjem izjav $P_1,\ldots,P_n$, $Q_1,\ldots,Q_m,\ldots$ v nize ekvivalentnih izjav po postopku [1] . Pri snovanju mikroprogramiranih sistemov se je zanje udomačil izraz mikroukazi, za celotno polje pa izraz vertikalno funkcionalno polje.

V tem smislu sta na sliki 3.16 podani še blokovni shemi in struktura krmilnega polja za sekvenčno in paralelno vezane univerzalne operatorje.



Slika 3.16: Blokovni shemi in struktura krmilnega polja za sekvenčno in paralelno povezane operatorje

Snovanje operatorjev lahko tedaj pričnemo z izdelavo grafa, ki je v začetku selektorska operacija, ki jo po potrebi razgradimo v sestavljeno operacijo, sestavljeno iz selektorskih, sekvenčnih in paralelnih operacij. Tako dobljen graf pogosto imenujemo krmilni graf operatorja, saj določa precedenco operacij pri izvajanju. S pomočjo krmilnega grafa pa nato izdelamo blokovno shemo operatorja, ki izvede operacije določene z krmilnim grafom. Naslednji korak je logično snovanje, ki nas pripelje do logične sheme operatorja.

4. ZAKLJUČEK

Predlagani postopki snovanja in izgradnje logičnih modelov računalniških struktur so splošni in niso omejeni samo na snovanje mikroprogramiranih sistemov. Osnova postopkov so selektorska, sekvenčna in paralelna operacija, ki so lahko tudi vse zaključene. Z dodajanjem semantike lahko z njimi modeliramo najrazličnejše računalniške strukture na različnih abstraktnih nivojih.

Pokazali smo, kako lahko iste abstraktne strukture uporabljamo tako za snovanje krmilnih mehanizmov kot operatorjev, ki jih le-ti krmilijo. Izkazalo se je, da v splošnem ni mogoče postaviti ostre meje med krmilno in operacijsko strukturo izbrane računalniške strukture. Meja je s stališča zunanjega opazovalca (uporabnika, programerja itd.) določena z abstraktnim nivojem na katerem opazujemo obnašanje izbrane računalniške strukture. Od tod izhaja tudi programski model izbrane računalniške strukture za izbrani abstraktni nivo opazovanja. Po drugi strani pa lahko v splošnem isto računalniško strukturo opazujemo kot krmilno strukturo, operacijsko ali pomnilno strukturo. Ugotovili smo namreč, da v splošnem sestavljene strukture vsebujejo vse tri komponente. Od konteksta opazovanja je odvisno kako bomo "videli" takšno strukturo. Potrdimo to z zgledom: mikroprogramirana krmilna enota je brez dvoma krmilna struktura, saj določa operacije in njihovo zaporedje pri izvajanju. Hkrati pa je tudi pomnilna struktura, saj na adrese iz različnih virov odgovarja z različnimi podatki, ne glede na to kaj ti podatki predstavljajo. V njej lahko vidimo tudi operacijsko strukturo, posebej če jo opazujemo v kontekstu instrukcijski niz - mikroprogramirana krmilna enota - mikroinstrukcija - operacijska enota - izjave po končani operaciji, kjer je krmilna enota prva izmed dveh sekvenčno povezanih operatorjev (operacij).

Takšno menjavo kontekstov opazovanja pri snovanju pogosto uporabljamo, saj nam menjava konteksta prikaže problem v novi dimenziji in nas s tem navede na rešitve, do katerih bi lahko težko prišli z drugačnim kontekstom opazovanja.

5. LITERATURA

[1] M. Gerkeš: Logični modeli računalniških struktur, Informatica 3, str. 1 - 12, Ljubljana 1985.

[2] C. B. Jones: Software Design: A Rigorous Approach Prentice-Hall International, 1980.

[3] J. Virant: Preklopne funkcije, strukture in sistemi, Univerza Edvarda Kardelja v Ljubljani, Fakulteta za elektrotehniko, Ljubljana 1983.

[4] M. Gerkeš, M. Pernek, M. Paglavec: Aplikacija bipolarnega mikroprocesorja. Poročilo o delu za leto 1984, URP/RP: Računalniška oprema 03-2570, PORS 3, Visoka tehniška šola, Maribor, 1984.

Raziskavo je sofinancirala Raziskovalna skupnost Slovenije PoRS 03.