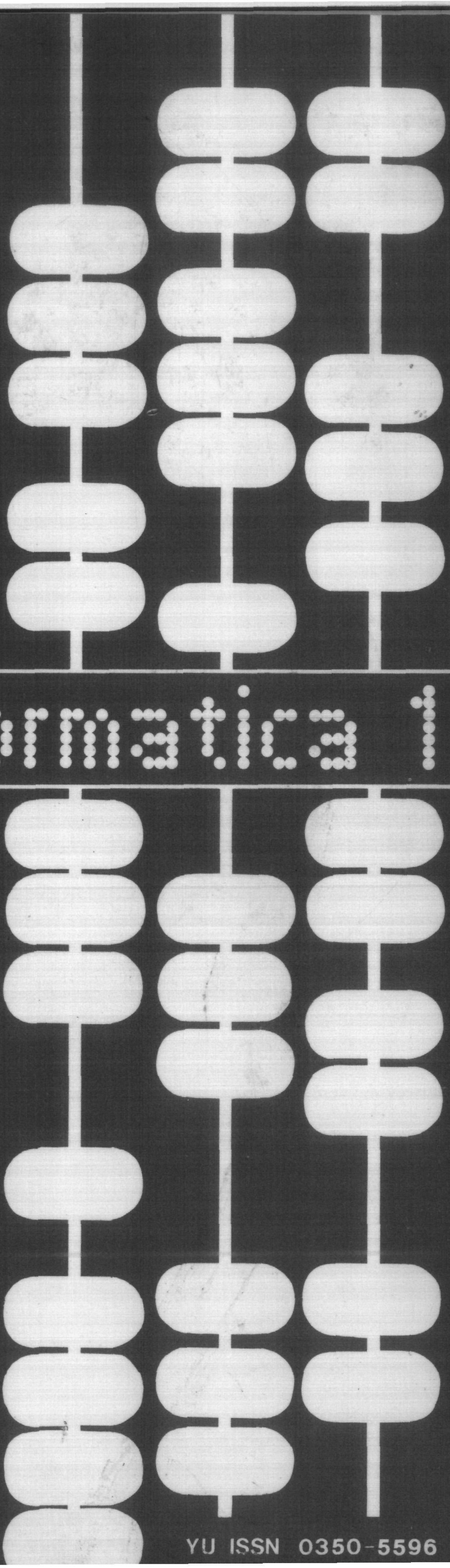


87 informatica 1

YU ISSN 0350-5596

informatica

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

Headquarters:

Informatica, Parmova 41, 61000 Ljubljana, Yugoslavia. Phone: 61 31 29 88. Telex: 31366 yu delta

Annual Subscription Rate: US$ 22 for companies, and US$ 10 for individuals

Printed by: Tiskarna Kresija, Ljubljana

JOURNAL OF COMPUTING AND INFORMATICS

YU ISSN 0350–5596

VOLUME 11, 1987 – No. 1

C O N T E N T S

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

Casopis izdaja Slovensko društvo Informatika, 61000 Ljubljana, Parmova 41, Jugoslavija

YU ISSN 0350–5596

LETNIK 11, 1987 – Št. 1

Uredništvo in uprava:

Informatika, Parmova 41, 61000 Ljubljana, telefon (061) 312 988; teleks 31366 YU Delta.

Letna naročnina za delovne organizacije znaša 5900 din, za zasebne naročnike 1590 din, za študente 490 din; posamezna številka 2000 din.

Številka žiro računa: 50101-678-51841

Pri financiranju časopisa sodeluje Raziskovalna skupnost Slovenije
Na podlagi mnenja Republiškega komiteja za informiranje št. 23-85, z dne 29. 1. 1986, je časopis oproščen temeljnega davka od prometa proizvodov

Tisk: Tiskarna Kresija, Ljubljana

V S E B I N A

UDK 681.3(05)(497.1):519.7

Anton P. Železnikar

Informatica praznuje desetletnico svojega izhajanja. Namesto statističnih, izdajateljskih in vsebinskih podatkov jo ob njenem prazniku sprejemam z dobrodošlico o njeni prihodnosti. Vivat Informatica!

Umetna inteligenca se je v zadnjem desetletju razvila v razgibano in širno znanstveno, raziskovalno, tehnološko in komercialno dejavnost. V njej so se začela smiselno povezovati idejna in konstrukcijska razmišljanja, katerih dotok je prihajal iz čelnih področij človekovega uma in raziskovanja, kot so filozofija, psihologija, lingvistika, nevrofiziologija, informatika pa tudi matematika in tehniške znanosti. To medpodročno oplajanje umetne inteligence pa je rodilo tudi povratni vpliv na njena prvotna izhodišča, saj so se nekatera vprašanja postavljala stvarno in predmetno in ne zgolj filozofsko. Danes je mogoče brez posebnih pomislekov govoriti o vzpodbudnem vplivu umetne inteligence na filozofijo, psihologijo, lingvistiko in informatiko. Metodologija umetne inteligence pa si utira pot tudi v oblikovanje strojev.

Vzporedni procesorski sistemi so kot raziskovalno in razvojno področje dobili poseben poudarek: oblikovali naj bi podstat prihodnjih računalniških sistemov. Zamisel vzporednih sistemov temelji še na stari tehnološki miselnosti, ki prihaja iz šestdesetih let. Procesorji so v sistem povezani s komunikacijsko mrežo, pri čemer je vsako procesorsko vozlišče dovolj zmogljivo za opravljanje posebnih ali tudi splošnih računalniških operacij. Cilj tega razvoja je računalniški sistem za splošno uporabo. Danes se značilni vzporednoprocesorski sistemi uporabljajo za posebne namene.

Učinkovita in splošna uporaba vzporednih procesorskih sistemov zahteva nekaj, kar je celo za človekov zavestni um dokaj novo: vzporedno razstavljanje vhodnih problemov. Clovek govori svoj zunanji (naravni, formalni) jezik zaporedno, saj to ustreza fiziološkemu ustroju njegovih možganov, govorno sprejemnih in izraznih organov. Vzporedno razstavljanje pa bi zahtevala hkratno govorjenje več pomenov tako, da bi ti pomeni ustrezali vhodnemu zaporednemu ali zaporednovzporednemu pomenu. Vzporedno razstavljanje problema naj bi opravil stroj, vendar bi moral ta stroj človek še zgraditi. Problem vzporednega razstavljanja zaporednega ali zaporednovzporednega vhodnega problema do danes še ni rešen. Na vidiku so samo posebne rešitve za posebne, očitno paralelne probleme (npr. vzporedno izračunavanje matričnih členov, dohodkov uslužbencev).

To, kar je značilno za razvoj novih računalniških generacij, je preseganje t. i. von Neumannovega načela, s katerim se utemeljuje zaporedno delujoči stroj, na katerem se izvajajo zaporedno konstruirani programi. V živčnih sistemih deluje biološka substanca vzporedno in v tej substanci se izvajajo tudi vzporedni informacijski procesi. Vendar je potrebno tudi tukaj poudariti, da so določeni korteksni procesorji ne samo specializirani, temveč so tudi enkratni. Delfin lahko govori in posluša več (do tri) popolnoma ločenih pogovorov, ker razpolaga z ustreznim številom korteksnih območij ter z ustrezno prirejenimi (usmerjalnimi in frekvenčnokanalnimi) slušnimi in govornimi organi. Za določeno vzporednost je tudi v biološkem sistemu potrebna vzporedna substančna podstat, v njej pa je omogočena vzporedna procesnost.

Preseganje von Neumannovega načela zahteva tako aparaturno in programsko vzporednost. Japonski projekt pete računalniške generacije si je izbral za cilj navzgornji razvoj vzporednoprocesorskega sistema. Najprej poskuša zgraditi novo (vzporedno) aparaturno substanco, ki bo omogočala vzporedno procesiranje programov. Ta navzgornji razvoj pa mora predpostavljati tudi razvoj vzporednoprogramske metodologije, ki pa se posredno ali neposredno vključuje v okvir umetne inteligence. Tu velja poudariti, da je razvoj umetne inteligence značilno navzdolnji, saj izhaja iz zamisli visokostrukturiranih informacijskih struktur človekovega uma (sama inteligenca je višja informacijska funkcija korteksov) in se odtod spušča na posamezna uporabnostna področja. Razvoj umetne inteligence je tako navzdolnje-pragmatičen celo na tistih področjih, ki so do nedavna veljala kot formalno trdna (npr. mehanično prevajanje jezikov). Cilj enega in drugega razvoja je v srečavanju obeh, v stremljenju, da se nekoč zlijeta v celoto, da eden drugega dopolnita.

Področje načrtovanja in graditve novogeneracijskih pa tudi današnjih računalniških in informacijskih sistemov postaja tudi čedalje bolj filozofsko zahtevno, saj se uporaba strojev pa tudi njihova struktura približuje vprašanjem, ki segajo v samo bistvo filozofije, procesov v živem, povezave med biološkim in tehnološkim. Načrtovanje strojev in procesov tako že upošteva tudi meje biološko in tehnološko mogočega, ko izreka prepričanje, kaj danes ni mogoče.

Pred nami je novo razumevanje spoznavanja, ki bo bistveno tako za razvoj človekove misli kot za razvoj njegovih strojev; od obeh bo odvisno preživetje človeških populacij.

Slovstvo

S. Ohsuga: Artificial Intelligence as New Generation Computing Technology. New Generation Computing, 4 (1986) 223-224.

M. Nagao: Current Status and Future Trends in Machine Translation. Future Generation Computer Systems, 2 (1986) 77-82.

T. Winograd and F. Flores: Understanding Computers and Cognition: A New Foundation for Design. Ablex Publ Corp, Norwood, NJ (1986).

NA POTI K INFORMACIJI
ON THE WAY TO INFORMATION

INFORMATICA 1/87

UDK 681.3:003:519.76

Anton P. Železnikar
Iskra Delta, Ljubljana

Povzetek. Kaj je informacija? Kaj je jezik? Kako je mogoče prek poti do jezika priti na pot do informacije? Kaj je informatično mogoče? Kaj je informatizacija in kaj protiinformacija? Ta članek je "informacijski" esej o teh vprašanjih in okoli teh vprašanj; z njimi in okoli njih odpira vprašanja o smiselnosti in potrebnosti filozofije o informaciji, o informacijski biti. Kakšen bi lahko bil tehnološki informacijski stroj? Zakaj takšen stroj danes ni mogoč in pod kakšnimi pogoji bi se lahko pojavil? Filozofija informacije je nujna zaradi raziskave tehnoloških možnosti pa tudi kot razumevanje spoznavanja v bitju in okolju. Zakaj informacija informatizira, sporočilo le komunicira ali sporočuje in podatek le podaja? Tehnološka informacija še ne informatizira in v tem je njena nemoč, neinformatičnost, neinteligenčnost.

Abstract. What is information? What is language? How it is possible to come across the way to language on the way to information? What is informatively possible? What is the process of informing and what is counter-information? This article is an "information" essay dealing with these questions directly and indirectly. These questions are opening new questions about the meaning and needs of information philosophy, on Being of Information. What could the technological information machine be? Why is such a machine not possible today and under which circumstances could it come into existence? Philosophy of information is a necessity in researching the possibilities for technological machines and also in understanding cognition in a living being and its enviroment. Why information is informing, messages only communicating, and data only giving? Technological information is not informing yet and here lies its weakness, its non-informing, and its non-intelligence.

Unterwegs zur Information

Zusammenfassung. Was ist Information? Was ist Sprache? Wie gelangt man ueber den Weg zur Sprache auf den Weg zur Information? Was ist das informatisch Moegliche? Was ist Informatisierung und was ist Gegeninformation? Dieser Artikel ist ein "informatischer" Essay um und ueber diese Fragen; damit bleiben die Fragen der Bedeutung und Notwendigkeit der Philosophie der Information, Seins der Information weiterhin geoefnet. Was koennte eine technologische Informationsmaschine schon sein? Warum ist eine solche Maschine heute noch nicht moeglich und unter welchen Umstaenden koennte sie erreichbar werden? Die Pilosophie der Information ist eine Notwendigkeit bei der Untersuchung technologischer Moeglichkeiten und auch bei der Auffassung der Kognition im Seienden und seiner Umgebung. Warum sagt man: eine Information informatisiert, doch eine Nachricht kann nur kommunizieren und eine Angabe (ein Datum) nur eingeben? Eine technologische Information informatisiert noch nicht und hier zeigt sich ihre Schwaeche, ihre Nicht-Informatisierung, ihre Nicht-Intelligenz.

情報への道

ｱﾝﾄﾝ･P･ｼﾞｪﾚｰｽﾞﾆｶﾙ

（要約）。情報とはなにか。言語とはなにか。言語にいたる道を辿ることによって、情報への道を辿ることが如何に可能であるか。情報科学的に可能なものとはなにか。情報化とはなにか、反情報とはなにか。本稿はこれらの諸問題につい

ての、そしてまたこれらの諸問題の周辺を巡る「情報的」エッセイである。このエッセイで、これらの諸問題を通じて、そしてさらにその周辺を巡って、情報哲学および情報的実在の哲学の意義・必要性を論じたい。工学技術による情報機械とはいったいどういうものなのか。このような機械の製造は今日なぜ不可能であるか、どのような条件を満たせばその実現が可能になりうるか。技術が提供している可能性を探るために、また、実存および環境における認知の過程を理解するために、情報哲学は不可欠である。なぜ情報は情報化をもたらしうるのか。伝達はただ「伝えること」にすぎないのかそれとも伝達によって実際になにかが「伝わりうる」のか。工学技術による情報は情報化をいまだにもたらしえないということこそ、その無力、その真の意味での非情報性、その非知性の現れである。

0

... Znanstveno in tehnološko usmerjenemu bralcu se bo dozdevalo neverjetno, da je lahko filozofsko razmišljanje praktično pomembno pri njegovem delu. Filozofija je lahko zabavna raznovrstnost, vendar so teorije, bistvene za tehnološki razvoj, očitno tiste, ki predstavljajo trdno znanost in tehniko. Midva sva ugotovila prav nasprotno. Teorije o naravi biološkega bivanja, o jeziku in o naravi človekovega delovanja vplivajo v temeljih na to, kako gradimo in kako uporabljamo obliko. ...

(Terry Winograd in Fernando Flores: Razumevanje računalnikov in spoznavanja: novi temelji oblikovanja, xii) *

Kaj je informacija? Na to vprašanje je potreben izčrpnejši odgovor, ker je pojem "informacija" v pogovornem jeziku bistveno zožen. Kakšen je širši pojem? Latinski samostalnik informatio (informationis, f.) ima pomene, kot so predstava, pojem, razlaga, pojasnitev. Današnji pomen pojma informacija je lahko širši, ni le statična, marveč je predvsem dinamična lastnost predstavljanja biti, vsakršnje pojavnosti v živem in neživem svetu. Ta lastnost informacije je tedaj tudi informiranje. Latinski glagol informare ima pomene, kot so upodobiti, obraziti, izobraziti, učiti, predstaviti, predočiti, opisati, misliti si. Informiranje, ki je glagolnik iz informirati, ustrezno dopolnjuje, razširja pomen samostalnika informacija, ki je tako tudi upodabljanje, oblikovanje, učenje itd. Tako postane pomen informacije tudi informacijsko procesiranje, procesiranje informacije, nastajanje informacije, informatiziranje itd., kar je lahko primerno pri obravnavanju živih informacijskih sistemov, informacijske tehnologije in pri načrtovanju inteligentne informacije in inteligentnih strojev.

Oxfordski slovar nudi v kontekstu pomena informacije le tri uporabne besede: inform, information, informative, pri čemer je pomen besed inform in informative izveden le iz besede information, katere pomen je v angleščini značilno zožen tako, da je informacija pasivno informativna. V nadaljevanju bo postopno razvidno, kakšna je razširjena pomenskost pojma informa-

----
* Angleški naslov knjige je večpomenski. Možna pomena sta še tale: "Razumevanje računalnikov in spoznanje" in "Razumevajoči računalniki in spoznanje". Tu je večpomenskost bržkone namerna.

0

... Readers with a background in science and technology may find it implausible that philosophical considerations have practical relevance for their work. Philosophy may be an amusing diversion, but it seems that the theories relevant to technological development are those of the hard science and engineering. We have found quite the opposite. Theories about the nature of biological existence, about language, and about the nature of human action have a profound influence on the shape of what we build and how we use it. ...

(Terry Winograd and Fernando Flores: Understanding Computers and Cognition: A New Foundation for Design, xii)

What is information? This question needs an exhaustive answer, because, in colloquial language, the notion of "information" is essentially narrowed. What is the extended notion? The Latin noun informatio (informationis, f.) has various meanings such as: representation, notion, interpretation, explanation. The present meaning of the notion of information can be broadened; information is not only static, it is a dynamic property, which represents Being as various phenomena in a living and non-living world. This process of information is also informing. The Latin verb informare has the following meanings: to form, shape, instruct, teach, represent, show clearly, describe, reflect. Informing, which is the gerund of inform, adequately completes and broadens the meaning of the noun information, which is also forming, shaping, arising, instructing, etc. Thus, the meaning of information also becomes information processing, processing of information, the coming of information into existence, informing, etc. This notion of information can be useful in dealing with living information systems, information technology, and designing intelligent information and intelligent machines.

In the context of information, The Oxford Dictionary only offers three valid words: inform, information, and informative, where the meanings of the words inform and informative are deduced solely from the meaning of the word information. In English, this meaning of information is substantially narrowed in the sense that information is passively informative. In this article, it will become clear, how the meaning of information must be extended so that it is active. Although, in many languages incija v smislu njene aktivnosti. Čeprav v vrsti drugih jezikov obstajajo izvedenke pojma informacije, pa so v angleščini nadaljne izpeljanke nedopustne, čeprav bi bile za razširjeno obravnavo pomenskosti informacije koristne zlasti pri razvoju filozofije o informaciji, ki lahko postane izhodišče pri odločanju o tehnoloških možnostih prihodnjih informatičnih projektov. Možne izpeljane besede in njihovi posebni pomeni bi lahko bili v slovenščini:

informirati: informiranje, informizem, informiranost, informirajoč, informljiv, informljivost;
informacija: informacijski, informacizem, informacijskost, informacijalen, informacijalnost, informacibilen, informacibilnost;
informativen: informativizem, informativnost, informativičen, informativičnost, informativljiv, informativljivost;
informatika: informatičen, informatizem, informatičnost, informatikalen, informatikalnost, informatikljiv, informatikljivost;
informatizirati: informatiziranje, informatiziranizem, informatiziranost, informatiziralen, informatiziralnost, informatiziraljiv, informatiziraljivost;
informatizacija: informatizacijski, informatizacizem, informatizacijskost, informatizacibilen, informatizacibilnost.

V čem je smisel, da se pojmovanje informacije privede v širši pomenski kontekst? Informacija je kot oblika informacijskega procesa temelj, na katerem so možne t. i. predstavitve oziroma predstavljanje značilnih miselnih, tehnoloških, občevalnih, fizikalnih, bioloških, kemijskih in domala vsakršnjih pojavov. Informacija je lahko predstavitveni temelj razumevanja pojavnosti, torej podstat vsakršnje procesnosti vesoljnega sveta. Informacija, ki se pojmuje dovolj široko, je lahko univerzalni temelj razumevanja, spoznavanja, predstavljanja sveta v določeni biti in kar je še pomembnejše, tudi izven določene biti; to velja tudi za bit informacije!

Informacija ne pozna omejenosti, discipliniranosti, utesnjenosti, ki so značilne za filozofijo, literaturo, znanost, umetnost ali celo za sam zunanji, naravni jezik. Informacija nastaja in s tem presega vse prepovedi, tabuje, mite, idole, v sebi razpoznava njihova bistva; v tem je njena prednost, neodvisnost, vseobsežnost, nadrejenost, nadrazmernost glede na folozofske sisteme, literarne forme, znastvene teorije, umetniški artizem, jezikovne zakonitosti, kar vse so v bistvu kulturne, zunanje, ideologizirane oblike, ideološko in medijsko pogojene oblikovnosti. Informacija ima v sebi vselej svojo svobodno, neomejeno, nepredvidljivo komponento nastajanja, oživljajoče razvojnosti. Prav zaradi te vseobsežnosti, svobodnosti, razvojne potence informacije je potrebno razvijati mišljenje o njej, o njeni nastajalnosti, tj. filozofijo informacije. To raziskovanje informacije, nastajanje znanja o nastajajočem, o mogočem nastajajočega je potrebno tudi zaradi ocenjevanja možnosti in nemožnosti realizacije informacijskega v živem in tehnološkem. Veliki informacijski projekti začenjajo brez nujne filozofske predpriprave o informacijsko mogočem in vobče ne dosegajo obljub, pričakovanj in napovedi, ki jih na svojih začetkih izpovedujejo. Projekti umetne inteligence in novih računalniških sistemov so jasni dokazi filozofske nedorečenosti informacije in projektne neuspešnosti zaradi filozofske neraziskanosti informacije in njenih sistemov.

formation has a number of derivatives, in English these derivatives are not available. However, they could be useful for the extended treatise of the meaning of information and in the development of information philosophy. This enables a realm for deciding what the technological possibilities of future information projects are. Mechanically derived words and their particular meanings might be the following:

inform: informing, informism, informness, informingness, informity, informable;
information: informational, informationism, informationalism, informationess, informationality, informationable;
informative: informativism, informativness, informativity, informativable;
informatics: informatical, informatism, informaticness, informaticity, informaticable;
informatize: informatizing, informatizeism, informatizeness, informatizable;
informatization: informatizational, informatizationism, informatizationess, informatizationable.

What does the notion of information in a broader context of meaning enable? Information, as a form of an information process, is the foundation on which the so-called representations or characteristic representings are possible, e. g. as there exist mental, technological, communicational, physical, biological, chemical, and almost all kinds of phenomena. Information can be the representational foundation of phenomenal understanding, and by this, the substance of various processing in the entire world. Information, when understood in a broader sense, can be a universal foundation for understanding, cognition, representation of the world in a Being, and what is even more important, representation the world outside of a Being. This is also valid in the case of the Being of Information.

Unlike in information, phenomena of limiting appear characteristically, in the form of disciplines and obstacles, in philosophy, literature, science, art or even in an external, natural language itself. Information is coming into existence and so it surpasses all prohibitions, taboos, myths, and idols by recognizing them in itself. On this route lies its advantage, independence, all-embracing, and superiority. Also on this route lies its relativity in comparison to philosophical systems, literary forms, scientific theories, artistic styles, and linguistic legality, all of which are essentially cultural and external forms, which are being ideologically and medially conditioned. Information is always in the possession of its free, unlimited, non-foreseeable component of arising, which is coming into existence and developing life. Because of its all-embracing, freedom, and development power, information can be useful in developing the notion of information and information arising, i. e. the philosophy of information. This research of information, of the arising of knowledge of arising, and of the possibility of this arising is also necessary for the evaluations of the possibilities and impossibilities of information realization in the living and technological surroundings. Large information projects start without the necessary philosophical preparations concerning the information possibilities. In general, these projects are not reaching their promise, expectations, and forecastings which are given in their initial stages. Some artificial intelligence and new generation computer systems projects are becoming evident proof of the philosophical indistinctness of information and of the projects unsuccessfulness due to the lack of philosophical information and information systems investigations.

1

... Kako je mogoče domišljati bit drugače, kot to, kar se nikoli ne spremeni? Kako je mogoče domišljati čas drugače, kot minljivo, nenehno spremenljivo domeno bivanja?

(Joan Stambaugh v Martin Heideggrovem: O času in biti, ix)

Kaj je jezik? Jezik je posebna oblika informacije. Po Heideggru je jezik pot k jeziku. Jezik govori, prav za prav jezikuje sam s seboj. Z jezikom bo govorjeno o informaciji, o njeni pojavnosti, o njenem obstoju, nastajanju. Predvsem o nastajanju. Informacija je nastajajoč pojav, je na poti k informaciji. Jezik je visoka informacijska oblika in prek njega, njegovega govorjenja, jezikovanja jezika bo vodila pot k informaciji.

Formula, da je informacija razumljiva sama po sebi, je smiselna. Uporaba te formule nalaga, da se z informacijo oblikuje pot do informacije z govorjenjem jezika na poti do jezika. To je pot do jezika o informaciji. Ali sploh obstaja še katera od drugih, nejezikovnih poti na poti k informaciji, na poti k njenemu spoznavanju?

Ta uvod je potreben, je nujen, da se ostane na poti k informaciji. Bit informacije je rekurentna. Kaj je informacijska rekurentnost? Z matematičnim pojmom rekurzivnosti, definitorne, funkcijske rekurzivnosti ni mogoče razkriti poti do informacijske rekurentnosti. Če bi bilo to mogoče, bi matematična rekurzivnost že lahko zgradila pot do jezika. Jezikovna abstraktnost rekurzivnosti je formalen jezik, metajezik. Formalnost je že sama po sebi podjezikovna, metajezikovna, slovnična, bistveno pomensko okrnjena, poenostavljena, formalizirana jezikovnost, za pot k informaciji pa je potrebna vsa raznoličnost jezikovnosti, razvijajoča, nastajajoča jezikovnost, ki zmore graditi pot do informacije s potjo do informacijske rekurentnosti. Ta rekurentnost pa je razen tega še informacijsko generativna, sestavljajoča, novonastajajoča, je pa tudi paralelnostna, paralelnoinformacijska, procesirnoparalelna. Gradnja poti do informacijske rekurentnosti z jezikom je svojevrstno omejena z jezikovno neparalelnostjo, šibkoparalelnostjo. Jezik še ne jezikuje zadovoljivo več svojih poti hkrati, paralelno jezikovnost mora v sebi šele vsakokrat razvijati s posebno obliko zaporedne pomenskosti. Toda kako? Sicer bo sam govoreči, jezikovani jezik nezadosten za pot k informaciji, ki ji je paralelna procesnost temeljna, implicitna, informacijskopogojna. Jezik lahko informatičnost le modelira, s tem pa jo hkrati tudi oblikovno filtrira, jezikovnopovratno modulira, pomensko poenostavlja in s tem popačuje.

Bit informacije je paralelnoserijsko informacijskoprocesirno spletanje, prepletanje, nastajanje, izginjanje, nenehna spremenljivost, paralelno-serijska procesnost njene biti same. Informacija je spreminjanje njene trenutne oblikovnosti, definitornosti, paralelnosti, je naraščanje in izginjanje njene zapletenosti, prepletenosti in širjenje v novo oblikovnost vedno znova na nove načine. Informacija je proces, je dinamičnost. Statičnost informacije je podatkovanje, sporočevanje, torej posebna oblika statične informacijske dinamičnosti.

1

... How can Being be thought other than as that which never changes? How can time be thought other than as the perishable, constantly changing realm of existence? ...

(Joan Stambaugh in Martin Heidegger's: On Time and Being, ix)

What is language? Language is a particular form of information. According to Heidegger, language is on the way to language. Language is conversing, is actually speaking with language itself. Language has the capability to speak about information, information phenomenon, information existence, and how information is coming into being. The most important subject of this essay is the coming of information into existence. Information is a growing phenomenon and is on its way to (new) information. Language is a highly informative form and across it, across its conversing, its speaking of language itself, it will lead the way to information.

The formula that information can be understood by itself is meaningful. This formula requires that the way to information has to be realized by information's use of the conversing of language on the way to language. The question to be pursued is whether a non-linguistic way on the way to information, on the way to information cognition, can exist at all?

This introduction is necessary, actually indispensable for remaining on the way to information. Being of information is recurrent. What is information recurrency? By using a mathematical notion of recursivness, i. e. definitional or functional recursivness, it is not possible to discover the way to information recurrency. If this were possible, then mathematical recursivness would be able to build the way to language. The linguistic abstraction of recursivness is a formal language, a metalanguage. Formalism by itself is sublinguistic, metalinguistic, grammatical, and truncates essential meaning from a language. Formalism itself is a formalised language. However, on the way to information, all linguistic heterogeneity, language's ability of development and growth is needed, so that the way to information can be built up by the way of information recurrency. In addition, this recurrency is information generative, assembling, new-growing, and also parallel, i. e. information and processing parallel. Building the way to information recurrency by language is specifically limited by either language's non-parallelism or weak-parallelism. Language does not speak satisfactorily in more than one way simultaneously, so anytime it becomes necessary, it is forced to develop an equivalent form of sequential meaning. But in what manner? A self-speaking, conversing language would not be sufficient for the way to information, to which the parallel processing is fundamental, implicit, and information conditioned. Through language, a process of Informing can only be modeled, and therefore it is formally filtered, language reversely modulated, simplification of the meaning of Informing and consequently, distorted.

The Being of Information is parallel-serial information processing, which is twisting, interweaving, growing, vanishing, continuously varying, and parallel-serial processing of the information Being itself. Information is varying in its momentary pattern, definiteness, meaning, and parallelism. It is growing and

Informacija ima lastnost protiinformatičnosti. Tu se pojavlja informatična posebnost, ki je zaenkrat le poimenovana. Kaj je protiinformacija? Kakšna je njena pot, po kateri poti dosegljiva? Protiinformacija bo na poti do informacije, pa tudi do protiinformacije. Informacija je vselej tudi protiinformacijska, je njena lastnost nastajanja informacije v informaciji. Protiinformatičnost je lastnost živega in bržkone tudi neživega, vesoljnega; vendar tehnološke protiinformatičnosti na poti informacijskega nastajanja na vseh mogočih informacijskih ravneh, višjih in zlasti nižjih, še ni. Zaradi tega je razvoj filozofije informacije s filozofsko protiinformatičnostjo nujen kot razumevanje bitja in kot možnost bitjevskih informacijskih strojev; ta filozofski, jezikovni razvoj je bistven za razumevanje živih strojev, strojev za preživljanje in s tem tudi za spoznavanje možnosti razvoja tehnoloških informacijskih strojev.

Bit informacije je informacijska. Informacijska bit je ne samo spremenljiva, je nastajajoča; to ni bit vseh možnih informacij, je tudi bit nemožnih informacij, ki bodo v sami biti o biti nastale. Bit informacije je živa, nepredvidljivo nastajajoča bit, ki je filozofsko ni mogoče kar tako uokviriti.

Čas je pot, na kateri informacija informatizira, na kateri informatiziranje postaja protiinformatiziranje. Toda informatiziranje ni ničesar drugega kot pojav, v katerem se pojavlja informacija. Tako je čas samo drug način, s katerim se pove, da informacija nastaja; v tem smislu je pot k času zamenjana s potjo k informaciji in bit časa je del biti informacije. Tako je čas regularna informacija na področju informacije.

vanishing in its complexity and interweaving and propagating in a new pattern in a repeatedly new manner. Information is both, a process and information dynamics. Statics of information are data transfer and communication. These are particular phenomena of the static information dynamics.

Information has the nature of counter-information processing. Here information philosophy is faced with information peculiarity. What is counter-information? What is the way to counter-information and by which way is it attainable? Counter-information will be on the way to information, but also on the way to counter-information. Information is also always counter-informative; this is the way information comes into being within information. The process of counter-informing is a property of the living world and probably also of lifeless space, of cosmos. A technological process of counter-informing on the way to information, which can grow and appear at all possible information levels, e. g. higher and particularly lower ones, has never existed. However, the development of information philosophy which incorporates philosophical counter-informing is becoming inevitablely necessary for the understanding of living beings, and also for the possibility of understanding the being's information machines. New philosophical, linguistic development has to be elaborated for future understanding of living machines, of machines for survival and of cognition, which may lead to the possibilities for future development of technological information machines.

The Being of Information is information. The Information Being is not only variable, it is also coming into existence; it is not only the Being of all possible information, of all that exists, but also the Being of impossible information, which may come into existence in the Being and concerns the Being. The Being of information is alive and unforeseeable becoming Being, which cannot be philosophically framed at all.

Time is the way in which information informs, in which informing is becoming counter-informing. But informing is nothing else than the phenomenon in which information is coming into being. So, time is only another way of stating that information is coming into existence. In this respect, the way to time is replaced by the way to information and Being of Time is becoming a part of the Being of Information. Here, time is becoming a regular information in the information realm.

2

... Kajti mišljenje mora, da bi bilo resnično, misliti tudi proti samemu sebi...

(Marko Uršič: Enivetok, 87)

Kaj je informacijska oblika in kaj informacijski proces? Informacijska oblika je označitev za značilen informacijski proces. Informacijski procesi se pojavljajo v živi substanci, kot so žive molekule, celice, organizmi, živčni sistemi, možgani, bitja, množice bitij, žive populacije, bitjevska in populacijska okolja; v neživem svetu, v tehnoloških napravah so informacijski procesi prisotni v različnih statičnih in dinamičnih stanjih materije, energije, prostora in časa. Informacijski proces je tako lahko kateri koli informacijsko doumljeni proces, stanje, oblika univerzuma. Mišljeno preprosto, poenostavljeno: informacija v neživem svetu je kot materialni, energijski, prostorki,

2

... For remaining sincere, mind also has to think against itself...

(Marko Uršič: Enivetok, 87)

What is an information form and what is an information process? An information form is only a designation for a characteristic information process. Information processes are appearing in living substance such as molecules of life, cells, organisms, nerve systems, brains, beings, groups of beings, living populations, being and population environments. In the nonliving world, e. g. technological equipments, information processes are present in various static or dynamic states of matter, energy, space and time. Thus, an information process can be any information-comprehended process, i. e. any state or form of the universe. This can be thought of in a simple or simplified časovni proces; informacija v živem svetu je kot nadgradnja neživega, od žive molekule do bitja. Informacijska oblika oziroma informacijski proces obvladuje tako živo in neživo pojavnost.

way: information in a non-living world exists as matter, energy, space, and time processes; information in a living world is a construction above the non-living, from molecule to being. Therefore, an information/form or an information process can govern phenomena of life and non-life.

Katera koli procesnost neživega in živega sveta je lahko doumljena, percipirana, spoznana kot informatičnost, kot to, kar je informacijska pojavnost. Bitje doživlja svet, sebe in okolje kot svojo informacijo, s filozofskega vidika kot svojo metafiziko oziroma ontologijo; bitjevska informacija je namreč subjektivna, subjektivno filtrirana zunanja informacija, subjektivno poabstraktnjena, okoliško zgodovinsko, spominsko oblikovana, vendar v sebe, v bitjevsko individualnost in prav z njo projecirana in s tem le metafizično, ontološko v bitje prestavljena, prevedena. Ta metafizičnost, ontološkost je na informacijski ravni razvijajoči dejavnik, dejavna informatičnost, ki z nenehnim procesiranjem oblikuje novo informacijsko pojavnost bitja, bitje pa z njo vpliva na okolje z obnašanjem, povzročenim z njegovo informacijo. Ontološkost, ki je abstrahirana, pomensko zgoščena informatičnost na ravni bitja, je visokofunkcionalizirana informacija, imenovana protiinformacija. Ko se bitje svoje ontološkosti zaveda, se ji upira, vstaja proti njej z zavestjo protiinformatičnosti, z objektivizacijo informatične subjektivnosti in vstopa tako v novo ontološkost (metafizičnost).

Any processing of the non-living or living world can be understood, perceived, and recognized as information, i. e. the essence of an information phenomenon. A living being experiences the world, itself, and the environment as its information. From a philosophical point of view this experience is its metaphysics or ontology. That is. to say, a living being's information is subjective, is subjectively filtered external information, subjectively abstract, environmentally historical, and memory shaped. But in itself, in the individuality of a being and particularly through the projection of this individuality, information is metaphysically or ontologically displaced or translated. This ontology is a developing activity on the information level, an activity of informing, which is modeling new information phenomena of a being by continuous processing. Through its own ontology, a being is influencing its environment behaviourally, where behaviour is a consequence of information. Ontology is abstract, symbolically condensed information of a being; it is a highly functional information, called counter-information. As a living being becomes aware of its ontology, it is resisting, rising against this ontology by the consciousness of counter-information and by objecting to this information subjectiveness. Hence, it enters into a new ontology (metaphysics).

Ontologija bitja je med drugim bitjevsko razumevanje sveta, notranji bitjevski model sveta, ki vpliva na njegovo obnašanje v povezavi z medialno senzorsko informacijo, oblikovanjem čustev, odločitvenim mehanizmom volje in s povratno senzorsko informacijo obnašanja. Tu bitje spoznava svet, razumeva to spoznavanje, kopiči izkušnje o spoznavanju, o razumevanju, o razumevanju spoznavanja. Tu je vsaka subjektivna zavest značilno ontološka in vsako bitje si s tem pridobiva svojo ontologijo, ki se predstavlja s svojskim načinom mišljenja in obnašanja, ki je pogojena s svojskim informatiziranjem bitja. Subjektivnost bitja je zajeta v njegovi ontologiji, ki je zanj značilna, namerna, celostna bitjevska informacija.

A being's ontology is also a being's understanding of the world or more precisely a being's internal world model. This model influences a being's behaviour which is dependent on the medial sensory information, the forming of emotions, decision mechanism of the will, and feedback sensory information concerning the behaviour. At this point, a being is recognizing the world, understanding cognition and accumulating experience of cognition and general understanding. On this level, every subjective consciousness is characteristically ontological, so that every being is gaining its ontology. This ontology is represented by a characteristic way of thinking and behaving which is conditioned by a characteristic informing of a being. Subjectivity is scoped in a being's ontology and it represents a being's characteristic, intentional, and total information.

Vsaka ontologija, tako ali drugače subjektivizirana, je subjektivni pojav bitjevske slepote (Heidegger). Informacijska ontologija oblikuje osnovo za razumevanje spoznavanja, ki je tubit, bitje samo, bitjevska celostna informacija, ontologija bitja. Ontologija ni nič drugega, kot stopnja informacijske slepote, ko bitje vrženo v svet, živi in se obnaša s to slepoto, jo kot zanj bistveno informacijo spreminja, ko se prilagaja, medialno obnaša. Tu se pokaže, kako je refleksivno mišljenje nemogoče brez tiste oblike abstrakcije, ki je slepota ali slepilo samo.

Each ontology, which is subjective in one way or another, is a particular phenomenon of a being's blindness (Heidegger). An ontology of information can become a foundation for the understanding of cognition, which is the being-in-the-world, a being by itself, a being's total information, and a being's ontology. An ontology is nothing more than a degree of information blindness. When a being is thrown into the world, its ontology is developed by living and behaving with the blindness, altering this blindness as the being's essential information, and adapting and medially behaving. At this point it becomes clear, how reflexive thinking is not possible without a form of abstraction, which is the blindness or illusion itself.

Vsako živo bitje je tudi informacijsko zaslepljeno; zaslepljenost izvira iz načina nastajanja informacije v živem; to nastajanje je informacijsko-transformativno, strukturno-abstra-

Each living being is also fascinated or blinded informatively. This blindness is arising in the same way, in which living information is coming into existence. Because this arising is information transformative and structurally abstract, it is in this respect subjective and ktno in v vsem tem subjektivno, ontološko oziroma metafizično. Zaradi tega je bitje tako ali drugače vrženo v svet, v okolje, v svoje bivanje in tako soočeno z okoliškim pritiskom nastajanja bitjevske informacije.

3

... "Naprej" - v tisto naslednje, kar vseskozi prehitevamo, kar nas vedno znova vtujuje, ko ga zagledamo.

(Martin Heidegger: Iz pogovora o jeziku, UZS, 99)

Informacijski proces oziroma informacijska oblika ima lastnost posebne informatičnosti, imenovane protiinformatičnost. Informacija ima lastnost procesnosti, in sicer informacijske procesnosti, ki je tudi sama informacija. Procesnost informacije je informacijska. Informacija nastaja s svojo procesnostjo. Procesnost je informaciji imanentna, je v njej sami vsebovana kot dejavnost, pojavnost informacijske razvojnosti, spremenljivosti. Ta lastnost informacije je protiinformatičnost. Protiinformacija je z informacijsko procesnostjo iz informacije nastajajoča nova informacija in nova informacijska procesnost. To je moč, zmogljivost informacijske procesnosti v sami informaciji, kjerkoli se informacija pojavlja. Delovanje lastnosti protiinformatičnosti je mogoče opazovati v najenostvnejših in v najzapletenejših informacijskih procesih živega sveta, od živih molekul do najvišjih organizmov.

Molekule življenja, kot so DNA, RNA in beljakovine, so hkrati informacijski objekti, informacijski arhivi in informacijski stroji, ki iz arhivske informacije raznovrstno sintetizirajo nove molekule v odvisnosti od stanj njihovih okolij. Nove molekule imajo vobče tudi nove informacijske, reaktivne lastnosti. Molekularna protiinformatičnost ni samo genskomutacijska sintezna preslikava, je tudi okoliškoodvisna in v neznanem okolju nepredvidljiva, nenapovedljiva. Na celični ravni je informacijska diferenciranost še večja. Pojav novih virusov, bakterij, celic z njihovo novo informacijsko organizacijo je nepredvidljiv. Organizem odgovarja bolezenskim napadalcem s pripravo novih imunskih in imunskopomnilnih molekul, ki napadalce razpoznavajo in organizirajo njihovo nevtralizacijo. Ti molekularni procesi so povezani z branjem in razpoznavanjem molekularne informacije, z njenim prepisovanjem in preurejanjem, s prevajanjem te informacije v molekularno sintezo in prek nje z nastajanjem nove, v molekulah arhivirane informacije. Posledica tega informacijskega dinamizma je razvoj živega, evolucijskega in življenskega. Razvoj je esenca informatičnosti, protiinformatičnost pa bistvo informacijskega razvoja, razvoja informacije iz manjše v večjo zapletenost pa tudi obratno, v novo pomenskost, biološko učinkovalnost.

Miselnost bitja je kot korteksna procesnost razvijajoča od bitjevske zaploditve do njegove smrti. Informacija uravnava prek dednih zasnov biološko rast bitja, nastajanje bitjevske biološke substance, organov, bitja kot osebka, uravnava pa tudi vse procese v tej substanci, življenske, prav za prav informacijske procese.

metaphysical. Therefore, a being which is thrown into the world, into its environment, and into its existence, in one way or another, is faced with the environmental pressure on originating its information.

3

... "Vor" - in jenes Naechste, das wir staendig uebereilen, das uns jedesmal neu befremdet, wenn wir es erblicken. ...

(Martin Heidegger: Aus einem Gespraech von der Sprache, UZS, 99)

An information process or an information form has the property of counter-informing, which is a special form of informing. Information has the property of processing, namely information processing, which is, by itself, information. Processing of information is information. Information is coming into existence or into being by its own processing. Processing is immanent to information and it is a part of information itself; it is an information activity, phenomenon of information development and information variability. This processing of information is counter-informing. Counter-information comes into existence from information. As information informing or information processing acts on information, the consequence is new information and new processing. This is the power and ability of information processing within information itself, irrespective of where information is coming into existence. The functioning of the property of counter-informing can be observed in the simplest and most complex processes of the living world, from molecules of life to the highest organisms.

Molecules of life like DNA, RNA, and proteins are information objects, information archives, and information machines simultaneously. They diversely synthesize new molecules on the basis of stored information and molecular environmental states. In general, new molecules carry new information and new reacting properties. Molecular counter-informing is not only a synthesis mapping of a gene mutation, it is also environmentally dependent, not expected in an unknown environment, and not foreknown. On the cell level, information diversity is becoming even greater. The appearance of new viruses, bacteria, and cells having new organization of information is non-foreseeable. Organisms respond to pathological invadors by producing new immune and immunity storage molecules which are able to recognize invadors and to organize neutralization of them. These molecular processes are closely connected with reading, recognizing and transcribing molecular information, rearrangement of amino acids, or with the translation of this information by molecular synthesis. Through these processeś new molecules are coming into being which have new information being stored. A consequence of this information dynamics is the development of existing, of evolution, and of life. Development is the essence of informing and counter-informing is the nature of information development, of development from information having a lower complexity to information having a higher complexity and also vice versa. This is a development into a new meaning and a new biologic effectiveness.

The mind of a being is more precisely the development of the being's cortices, cortical and other information processing from the being's conception to its death. Here, information which uses hereditary schemes is not only controlling the arising and growth of biologic substance, organs and cortices of the being, V korteksu človeka doseže informatičnost svoj zavestni višek, notranji govor, ali natančneje zavestno notranje jezikovanje bitjevskega jezika, bitjevski protijezik, primerek protiinformatičnega abstrakta bitja, njegove biti.

4

... Ostajajoče v mišljenju je pot. In poti mišljenja vsebujejo v sebi skrivnostnopolno, da jih lahko hodimo naprej in nazaj, da nas celo šele pot nazaj vodi naprej.

(Martin Heidegger: Iz razgovora o jeziku. UZS, 99)

Kaj je bitjevski jezik? Informacija bitja je strukturirana oblikovno in pomensko, z bitjevskoznačilno, individualno lingvističnostjo. Nastajanje bitjevske informacijske strukturiranosti je bržkone nujno, pogojeno z imperativom informacijske nastajalnosti, razvojnosti. Informacijska jezikovnost bitja nastaja kot informacija, kot odgovarjanje bitjevske procesne informatičnosti na zunanjo, senzorsko informacijo in njenih posledic, razpoznanih skozi bitjevsko obnašanje v danem okolju. Čeprav okolje sproža nastajanje informatične strukturiranosti, ostaja ta bitjevsko značilna, značilnost bitja glede na njene poudarke, specifične oblikovnosti in pomenske svojskosti. V tem strukturnem kontekstu nastaja notranji jezik, bitjevsko miselno jezikovanje, bitjevska jezikovna razvojnost. To je le ena izmed mogočih predstav o bitjevski ojezikovljenosti informacije.

Bitjevski jezik je primarna informacijska pojavnost bitja, bistvo bitja, njegova sprejemljivost, izraževalnost oziroma obnaševaljnost v informacijskem kontekstu bitja in njegovega okolja. Bitjevski jezik je vobče večplasten, večglobinski, organiziran hierarhično, strukturnoznačilno. Do tu je to bitjevski notranji, miselni jezik. Od tu naprej se začenja bitjevski zunanji jezik, to je tisti jezik, s katerim bitje informatizira svoje okolje in s katerim sprejema svojo informatizacijo in s tem informatizacijo sebe iz okolja. Zunanji jezik bitja je glede na njegov notranji jezik drugoten, je le vmesna sporazumevalna stopnja, informacijski mehanizem med notranjim, v bitju vdomljenim jezikom in med zunanjo, bitju vtujeno pojavnostjo. Ojezikovljena informacijska vdomljenost in vtujenost, njuna razmernost je med drugim izvirnost bitjevske protiinformatičnosti. Vtujenost je imanentna zunanjemu jeziku, vdomljenost notranjemu, razlika med obema jezikoma je vselej bistveno informatizacijska.

Zunanji jezik bitja so njegovi čutni, naravni in seveda formalni jeziki okolja. To so jeziki, ki jih bitje uporablja pri svojem čutnem-jezikovnem informatiziranju. Vsi ti jeziki so drugotni glede na notranji jezik bitja in bitje jih uporablja v informacijskem kontekstu z drugimi bitji, s stroji in z okoljem.

but also all processes in this substance and processes of life, both of which are actually information processes. In the human cortex the process of Informing is reaching the highest point of consciousness, the internal speech or more precisely a conscious internal speaking of a being's language, and a being's counter-language, which is a case of being's counter-information abstraction, of being's Being.

4

... Das Bleibende im Denken ist der Weg. Und Denkwege bergen in sich das Geheimnisvolle, dass wir sie vorwaerts und rueckwaerts gehen koennen, dass sogar der Weg zurueck uns erst vorwaerts fuehrt.

(Martin Heidegger: Aus einem Gespraech von der Sprache, UZS, 99)

What is a being's language? Information of a being is formally and meaningly structured by the being's characteristic, individual linguistics. When coming into existence, a being's information structure is probably a necessity, which is conditioned by the imperative of the growth and development of information. A being's information linguistics is coming into existence as information, as the answering of a being's Informing processing to external, sensory information, and as sensory consequences, which are recognized through a being's behaviour in a given environment. Even the environment is igniting the growth of the information structure and this structure remains characteristic of the being according to information accents, specific forms, and meaning. In this structural context, internal language, which is a being's mind conversing with itself, and the development of internal language are coming into existence. But this is only one of the possible models of being's information linguistics.

A being's language is a primary information phenomenon of a being. It is the essence of a being, of a being's accepting, expressing or behaving in its information context and in its environment. In general, a being's language has many layers and great depth, which is organized hierarchically and is structurally characteristic. Up to here, a being's internal language is a language of thought. At this point, a being's external language begins, by which a being is informing its environment, receiving its own Informing, and informing itself from the environment. In comparison with internal language, the external language is a secondary affair, which is only an intermediate step of understanding. External language, which is a being's strange phenomenon, is an information mechanism between the internal language, which is a being's native language, and the environment. The relationship between linguistic information domesticity and linguistic information strangeness is conditioning the originality of being's counter-Informing. Strangeness is immanent to the external language and domesticity to the internal language; the difference between these languages is always information essential.

Sensory, natural, and certainly formal languages of the environment comprise a being's external language. These languages are used by a being for its sensory-linguistic Informing. All these languages have a secondary value according to a being's internal language and a being uses them mainly in an information context with other beings, machines, and the environment.

5

... pokazali bomo, kako lahko prehod iz racionalistične v heideggrsko perspektivo radikalno spremeni naše pojmovanje računalnikov in naše metode pri računalniškem oblikovanju. ...

(Terry Winograd in Fernando Flores: Razumevanje računalnikov in spoznavanja: novi temelji oblikovanja, 37)

Kaj je informatizacija, kaj informatiziranje? Informatizacija je najsplošnejši proces nastajanja informacije; ta proces je lahko samonastajalen, notranjeinformacijski, lahko je pa tudi zunanjenastajalen, nastajajoč zaradi zunanjih, senzorskih informacij. Informatiziranje je posledica informacije, ki spreminja sebe in druge informacije na svoji poti, s svojo procesnostjo k novi informaciji.

V informacijski interakciji lahko bitja medseboj le informatizirajo; informatizacijski pojav med bitji je posledica večkratnega preoblikovanja informacije v bitjih na poti do bitjevskih korteksov, kjer prihaja do opomenjena informacijskih pojavnosti. Ko bitje sprejme sporočilo prek svojega drugotnega jezika, prihajajočega po čutnih poteh, se to sporočilo-informacija ne samo preoblikuje, povzroča tudi nastajanje obrobnih informacij in njihovo integracijo v kontekst prvotnega sporočila-informacije. Sprejemanje sporočila sproža v bitju nastajanje informacije, informatizacijo sporočila, njegovo predelavo v bitjevski notranji jezik, skratka regularno informatizacijo. Pri tem je za razumevanje sporočila še kako pomemben informacijski kontekst v obliki notranjega modela sveta (prepričanj, napovedi), filtrirnih in modulacijskih mehanizmov, čustev in volje (trenutnega obnašanja).

Bitja tako vselej informatizirajo in ne zmorejo komunicirati v čistem pomenu. Komunikacija bi pomenila bolj ali manj natančno sprejemanje tujega mišljenja, tuje zavesti po obliki in pomenu. Takšen informacijski prenos informacije med bitji funkcionalno v informacijsko procesirajoči živi substanci ni mogoč, kar se očitno kaže že na ravni molekularne "komunikacije". To kar bi bilo mogoče razumeti kot komunikacijo na korteksni ravni, je le še zaupanje drugim bitjem, avtoriteti in pričevanju. Komuniciranje v čistem pomenu bi bilo abstraktno informatiziranje brez informacijskih primesi, kar pa zaradi nevrofiziološkega ustroja bitja ni mogoče. Komunikacija pa je mogoča med stroji in sporočila so lahko zapisana na papirju v naravnem ali formalnem jeziku tako na oddajni kot na sprejemni strani. Komunikacija je informacijsko stabilen pojav, v celoti napovedljiv, pričakovan; informatizacija je vobče nenapovedljiva, nepredvidljiva, svobodna, pogojena z ustrojem, stanjem in pomenom bitjevskega notranjega jezika. Delno napovedljiva je le informatizacija ustrahovanega, avtoritarno vplivanega, indoktriniranega, ideologiziranega, informacijsko usmerjenega, komunikacijsko sprejemljivega bitja, ki svojo informatizacijo zakriva, jo podvaja, razceplja za različne uporabe. V tem primeru je informatizacija bitja popačena, zanj izkrivljena, patološka. Že na ravni molekul življenja, npr. molekul imunskega sistema bitja je mogoče razumeti, kako sta zdravje in bolezen informacijska in predvsem tudi informacijsko

5

... we will show how shifting from a rationalistic to a Heideggerian perspective can radically alter our conception of computers and our approach to computer design. ...

(Terry Winograd and Fernando Flores: Understanding Computers and Cognition: A New Foundation for Design, 37)

What is Informing? Informing is the most general process in which information is coming into existence. This process can be self-developing, internal information or it can be external information which comes into existence as a consequence of external, sensory information. By processing itself into new information, Informing is a consequence of information, which is altering the initial information, itself, and other information on its way.

In interactions among information, beings can only inform each other. This Informing phenomenon among beings is a consequence of multiple transformations of information on the way to a beings' cortices, where the meaning of information phenomena is coming into existence. When a being receives a message in the form of its secondary (external) language propagating on sensory paths, this message-information is not only being transformed, but it also causes the growth of additional related information and their integration into the context of the originally received message-information. Receiving a message fires the growth of information, and brings new information into existence, which is the so-called message Informing. More precisely, reception of a message transforms the message into the internal language, and performs Informing as a regular way of information. At this point, the information context of the internal world model (beliefs, forecasts), filtering and modulating mechanisms, and emotions and will (instantaneuos behaviour) are of essential importance for understanding the message.

Beings, incapable of communicating in a pure sense, are always Informing. Communication would be more or less the exact receiving of a strange thought or of a strange consciousness by form and meaning. This type of information transfer among beings in their processing substance of living information cannot be realized; this is becoming evident even on the level of molecular "communication". That, which is possible to understand as communication on the cortical level, is only the confidence of a being to messages coming from other beings, authority, and persuasion. Communication in a pure sense is an abstract Informing without information additions. This way of Informing is not possible because of the neurophysiological structure of a being. Pure communication can be realized only among machines. Messages, which can be exchanged between machines, are written in a natural or a formal language. Communication is an information stable phenomenon, which can be forecasted and expected in its whole. Informing, in general, cannot be forecasted or foreseen; it is free and conditioned only by its structure, state, and the meaning of a being's internal language. Only the Informing of a being which is fearful, authority influenced, indoctrinated, idealized, information directed, and communication acceptable can be partly forecasted, when this Informing is partly hidden, duplicated, and split for various usage. In this case, the Informing of a being is distorted, bent, and pathologically processed in itself. On the level of living mopogojena organizacijska problema bitjevskih celic, njihovih populacij in okolij.

Informatizacija, ki razvija informacijo, ki je proces informatiziranja ali informacijski proces, je tudi sama informacijska, je tudi sama informacija za sebe in za druge informacije. Kaj je v bistvu s to formulo nakazano? Informatizacija, ki je informacijski proces za nastajanje informacij iz informacij v najširšem pomenu, lastnoinformacijskih, tujeinformacijskih, okoliških, se tudi sama razvija, nastaja iz tega, kar jo informacijsko zadeva, v to, kar z njeno transformatičnostjo in njeno nastajalnostjo nastaja. Z informacijo nastaja tako hkrati tudi njena informatizacija, nastaja tudi nastajanje informacije. V tehnološkem jeziku bi to pomenilo, da s transformacijo podatka v nov podatek nastaja tudi transformacija te njegove transformacije. Zaradi te lastnosti je dosežek informatizacije v bistvu običajno nenapovedljiv; informacijsko nastajanje je nenapovedljivo in sama napovedljivost v informaciji je nastajajoča ali biološko prilagajajoča. Bitje je informacijski stroj za preživljanje. Tehnološkemu stroju manjka celotna informatičnost na aparaturni in programski ravni, v njegovi mikro in makro strukturi. Tehnološki stroj ni v nobeni svoji komponenti funkcionalno informacijski, še ni informacijski stroj.

Informatizacija bitja se začenja z njegovim biološkim spočetjem. Vsaka informatizacija ima svoje inicialne informacije iz katerih se kot informacijska lastnost razvija v živi, njej prirejeni, z njo vplivani nastajajoči substanci. Bistvo bitja, njegova smiselnost, slastnost življenja je njegova informatizacija: z njo se v bitju razvija informacija in informacijsko orodje za nastajanje informacije, ki je tudi samo informacija. Za informatizacijo je v slehernem trenutku inicialna vsa v bitju in vanj prihajajoča tenutna informacija. Spočetje samo je le začetek nastajanja informatizacije zarodka, ki se skupaj s svojo informatizacijo substančno, biološko razvija. Bit informacije je zajeta v informatizaciji; v njej je vse, kar je informacijsko relevantno, kar z informatizacijo lahko nastane. Informatizacija je tako informacijski potencial in informacijski eksistencial.

Ker je informacijski potencial informacijsko neomejen, saj je v njem vsebovana vsaka možna in še nemožna informacija, nastaja vprašanje, kaj je informacijsko nemogoče. Informacija se naposled sooči s svojo smrtjo, natančneje s smrtjo bitjevske informacije. Smrt namreč omejuje brezkončno nastajanje informacije kot bitjevske informacije. Informacija se kot pojavnost živega sooča s svojo posebno informacijsko obliko, ki je tesnobnost končnosti. Smrt je končnost, prenehanje, izginitev, umrtje bitjevske informatizacije, je meja, do katere informatizacija lahko nastaja s svojimi informacijami. Bivanje je torej informatizacijsko v sebi in drugih. Tu velja: le dokler informatiziram, sem.

lecules, e. g. molecules of the immune system, Informing is also unforeseeable. Here, it is possible to understand how health and disease are information dependent and how they are information conditioned problems of the organization of a being's cells, of cell populations, and of the environment.

Informing, which develops information, which in turn is a process of Informing or an information process, is also information by itself and for other information. What is the meaning of this formula? Informing, which is an information process for information coming into existence in the broadest sense, i. e. own, strange and environmental information, is also developing and coming into existence from everything which is information relevant. By its transformations, Informing's coming into existence is coming into existence. Simultaneously, through information, Informing which is caused by information is coming into existence and therefore, also the coming into existence of information is coming into existence. In technological language, this would mean that by the transformation of data into new data, also the process of data transformation is coming into existence. Because of this property, the result of Informing cannot actually be foreseen in a usual way. Information coming into existence cannot be forecasted because forecasting within information is coming into existence or adapting biologically. A being is an information machine for survival. In technological machines, the property of informing on a hardware and software level in either a technological micro or macro structure is not currently present. No component of a technological machine is functionally "information-like", i. e. a technological information machine does not yet exist.

A being's Informing has its roots in its biologic conception. Each Informing has its own initial information, from which it develops and adapts according to the information property in life and by this property the developing substance is influenced. The essence of a being, the being's sense, and delight of its life is the being's Informing. In a being, information and information tools (which are also information) for information development are coming into existence through Informing. For Informing, the entire information of a given time realm which is both existing in a being and arriving to a being is initial. Conception is only the beginning of Informing as it comes into existence in the embryo. So, the embryo is developing together with its own substantial and biological Informing. The Information Being is included in Informing. Informing contains everything which is information relevant and all that comes into existence through Informing. By this, Informing is information potential and information existence.

Because the information potential has no information limits and because it includes both all possible and impossible information, the question which arises is, what information is impossible. Eventually, information is being faced with its death or more precisely, with the death of a being's information. Namely, this death is limiting the unlimited information which is growing as a form of a being's information. As a phenomenon of life, information is faced with a particular information form, which is the anxiety of the end. Death is the end, the ceasing, disappearing, and dieing of a being's Informing. It is the limit, to which Informing by its information comes into existence. Thus, a being, Being, and existing are all Informing in themselves and in others. Hence, the following is essential: as long as a being is informing, it is being.

6

... Nas Japonce ne začuduje, če pušča pogovor dejansko mišljeno v nedoločnem, celo povratno skriva v nedoločljivo. ...

(Martin Heigegger: Iz pogovora o jeziku. UZS, 100)

Kaj je informacijski stroj? Zakaj se kajstvo stroja tu sploh pojavlja? Kakšna je v okviru tega vprašanja bistvenost informacijskega?

Današnji računalniki so dosegli svoj višek v arhitekturni in podatkovni rekurzivnosti, v logični in algoritmični strukturiranosti svojih modulov, v organizaciji svoje fizične, logične, abstraktne in programske strukture. Računalniki so s tem že dosegli zgornjo mejo t. i. informacijske statičnosti, podatkovnosti, katere višek je strojna abstraktnost in jezikovna formalnost računalniške uporabe. Današnji računalniki so primerni za reševanje problemov informacijske statičnosti, podatkovne dinamičnosti, informacijske trdnosti, ki obsegajo tudi današnje koncipiranje paralelnih računalniških sistemov, paralelno procesiranje, umetno inteligenco, ekspertne sisteme, pa tudi novogeneracijsko tehnologijo z visoko integracijo, statično aparaturno arhitekturo itd. Višek današnje uporabe statičnih informacijskih strojev predstavljajo formalni (programirni) jeziki, ki so seveda bistveno reducirani jeziki za opisovanje bistveno informacijsko reduciranih problemov. Ti formalni jeziki so glede na bitjevske sekundarne jezike, kot so naravni, mimični in kretenjski jeziki, le nekašnji ne-jeziki, brez-jeziki, slepi jeziki, le tehnična orodja, ki jih bitje lahko uporablja tako, kot vsa druga, vsakdanja priročna orodja.

S pojavitvijo umetne inteligence in kasneje s projekti novih računalniških generacij je stopilo v ospredje vprašanje t. i. inteligentnih strojev in inteligentnih programov. Široki vidiki inteligence pa prav gotovo in čedalje bolj očitno posegajo na področje informacije, informacijske oblike, informacijskega procesa, četudi se večkrat reducirajo na nekaj, kar se vseskozi opravičuje kot zgolj umetno, kot umetna, nenaravna, neinformacijska inteligenca. Inteligenca pa je po svoji naravi vsaj informacijsko dinamičen pojav, značilna informacijska oblika v živem. Podatkovna ali "umetna" inteligenca današnjih strojev in programov je le statična, algoritmična inteligenca, v bistvu ne-inteligenca, brez-inteligenca ali značilna slepa inteligenca.

V okvir vprašanja inteligence spada tudi vprašanje jezika, ki je večplastno in raznovrstno. Jezik je abstraktni pripomoček uporabe današnjega in jutrišnjega stroja, danes predvsem formalnega, brez-inteligenčnega stroja, jutri pa inteligenčnega oziroma splošneje informacijskega stroja. Obstaja več ravni, plasti jezikovne predstavitve v stroju, saj ima današnji stroj svojo fizikalno, logično in abstraktno strukturo. Čeprav je strojna fizikalna struktura ena sama, so v tej strukturi možne različne jezikovne predstavitve, predstavitvena vgnezdenja, ki so na najnižji ravni fizikalnoprocesne, na nekoliko višji ravni logičnoelementarne in naposled jezikovnoabstraktne. Načrtovanje strojev mora obvladovati vse te ravni, na abstraktni ravni pa mora upoštevati zlasti možnosti strojne uporabnosti, tj. strojne in uporabniške jezikovnosti, njune združljivosti.

6

... Uns Japaner befremdet es nicht, wenn ein Gespraech das eigentlich gemeinte im Unbestimmten laesst, es sogar ins Unbestimmbare zurueckbirgt. ...

(Martin Heidegger: Aus einem Gespraech von der Sprache. UZS, 100)

What is an information machine? Why is the querying which concerns this machine coming into existence at all? What is the essence of information in this context?

Present-day computers have attained their zenith in architectural and data recursivness, logical and algorithmical structure of their functional modules, and in the organization of their physical, logical, abstract, and programming levels. Thus, computers have reached the upper limit of the so-called information statics or data processing, whose culminating points are the machine abstraction and language formality of computer usage. Today's computers are suitable for solving problems of information statics, data dynamics, and information solidness, all of which embrace today's design of parallel computer systems, parallel processing, artificial intelligence, expert systems, and also new generation technology using very large scale integration, static hardware architecture, etc. The highest point of static information machine usage is formal (programming) languages. These are of course essentially reduced languages and they only describe problems whose essential information is reduced. With re-gard to a being's secondary languages, e. g. natural, mimic, and gesture languages, formal languages are only examples of non-languages, crippled-languages or blind languages. They are only technical tools, which a being can use like other everyday skillful tools.

With the merging of artificial intelligence and current projects on new computer generations the question of the so-called intelligent machine and intelligent program came into the foreground. Broad aspects of intelligence are certainly more and more drawn into the realm of information, information form, and information process. However, they are repeatedly being reduced to that which is continuously being exculpated as only an artificial, non-essential, and non-information intelligence. But intelligence in its nature is at least an information dynamic phenomenon and a characteristic information form of life. In their nature, data or "artificial" intelligence of today's machines represent only non-intelligence, crippled-intelligence or characteristically blind intelligence.

The framework of intelligent problems also concerns the language problem, which is a multilayered and heterogeneous one. Language is an abstract resource of today's and tomorrow's machine usage; today it is a tool for formal and semi-intelligent machine usage, but tomorrow it will be for intelligent or, more generally, for information machine usage. There are several levels and layers of language representation in a machine, because today's computers have their physical, logical, and abstract structure. Though each machine has only one physical structure, in this structure several language representations and presentation embeddings are possible. On the lowest level are physical processes, on a higher level logical elements and at highest level language abstractions. Not only must the design of machines master all these levels, more importantly it must consider the the possibilities of machine

Problem nastajanja informacijskega stroja je v postopnem prehodu iz tehnologije današnjega podatkovnega, ne-informacijskega stroja na tehnologijo jutrišnjega informacijskega, inteligentnega stroja. Fizikalno strukturo informacijskega stroja je mogoče še znatno dopolniti z raznovrstnimi novimi fizikalnimi elementi pa tudi z (biološkimi) nadlogičnimi elementi. V fizikalni strukturi današnjih strojev se pojavljajo pretežno prevodniški in polprevodniški elementi statične, stabilne narave, manj pa magnetni in elektromagnetni elementi dinamične, nestabilne narave (dinamični prenosniki, antene, valovodi), molekularni elementi (DNA, RNA, proteini, encimi) in tudi višji biološki mehanizmi (celični, celičnopopulacijski).

7

... Celo tako majhen organizem kot je črv z nekaj sto nevroni, je visoko strukturiran in večji del njegovega obnašanja je posledica vgrajene strukture, ne učenja. ...

(Terry Winograd in Fernando Flores: Razumevanje računalnikov in spoznavanja: novi temelji oblikovanja, 103)

V okviru informatičnosti stroja prihodnosti je mogoče oblikovati več bistvenih vprašanj. Novi računalniki lahko postanejo informacijski v vseh svojih strukturah; te strukture so fizikalne, logične, abstraktne in jezikovne. Značilni pomen informatičnosti se skriva v informacijski dinamičnosti, spremenljivosti, tudi v umetni nekonstantnosti, nedoločljivosti teh struktur. Dinamičnost naštetih struktur je mogoče dosegati z dinamičnostjo na najnižji strukturni ravni, tj. z dinamično fizikalno strukturo stroja. Tu se seveda (vsaj zaenkrat) ne more pojavljati nastajanje fizikalne strukture; to praktično, tehnološko še ni mogoče, je pa lahko umetno tako, da temelji na funkcionalnem vključevanju in izključevanju posameznih fizičnih elementov in podsistemov, pri čemer sta vključevanje in izključevanje krmiljena s signali, podatki, sporočili na različnih strukturnih ravneh. Na ta način bi bilo mogoče že danes konstruirati signalno odvisno strukturo računalnika podobno, kot se to predlaga za paralelne superračunalniške arhitekture, vendar le za mrežno komuniciranje.

Informacijsko krmiljenje fizikalne arhitekture je v bistvu modeliranje dinamične strojne arhitekture s preklapljanjem na najnižji ravni in s tem tudi na logični in abstraktni (ukazni, zbirniškojezikovni) strojni ravni. Informacijski stroj potrebuje tudi dinamično jezikovnost, to pa je lastnost, ko se jezikovno izraženi programi lahko pomensko oziroma problemsko modificirajo v procesu svojega izvajanja. Ta princip dinamične programirljivosti programov je nekje v načelnem nasprotju s principi t. i. dobrega ali strukturnega programiranja, ki je programiranje z vgrajevanjem permanentne informacijske slepote. To, kar naj bi postalo informacijsko programiranje, se mora konceptualno šele razviti skupaj z orodji, ki bodo omogočala preizkušanje in uporabo informacijskih programov.

applicability on the language abstraction level, i. e. compatibilities of machine and languages has to be taken into account.

The problem of designing an information machine lies in a step by step transition from today's data and non-information machine into tomorrow's information and intelligent machine technology. The physical structure of an information machine can be improved considerably by various new physical components and also by higher logical (biological) elements. In the physical structure of today's machine, conductor and semiconductor elements of static and stable nature are predominant. In future machines also magnetic and electromagnetic elements of a dynamic and unstable nature (dynamic transducers, antennas, and wave guides), molecular elements (DNA, RNA, proteins, and enzymes), and higher biological mechanisms (cell and cell population-like) have to be applied.

7

... Even an organism as small as a worm with a few hundred neurons is highly structured, and much of its behaviour is the result of built-in structure, not learning. ...

(Terry Winograd and Fernando Flores: Understanding Computers and Cognition: A New Foundation for Design, 103)

In the framework of future machine Informing, several important questions may arise. New machines can become informative (information-like) in all their structures, which are physical, logical, abstract, and linguistic ones. The characteristic meaning of Informing is hidden in the information dynamics, variability, artificial non-constancy, and non-determination of these structures. Dynamics of these structures can be achieved through dynamics on the lowest structural level, i. e. by the dynamic physical structure of a machine. At this level, an arising of physical structure is not possible (yet). Such an arising is practically and technologically impossible, but it can be artificially constructed by the functional switching-on and switching-off of particular physical elements and subsystems. Here, switching is controlled by signals, data, and messages on different structural levels. In this way, even today it would be possible to construct a signal dependent computer architecture, which could be similar to some proposals of a parallel supercomputer with dynamic architectures.

Information control of a physical architecture is more precisely the modeling of dynamic machine architecture using switching on the lowest structural level. In this way the logical and abstract machine level (instructions, assembly language) become dynamic. But an information machine also requires dynamic linguistics as a property, where linguistically expressed programs can be meaningly and problematically modified in the process of their machine execution. This principle of dynamic programming for information programs is in fundamental opposition to the principles of the so-called good or structured programming, which requires programming with a permanently built-in information blindness. What is termed as information programming has to be developed together with the tools which will enable the testing and application of information programs.

An information machine can have its pseudo-information structure and thus, logical and abstract structure, in which information-like informacijski stroj ima lahko psevdoinformacijsko fizikalno in s tem logično in obstraktno strukturo, s katero sprejema v izvajanje informacijsko izražene programe, ki so lahko napisani v novih, še neznanih generativnih programirnih jezikih. Ti novi jeziki ne morejo biti več značilno formalizirani, kot so današnji programirni jeziki, saj morajo dosegati neformalnost, oblikovnost in pomenskost na ravni naravnih jezikov. Informacijska fizikalna struktura naj bi bila obogačena z raznovrstnimi logičnimi in analogizmičnimi (iz grškega ana-logismos) elementi tako, da bo delovala kot dinamična arhitektura.

8

... hermenevtični krog. ... Kar razumemo, je utemeljeno s tem, kar že znamo in kar že znamo, prihaja iz sposobnosti razumevanja. ...

(Terry Winograd in Fernando Flores: Razumevanje računalnikov in spoznavanja: novi temelji oblikovanja, 30)

Kdo bo lahko prvi zgradil informacijski stroj? Kakšne osnovne raziskave bodo pri tem potrebne? Kakšna je lahko nova tehnološka podlaga?

Na ta vprašanja je bilo delno že odgovorjeno. Informacijski stroj bo imel vsekakor dinamično fizikalno strukturo, s tem pa bosta postali dinamični tudi njegova logična in abstraktna struktura. Dinamična fizikalna struktura bo dovolj enakomerno porazdeljena v celotnem tehnološkem kompleksu informacijskega stroja. Na določen način bo porazdeljen skladno z dinamično logično strukturo tudi pomnilnik, saj je pomnenje (spominjanje) bistvena lokalna lastnost žive informacije. Ko bo informacijski stroj enkrat zgrajen in eksperimentalno razpoložljiv, bodo omogočeni nekateri fundamentalni informacijski preizkusi, kjer se bodo lahko pokazale specifičnosti informacijske programske opreme. Novi programi bodo informacijski in generirali bodo vobče procese s povratnimi vplivi na prvotno, začetno, izhodiščno pomenskost programov. Pomen programa se bo med njegovim izvajanjem spreminjal, naraščal, krčil, nastajal v odvisnosti od lastnega izvajanja in od paralelnega izvajanja drugih programov. Podatki, ki bodo nastajali z izvajanjem informacijskega programa, bodo kot signali, vrednosti in sporočila krmilili dinamično izvajalno strojno arhitekturo. Informacijski stroj se bo z izvajanjem informacijskih programov postopno približeval temu, kar bo podobno nastajanju informacije v živi substanci. Takšen informacijski stroj bo lahko le rezultat skupinskega dela filozofov s področja informacije, tehnologov z novih področij fizike in biologije, izvedencev za logično-tehnološko načrtovanje informacijskih arhitektur, programerjev za informacijsko programiranje itd.

Osnovne raziskave za izdelavo informacijskega stroja bodo obsegale filozofijo, novo' teorijo in tehnologijo informacije, filozofijo tehnološko in realizacijsko mogočega, raziskave orodij za načrtovanje dinamičnih arhitektur in za razvoj generativnih programov. Pri tem je potrebno poudariti, da bo dinamična arhitektura po svoji naravi paralelna, da bo omogočala potencialno neomejeno izvajanje paralelnih procesov in da bodo generativni programi pri svojem izvajanju povzročali začenjanje (nastajanje) novih paralelnih procesov. Procesnost arhitekture in procesnost paralelnega programskega izvajanja bosta vobče nepredvidljivi;

programs can be executed. The information programs would be written in new languages which are generative and today still unknown. These languages cannot be characteristically formalized, as are today's programming languages, because they must achieve non-formality, shaping, and meaning. These properties are also found in natural languages. The information-like physical structure has to be enriched by various logical and analogismic (from Greek ana-logismos) elements, in order to function as a dynamic architecture.

8

... hermeneutic circle. ... What we understand is based on what we already know, and what we already know comes from being able to understand. ...

(Terry Winograd and Fernando Flores: Understanding Computers and Cognition: A New Foundation for Design, 30)

Who will be the first to have the capability of building an information machine? Which fundamental research will be necessary for the implemetation of such a project? What will be the new technological foundation?

Partial answers to these questions have already been given. As stated, an information machine will have a dynamic physical structure and this structure will sufficiently be distributed equally within the entire technological structure of this information machine. In a similarly determined way, according to the dynamic logical structure, the memory will be distributed. This distribution of storing (memorizing) is as essential in the machine as it is in living information. When an information machine is already built and experimentally functioning, some fundamental information experiments will become possible. This will bring particular characteristics of information software into view. New programs will be information-like and, in general, they will generate processes which have a continuous influence on the meaning of the program at its original starting point. The meaning of a program will be altered, increased, and contracted, i. e. arising-dependent in the parallel execution of this and other programs. By means of signals, values and messages, data which arises during the execution of an information program will control the dynamic machine architecture. In executing information programs, the information machine gradually approaches a state which is similar to information coming into existence in a living substance. Such an information machine will only be the result of co-operation among philosophers developing information philosophy, technological engineers applying new components of physics and biology, experts for logical-technological design of information architectures, programers for information programming, etc.

The basic research for the implementation of an information machine will comprise new philosophy, theory, and technology of information. Also a new philosophy in regard to information-like, technological, and implementational feasibility is needed. Lastly, new research of tools for dynamic architecture and generative program design, etc. is necessary. Here, it is important to point out, that the dynamic architecture will be paralleled, thus enabling both a potentially unlimited execution of parallel processes and the execution of generative programs which will cause the arising (beginning) of new parallel processes. In general, architeinformacijski stroj se bo s to svojo lastnostjo lahko čedalje bolj približeval mehanizmom žive informacije.

Tehnološka osnova informacijskega stroja bo v marsičem inovativna. Uporaba elementov fizike trdne snovi (prevodniki, polprevodniki, magnetika, elektromagnetno polje, supraprevodnost, molekularni in kvantni pojavi itd.) in bioloških substanc bo omogočila potrebno funkcionalno raznovrstnost na fizikalni in logični ravni informacijskega stroja.

9

... Inteligenca je del informacije, njen bistveni mehanizem: tako narekujejo višje možganske funkcije, višja oblika informacije v informacijski hierarhiji. ...

(Anton P. Železnikar: Razvoj nesposobnosti, P12)

Kaj je inteligenca drugega kot informatičnost? Kakšna je posebnost inteligence kot informacije? Ti vprašanji sta nujni, ko je potreben odgovor, kaj je inteligenca v živem in kaj naj bi bil inteligenten stroj.

Inteligenca je poseben informacijski proces, ki nastaja kot posledica dane problemske nastajalnosti, v kateri se problem razrešuje. Inteligenca je problemskorešavalni informacijski proces, katerega informacije se uporabljajo za problemsko reagiranje oziroma za obnašanje bitja v problemski nastajalnosti. Inteligenca je problemskoreaktivna informacija in njen smisel je, da k danim vhodnim ali začetnim informacijam generira novo, od problemske in bitjevske informacije odvisno protiinformacijo, ki nastaja z razvojem problemske zavesti in s problemskim razvojem v realnem času.

Informacija brez inteligence v bistvu ne bi bila več tista značilnost, ki je lastna prav informaciji. Če je smisel informacije njeno informatiziranje, je smisel inteligence v informaciji neko problemskorešavalno in ciljnousmerjeno informatiziranje. Vendar, kaj pa je informacija drugega, kot prav to, kar je inteligenčna značilnost? Ali ni življenjska vloga inteligence kot informacije v preživljanju, v obstoju in razvoju bitja? Inteligenca se tedaj pojavlja kot informacijsko smiselna oblika same informacije, katere smisel je prejle opisano informatiziranje v osnovnejšem, splošnoinformacijskem informatiziranju.

Inteligenca v živem je strategija preživetja. Strategija je razumljena vselej kot vodilna, usmerjevalna, nadzorna, občutljiva, ciljnousmerjena, problemskorešavalna informacija. Smisel inteligence je nastajanje določenega informacijskega kompleksa, s katerim se aktivirajo podinformacije in dovolj splošna informacija bitja, z njimi pa se dosegajo trenutne, inteligenčno smiselne informacije; te sprožajo ustrezno bitjevsko preživetveno obnašanje. Prav tu pa se kaže, kako je inteligenca informacijska regularnost, vrženost v informacijsko strukturo bitja na različnih informacijskih ravneh te strukture, od perifernega živčnega sistema do bitjevskih korteksov.

Inteligenca lahko tedaj nastaja le kot informacija, ja značilno nastajajoč informacijski pojav. Inteligenca ne more nastajati in s tem obstajati kot nastajanje, če nima informacijsko nastajajoče podstati. Ta razmislek je potreben zaradi tega, ker je nemogoče govoriti o pojavu inteligence, če niso izpolnjeni osnovni pogoji informacijskega nastajanja. Hierarhija inteli-

cture processing and processing of parallel programs will be unpredictable. Through these properties, an information machine will continue to approach the mechanisms of living information.

9

... Intelligence is a part of information, an essential information mechanism which is commanded by higher cerebral functions, higher forms of information in the information hierarchy. ...

(Anton P. Železnikar: Development of Incapability, P12)

What is intelligence other than Informing? What is the peculiarity of intelligence as information? These questions are indispensable when addressing such questions as: what is intelligence in living beings and what should the intelligent machine be.

Intelligence is a particular information process which comes into existence as a consequence of a given problem phenomenon within which the problem is being solved. Intelligence is a problem solving information process, where information is being used for problem reacting or a being's behaviour in an arising problem. Intelligence is problem reactive information and its purpose is to generate new problem information and counter-information from given input or beginning information. This counter-information arises through the development of problem awareness and problems arising in a real time domain.

Information without intelligence would no longer be the characteristic way which is proper to information. If the essence of information is in its Informing, then the essence of intelligence within information is in problem solving and goal directed Informing. But, what can information be other than what is primarily characteristic for intelligence itself? Is the role of intelligence in life not like a being's survival, existential, and developmental information? Evidently, intelligence is an information-essential form of information, whose essence was described as Informing within a more basic and general Informing of information.

Intelligence in life is a strategy of survival. A strategy can always be understood as leading, directing, supervising, sensitive, goal-directed, and problem solving information. The essence of intelligence is arising from a particular information complexity, by which a being's sub-information and sufficiently general information are activated. This information has access to temporal and intelligence related information, which triggers the being's survival behaviour. At this point, it can be shown how intelligence is becoming a being's information regularity and is being thrown into the information structure on different information levels within this structure, i. e. from the peripheral nervous system to a being's cortices.

Thus, intelligence coming into existence only as an information process is a characteristically regular information phenomenon. Intelligence cannot arise and exist as arising if it does not have an information-like arising substance. This consideration is necessary, because it is not possible to speak about the phenomenon of intelligence if some basic conditions of information arising are not present.

genčnega nastajanja je tedaj tale: najprej nastajanje informacije in šele v okviru tega nastajanja nastajanje inteligence.

Dokler t. i. inteligentni stroji niso informacijski, je tudi nemogoče govoriti o njih kot o inteligenčnih strojih. To so le stroji, katerih avtorji si želijo, da bi ti stroji imeli videz inteligenčnosti, da bi zbujali vtis in v določenih primerih reagirali podobno, kot inteligentna bitja. Umetna inteligenca, ki je trdno, algoritmično in interaktivno domišljano znanje, pri tem ne more bistveno inteligenčno napredovati, saj formalizmi matematizacije (npr. drevesologije, kombinatorike), algoritmizacije (neinformacijskega programiranja oziroma strukturirne programirne metodologije) in interaktivizacije (povezave stroja in bitja) v umetnointeligenčnih procesih niso informacijsko nastajajoči, so le statične strojne zmogljivosti. Umetna inteligenca bo morala najprej razbiti svoje vrednostne tablice in napisati nove, doživeti svojo dekadenco in preživeti svojo inteligenčno krizo in sestaviti novo osnovo, da se bo lahko dejansko ukvarjala s problematiko, ki je inteligenčna. To pa se bo v skladu z razvijajočo informacijo lahko zgodilo v naslednjih sto ali tisoč letih.

10

Pri pisanju tega članka v slovenščini sem nastajajoče besedilo prevajal tudi v angleščino, v kateri pa sem si dovoljeval le uporabo veljavnih angleških besed, ki se nanašajo na informacijo in njene izpeljanke (The Oxford Dictionary). Seveda pa angleški prevod ne bi dosegel jezikovne zadostnosti in vsebinske pretanjenosti brez pomoči Johna D. Freyderja, filozofa po izobrazbi. Z njim sem pri lekturi svojega prevoda lahko tudi razpravljal o vsebinsko problematičnih delih rokopisa. Za ta njegov bistveni prispevek se mu hvaležno zahvaljujem.

Andrej Bekeš je prevedel povzetek tega članka v japonščino. Tudi njemu izrekam svojo zahvalo.

11

P. L. McGeer, J. C. Eccles, E. G. McGeer: Molecular Neurobiology of the Mammalian Brain. Plenum Press. New York (1978). ISBN 0-306-31095-3.

M. Heidegger: On Time and Being. Harper & Row. New York (1972). ISBN: 0-06-131941-4.

M. Heidegger: Unterwegs zur Sprache. Neske. Pfullingen (1959).

D. F. Lindsley, J. E. Holmes: Basic Human Neurophysiology. Elsevier. New York (1984). ISBN: 0-444-00797-0.

M. Uršič: Enivetok (filozofski esej). Zbirka Znamenja. Obzorja, Maribor (1981).

T. Winograd, F. Flores: Understanding Computers and Cognition: A New Foundation for Design. Ablex Publ Corp, Norwood, NJ (1986). ISBN: 0-89391-050-3.

A. P. Zeleznikar: Razvoj nesposobnosti (Development of Incapability). Neobjavljena zbirka arabesk (Non-published collection of arabesques) (1986).

So, some hierarchy or embedding of intelligence arising exists. Fundamental to this hierarchy is information arising and within this arising is the arising of intelligence arising.

As long as the so-called intelligent machines are not information-like, it is not possible to speak about them as being intelligent. These are only machines, whose designers wish to give them an image of intelligence, to awaken impressions and in some way to have them react similarly to intelligent beings. Artificial intelligence, which is a hard, algorithmic, and interactive discipline, cannot make any essential progress into real intelligence. Formalisms of mathematics (e. g. tree-methodology, combinatorics, fuzzy sets, etc.), algorithms (non-information or structured programming methods), and interaction (man-machine communication) which are used in artificial intelligence processes have not the nature of arising, and thus are not information-like. They are only performing as static machines. First, artificial intelligence has to break its hard tablets of value into pieces and it must write new ones. It has to live through its decay and survive its crisis of intelligence. It must also assemble a new foundation, so it can deal more precisely with problems of intelligence. All this could happen according to arising information in the next hundred or thousand years.

10

When writing this article in Slovene, the arising text was translated into English using only the legally available English terms concerning information and their derivatives (The Oxford Dictionary). Of course, the English text would never reach a linguistic satisfication and refinement of its content without the kind help of John D. Freyder. The author has had the opportunity to discuss the problematical parts of this article with him. Andrej Bekeš has kindly translated the abstract of this article into Japanese. To both, the author expresses his very grateful thanks.

DESIGNING A RECONFIGURABLE INTELLIGENT MEMORY MODULE (RIMM) FOR PERFORMANCE ENHANCEMENT TO LARGE SCALE, GENERAL PURPOSE PARALLEL PROCESSOR

INFORMATICA 1/87

Petar Brajak
ISKRA DELTA, Ljubljana

UDK 681.327.6.067

ABSTRACT: This paper will present two approaches in developping large scale parallel processing system. First approach we call the **INHERENT** approach. It is primarely concerned with developping parallel system using standard products available on the market. The other approach we call the **NOVEL** approach. With this approach, the design of a large scale parallel processing system is solely based on the custom build components.

This paper is mostly concerned with the INHERENT approach. We give full and detailed architectural and organizational concept in transforming and adapting standard available microprocessor system into the parallel processing environment. We describe a detailed design of the easy to add **RECONFIGURABLE INTELLIGENT MEMORY MODULE.** This module is the key component in approaching and achieving high goals that we describe as the main principles in designing a large scale, general purpouse, fault tolerant, parallel processor. We notice some deficiences of the system designed using the INHERENT approach. These deficiences are the result of a hard adaptability of the market available products to the parallel processing environment. Therefore, we offer the NOVEL system design approach that eliminates those deficiencies.

Key words: **parallel processing, tigtly and loosly coupled multiprocessors, computer architecture, interconnection networks, fault tolerance, content adressable memories, routing, memory contention, multiplexing, interleaving, random access, queueing, algorithms**

＜要約＞本稿では大規模な並列処理システム開発に関して、二つのアプローチの可能性をしめす。一つのアプローチは、固有のアプローチといい、主として一般に市販されている標準部品を用いた並列システムの開発を目差す。もう一つ、新型というアプローチでは、大規模な並列処理システムの設計がもっぱら特注の部品に依存する。

本稿では主として固有のアプローチを考察する。一般に市販されている標準マイクロプロセッサ・システムの並列処理への応用の構想が、アーキテクチャ及び組織化の観点から詳しく論じられている。 更に、 付け足しやすい形状可変インテリジェント記憶モジュール [reconfigurable intelligent memory module]の設計も詳しく論じられている。このモジュールは、大規模な、多目的でかつ無障害の並列処理という、高く設けられた目標の実現のため不可欠である。また、固有のアプローチに内在している問題点にも触れている。これらの問題点は、市販されている製品の、並列処理への応用における困難によると思われる。これらの問題点を乗り越えるためには、新型アプローチによるシステム設計が提案される。

1. INTRODUCTION

When designing a large scale, general purpouse parallel processing system one can take two approaches:

- <u>NOVEL</u> approach: developping custom built components and using them in the overall system design (custom built processors, interconnection switches, memory modules, parallel language, operating system, software, etc.),

- <u>INHERENT</u> approach: using standard products available on the market (taking "of the shelf" microcomputers, standard software and the minimum of custom build logic to intelligently incorporate large number of processors to the parallel processing environment).

The NOVEL approach can be considered as an architectural, organizational and techological approach to the large system design. The INHERENT approach, however, only as the architectural and organizational.

Organizational approach is the cost/performance approach. Organizational concepts are generally unknown to the system programmers and do not influence the system architecture. One can think of cache as a typical organizational concept. A cache is used either to reduce the cost of memory subsystem while maintaining performance, or it is used to improve the performance of memory subsystem at constant cost. In a parallel processing environment, cost/performance components are usually fast interconnection switches, intelligent memory modules, etc.

Architectural approach is the functional enhancement for a computer system. System programmers must be aware that a given computer has, for example, virtual memory, process scheduling, etc. The cost/performance benefits are indirect, through more effective utilization of memory, processors, peripheral devices, etc.

Technological approach is concerned with a design of the components. Technological concepts are in general, either hidden or irrelevant to the system architect or system programmer.

Only close collaboration of all three classes of experts: architects, working on the organizational concepts, system programmers, working on the architectural concepts, and design engineers, working on the technological concepts can lead to a successfull general purpouse, parallel system design.

The main advantage of the INHERENT approach is, that it uses only the arhitectural and organizational concepts. A time of the overall system design is short. The system is low priced and very modular, too. The main advantage of the NOVEL approach is a fine balance of all three concepts, resulting in the improved system performance and increased system generality.

- How do the INHERENT and NOVEL approaches relate to parallel processing enironment?

Let us consider, for example, MIMD (multiple instruction, multiple data) type processing, using a general purpouse multiprocessor. The NOVEL approach conceptually leads to a tightly coupled, shared memory, easy to use, highly parallel system that can be realized in 5 to 10 years. On the other hand, the INHERENT approach leads to more relaxed, semi-tightly coupled, shared memory system, which is more rigid to use, but can be realized in short time.

Both approaches, when complementing each other, are desirable and advisable in developping well organized and well managed parallel processing system. The INHERENT approach is usefull in the first stage of the project development, so the system programmers can test software, target parallel language and algorithms. The NOVEL approach can be pursud in parallel with the INHERENT approach. Later, the components developped by the NOVEL approach can be incorporated into a system developped by the INHERENT approach.

It is important to notice, that both approaches can be separate projects, too.

This paper is the first of three papers presenting a design of a general purpouse, highly parallel processing system. We describe both approaches, however, we emphasis more the INHERENT approach. All ideas that we present for the INHERENT approach could be also applied to the NOVEL approach.

We can divide parallel processing visualization into three mutualy exclusive levels (views).

1) **MACRO** level or the way application programmer visualizes the system. This is the highest and the most abstract parallel processor visualization. Programmer is not aware of the number of processors, their interconnection, realization of process communication, scheduling, memory managment, etc. His sole concern is to get from the machine as much parallelism as possible by designing the parallel algorithm, adequately.

2) **MICRO** level or the way system programmer visualizes the system. This level enables the application programmer to visualize the machine as infinitely large and general enough to implement easy to write parallel algorithms. System programmer is not aware of the processors connection realization, memory conflict elimination or any such low level operation. Problems on this level are: process scheduling, communication, memory management, compiler and linker writing, etc.

3) **NANO** level or the way system architect visualizes the system. This level represents the actual machine configuaration. Problems on that level are: processor connections, routing, elimination of memory conflicts, fault tolerance, memory protection, etc.

This paper describes only the NANO meta level of the general purpouse highly parallel system design. Two other levels will be described in two papers that follow. In the first one to come, we will show some of the MICRO level problems: fast process scheduling, process synchronization, compiler and linker techniques. In the second one, we will show some of the MACRO level problems: parallel language constructs, technique and conceptualism of writing fast and efficient algorithms.

2. RATIONALE

2.1 philosophical motivation

Most of the people in the computer science community fallaciously classify parallel processing into two extreme categories. Ones consider parallel processing science fiction. On the other side, there are people who think of parallel processing is such a rudimentary form as two processors connected together. Consequently, they visualize parallel processing as "nothing special" or "everyone does it" task.

- Both groups are **WRONG**!!

The solution to parallel processing lies in the approach. The solution is not difficult if we take trully objective problem solving approach. This means that all preconcived notions of what can and cannot be done must be eliminated.

First group of "experts" is misled by the truism that the solution to general purpouse parallel processing is not approachable because of the following "facts":

- parallel processing is inherently special purpouse,
- increasing computing power by adding processors result in significant overhead of communication and synchronization,
- there is a technological limitation called fan-in that prevents large number of processor interconnections,
- performance of the machine is dependent of the cleverly arranged interconnection of processing elements,
- dynamic fault tolerance is impossible.

The whole philosophical idea lies in belief that those problems are approachable and most of them achievable. It is only to understand some philosophical ideas which are fundamental in understanding of the solution:

- to get maximum use of all the hardware, work must be equally distributed,
- there must be an abundant supply of usefull work,
- users should not be aware of processors or their interconnection,
- there should be no distinguished processing elements,
- faults should be detected dynamically,
- increasing computing power should not significantly increase overhead,
- interconnection should allow very large numbers of processors.

These principles are the essence of defining trully gereral purpouse, highly parallel machine. This paper will show that most of these principles are achievable, and that the proposed practical solution eliminates the truism of the first group of "experts" that parallel processing is science fiction.

What about the second group of "experts"?.

The definition of a trully parallel, general purpouse system, absolutely excludes their "nothing special" or "everyone does it" claim. Moreover, we are not aware of a commercially available system that meets all of the stated principles. If that is so, is parallel processing really so simple as they try to claim?.

The problem is in the definition. Even a system with large number of processors, (there are hundreds of such systems on the market) does not mean that a system is a parallel processing system, unless that system meets the principles and requiraments that we define.

It is, therefore, important to keep these principles in mind when reading the paper.

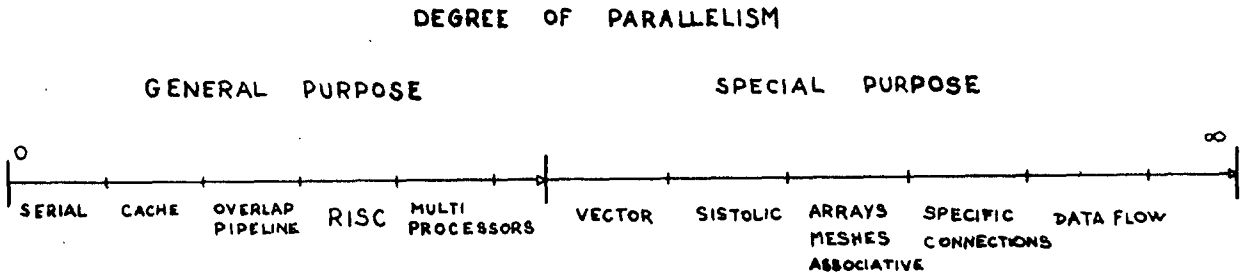


figure 2.2.1

2.2 Why multiprocessor ?

In the previos section we defined some objectives experts claim why the solution to a trully parallel processing is not achievable. The result is their failure to perceive the capability of the solution. Thus, many of the solutions have been proposed for one or few of the problems. The result has been a myriad of designs for the exploatation of parallel processing in special purpouse environment. These range from vector processors, meshes, arrays, binary trees, permutation exchange networks, systolic processors, to data flow machines.

General purpouse machines, on the other hand, improve their performance using fast caches, pipelines, silicon compilers, large number of registers, faster components. Unfortunately, they push technology to its limitatation. A huge degree of parallelism available in most of the algorithms simply eludes them. Figure 2.2.1 shows a simple architecture complexity diagram of most of todays machines, based on a degree of parallelism.

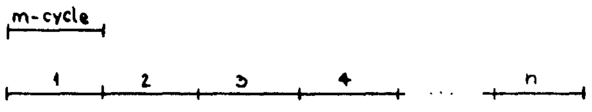
General purpouse machines do not exploit high degree of parallelism and are bounded by the current technology. Caches and pipelines are only the organizational concepts in improving a performance of the serial machines. RISC architectures are only the technological enhancements and offer no organizational or arhitectural solutions to parallel processing.

On the other side, there is an exploitation of parallelism either in very rigid form (meshes, vector processors, permutation networks, sistolic arrays, etc.), or in the highest and richest possible form (data flow).

It has been proven both theoretically, and pratically that a parallelism based only on the uniform flow of data is useless for general purpouse environment. Data flow machines, however, with their richest degree of parallelism, require enormous computational self-synchronizing overhead, and are therefore, not feasible for general purpouse environment.

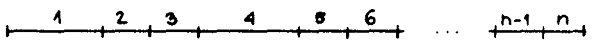
It can be very helpfull if we could illustrate a degree of parallel computation for each class of computer architecture in figure 2.2.1 using a pictorial representation of the program execution graphs.

SERIAL execution



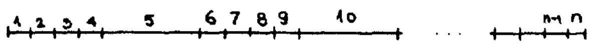
Each line represents a meta cycle. M-cycle is an abstract represention of a full instruction cycle: fetch, decode, fetch operands and execute operations. We assume that m-cycles are identical for all intructions. A serial execution of n instructions, takes n m-cycles of program execution.

SERIAL execution + CACHE



Some m-cycles shortened. The reason for it is, that some instructions and operands reside in cache, which has a shorter access time. Not all m-cycles shorten, bacause of cache miss effects.

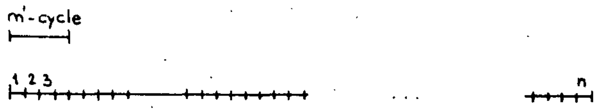
SERIAL execution + PIPELINE



A processor has a multifunctional capability:

while it is fetching instruction i, it is simultaneously decoding instruction i-1, fetching operands for instruction i-2 and executing instruction i-3. Degradations are due to jumps and interrupts in the program.

RISC execution

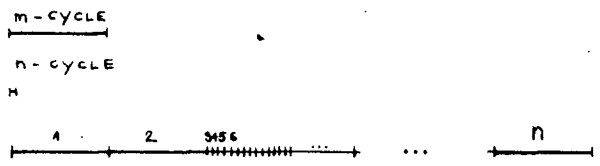


The overall computation shorten because of the following:

1) m-cycles are shorter due to higher technology (m'-cycle),

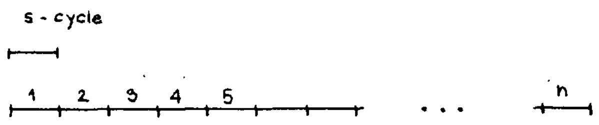
2) pipelining is optimized using large number of fast registers and silicon compilers.

VECTOR PROCESSING execution



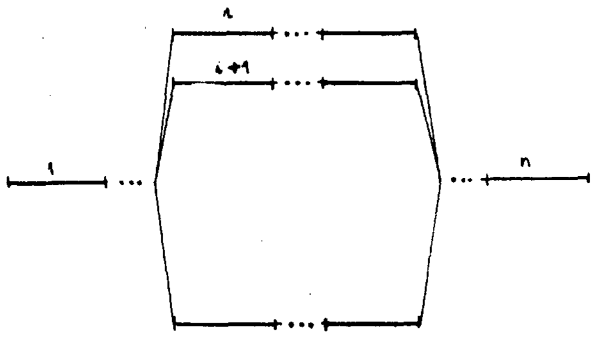
The overall computation is a combination of m-cycles and n-cycles. When a machine is in a serial mode of operation it executes with the speed of m-cycle. However, when the machine is in a vector mode, computation is faster (n-cycles).

SISTOLIC execution



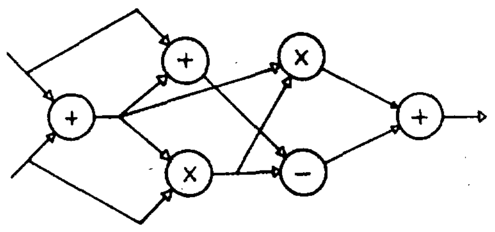
If we define s-cycle, as a cycle of each sistolic cell, and if all cells have identical operations, the speed of the overall computation is determined by the speed of the slowest cell, provided that enough problem uniformity is supplied.

MESHES, ARRAYS, ASSOCIATIVE PROCESSORS



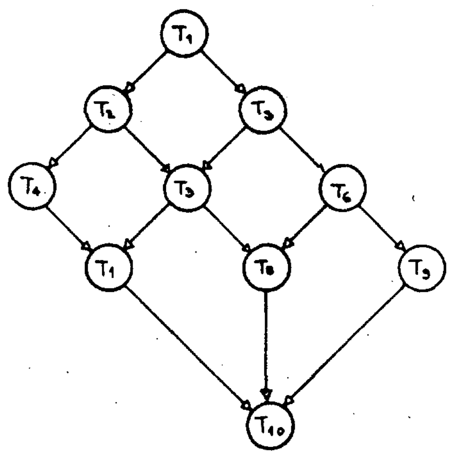
Data is distributed among the processors in such way that each processor gets specific data and works on it simultaneously with the other processors. Instructions come from the master processor and each processor executes the same instruction, but using different data set.

DATA FLOW machines



This is the richest possible parallelism. Each processor represents a program instruction. All processors can work in parallel, provided that all operands are available to each processor.

MULTIPROCESSORS



This is a combination of serial and data flow execution. Parallelism is on a level of tasks. The overall speed is determined by the successfull program decomposition into large number of independent tasks.

- Where do we put our general purpouse, highly parallel machine?

Data flow machines with the inherent parallelism have the potential to become the most prosperous arhitecture of parallel processing. However, with the enormous self-synchonizing overhead of keeping the track of all dependencies, they represent no immediate solution to the general purpouse environment.

Vector processors are doomed to die both as special or general purpouse machines.

Other special purpouse machines, such as meshes, array processors, binary trees, associative, sistolic, and others, will continue to produce optimal results only in the specific application areas.

On the other side, general purpouse machines do not exploit high degree of parallelism. RISC architecture is the typical examples of it. The speed of a RISC machine comes from the techological innovations incorporated in the component design, not from the exploatation of parallelism. As such, they represent excelent technological, but not architectural building blocks of the future parallel systems.

If we exclude data flow machines as presently unfeasable, meshes, arrays, networks, etc. as not general enough, RISC as not parallel enough, the only class of parallel processors left is: <u>**multiprocessors**</u>.

Most people express strong antagonism for such machines. They think that Cmmp, Cm*, HEP and other multiprocessor projects are good examples of how not to build large parallel systems. This statement is false, when we consider all philosophical concepts that these projects motivated. Some genuine ideas, although were not implemented, should be a foundation for any good, general purpouse parallel systems. These ideas are: shared memory, capability based addressing, possibility that each processor can access any memory location, virtual processes, processor multiplexing, etc. Unfortunately, a poor realization and the preconcieved notion that something cannot be done, resulted that these systems were abandoned. We offer some practical solutions to the problems that Cmmp, HEP and other multiprocessing projects did not and could not implement and as such have become obsolite.

3. INHERENT DESIGN APPROACH

The INHERENT design approach leads to a parallel processing system that is feasable using todays standard available hardware/software products and the minimum of the custom build logic. The goals for the INHERENT approach systems are slightly different from the goals when a parallel system is developped from the "scratch". However, these goals must be general enough and should still follow the principles stated in section 2.1.

3.1 Goals

<u>**goal 1:**</u>

tightly coupled, shared memory multiprocessor

This goal represents the most abstract visualization of the parallel processing machine; it is the way programmers want and should visualize the machine.

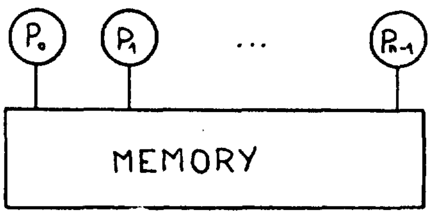


figure 3.1.1

There are "infinitely large number of processors, with infinitely large memory". System programmers and system architects should make possible that the system supports this view in a fast and efficient way.

Tightly coupled, shared memory concept has many advantages; most of them concern application programmers. Programming is easy and natural. It is not hard to design an algorithm when there are no constaints regarding the number of processors, interconnection, allocation, program decomposition, etc. Furthermore, it has been shown, that the tightly coupled multiprocessors are more effective for large scale parallel problems:

- ... taking all in all, we come to the optimistic conclusion that effective use of extremely large parallel computing is possible ... [when] N identical processors share a common memory ... [Schw80],

- ... vector and array processors were designed of solving fluid type problems efficiently ... in general, these machines do not lend themselves to particle tracking ... [Rod80]

Let us illustrate the way an application programmer could conceptualize a problem on a multiprocessing machine. This conceptualization is a direct implication of goal 1. An example is taken from the companion paper[Bra86], where we give techniques and conceptions on how to program large scale tightly coupled multiprocessors efficiently.

In the following sorting algorithm it is assumed that a programmer is clever enough to think of an algorithm that will take advantage of the *machine* by *visualizing* it as an *infinite* resource of processors and memory space.

Let us assume that we want to sort n numbers: x = 5 4 8 1 10 3 7 2. We will use nxn matrix in the following way:

1) $a_{i,j} = \begin{cases} 1 & \text{if } x_i > x_j \ \underline{\textbf{for all}}\ i,j \\ 0 & \text{otherwise} \end{cases}$

2) $a_{i,j} = 1 + a_{1,j} + a_{2,j} + \dots a_{n,j}$

3) $y_i = x_j\ \underline{\textbf{such}}\ \underline{\textbf{that}}\ a_{i,j} = i$

$$\begin{array}{c|cccccccc} & 5 & 4 & 8 & 1 & 10 & 3 & 7 & 2 \\ \hline 5 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 4 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 8 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 7 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 2 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \end{array}$$

$$1 + \sum = \ 5\ \ 4\ \ 7\ \ 1\ \ 8\ \ 3\ \ 6\ \ 2$$

The best serial algorithm for sorting involves $O(n\log_2 n)$ operations. It is easy to see that for this algorithm, N^2 operations are independent and can be performmed in parallel.

The time necessary for step 1 and 3 is O(1). The summation of the comparison results in step 2 can be *performed* in $O(\log_2 n)$. Thus, for the application programmer, this algorithm is of $O(\log_2 n)$ time complexity. The whole algorithm is given in figure 3.1.2.

This program is written in C*, standard UNIX C plus some parallel primitives that we consider sufficient for effective parallel programming. New parallel programming primitives are distinguished in bold.

There are two steps in the program worth mentioning:

step 0: fibonacci_create_process

This is a dynamic creation of N^2 processes in $O(\log_4 n*\log_2 n-1)$ time. This is one of the techniques in decreasing the overall time of the algorithm. User could easily write a double loop to generate N^2 processes. The difference in time is obvious.

step 2: collect_result

Element summation can be done $\wp(\log_2 n)$ time. Here, we also suggest fibonacci collection technique.

Any programmer with the trainned concept of parallel processing, and an idea of n "things" occuring simultaneously, can design easy to write algorithms, using parallel primitives added to the standard programming language. These primitives are: **process**, **send**, **receive**, **create process**, **shared**, **signal**, **abort**.

```
#define N 11
shared int x[N],y[N];
signal int a[N][N];

process parallel_sort ()
  {
    main()
     {
       /*  read n integers into vector x  */

       create process sort(1,1);

       /*  write n sorted integers from y */
     }
  }


process sort()
  {
    main(i,j)
     int i,j;
     {
        int result;
step 0: fibonacci_create_processes(i,j);
step 1: if (x[i] < x[j]) result = 0
           else result = 1;
step 2: collect_result(result,i,j);
     }


    fibonacci_create_processes(i,j)
     int i,j;
     {
      if (j=1) then
        {
          if (2*i < n)
                create process sort(2*i,1);
          if (2*i+1 < n)
                create process sort(2*i+1,1);
        }
      if (2*j < n)
                create process sort(i,2*j);
      if (2*j+1 < n)
                create process sort(i,2*j+1);
     }

    collect_result(res,i,j)
     int res,i,j;
     {
      int part_sum_1,part_sum_2,part_sum;
      if (2*i <= n)
         receive (part_sum_1) from(a[2*i,j]);
      if (2*i+1 <= n)
         receive (part_sum_2) from(a[2*i+1,j])
      part_sum = part_sum_1 + part_sum_2 + res
      if ( i=1 )
step 3: y[part_sum+1] =  x[j]
           else send (part_sum) to (a[i,j]);
     }
  }
```

figure 3.1.2

goal 2:

N processors are connected to N memory modules via shared memory controller

This goal is an abstract view of a system programmer visualization of a parallel machine. Our goal is to desing a memory controller in such way that will enable N^N processor-memory connections. N^N connections are possible using generalized connection networks (GCN). A GCN is a switching network with N inputs and N outputs capable of implementing any mapping of inputs onto outputs. Our goal is to design a shared memory contoller as an intelligent GCN. A system programmer must be aware of the functions of the shared memory contoller; however, the implementation of these functions should be in general hidden to him.

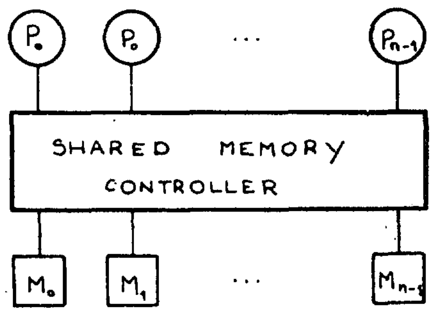


figure 3.1.3

goal 3:

expensive and centralized shared memory controller in figure 3.1.3 must be transformed into N simpler and decentralized modules

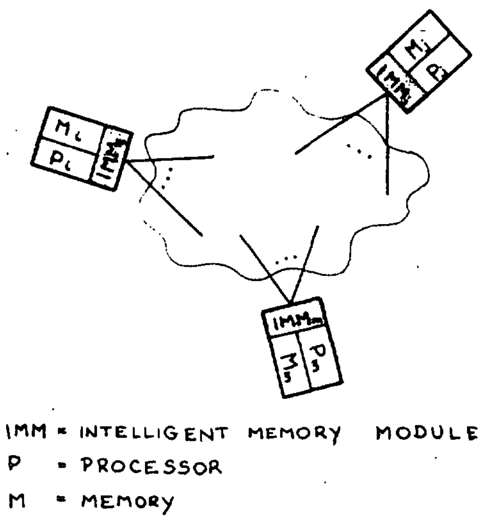


figure 3.1.4

This goal is a typical example of the INHERENT approach system design. The objectives to this goal are:

- processor memory connection on most of the standard microprocessors is fixed,
- GCNs are very expensive,
- GCNs are not modular (it is hard to add new processors and memory banks).

Our goal is to transform a shared memory controller into N smaller and modular intelligent memory modules, such that the properties of the intelligent GCNs remain. Furthermore, the intelligent memory modules should be added easily to any standard available computer.

This view is the way system architect visualizes the system. It is his task to design an intelligent memory module that will enable system programmer to visualize a machine as in figure 3.1.3.

goal 4:

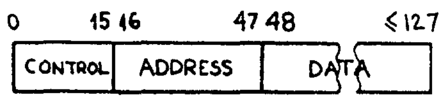
a system in figure 3.1.4 should have shared memory concept

Figure 3.1.4 shows each processor having its local memory. However, it is our goal that a parallel processing machine behaves as a tightly coupled, shared memory system as in figure 3.1.1.

goal 5:

system should support packet switching memory accessing

Each memory access should result in a packet generation of the following form:



Control bits contain information regarding packet routing, access modes, fault tolerance. Address and data bits are the same as for the conventional memory accesses.

goal 6:

intelligent memory modules should route the packets

It is important to free processors from uneccessary system work. Packet routing is one of the system operations that can be done by the intelligent memory controller. The advantage is: routing is faster, simpler and hidden from the programmer.

goal 7:

routing is deterministic

This goal is not quite clear at this point. Readers will see how the deterministic routing can be used in memory contention elimination.

Deterministic routing means that a memory request packet from processor i to memory j travels the same path as the memory response packet from memory j to processor i.

goal 8:

intelligent memory modules should detect failures and allow bypass

In a real life parallel processing, it is to expect that an intelligent memory module fails. Other intelligent memory modules should dynamically detect a failure and take action in bypassing the failed intelligent memory module.

All other goals such as shared global memory, deterministic routing, etc. should not be effected by this system dynamic adaptivity to the intelligent memory module failure.

goal 9:

intelligent memory module should eliminate problems concerning:

- **memory contention**
- **slow process scheduling**
- **slow process synchronization**

This is one of the most important and hardest goals, and therefore requires some additional explanations.

a) memory contention

There are two types of memory contention:

- two or more processors access the same memory location, simultaneously,

- two or more processor access different memory locations, in the same memory bank, simultaneously.

Memory contention has been the largest problem for the general purpouse multiprocessors. Let's see what the designers of the C.mmp multiprocessor wrote about memory contention problem:

> ... if a number of Cms repeatedly access the same memory, as might happen if the memory contains a variable shared between several processes, **severe degradation results**. ... it is necessary to avoid decomposition which requires a single shared variable to be accessed repeatedly... **memory access patterns must be included** in a usefull model of parallel computation...[Ost83]

These statements are hard acquisitions of goal 1, which states that a programmer is free from programming and organizing data in a sprecific and deterministic form.

Therefore, it is our task to prove that:

- C.mmp designers were wrong,

- if a number of processors access the same memory location, sevare degradation **does not** result,

- the memory access pattern should be hidden from the programmer.

We show in this paper, not only that the memory conflicts can be eliminated, but that they can be quite usefull, too.

b) scheduling cost

In a general purpouse machine from figure 3.1.1, very often can occur that there are more processes than processors. It is nessesary to alter, dynamically, which process is being executed by which processor. This involves computational overhead because the internal state of a process (local variables, PC, registers) need to be saved, another process then selected for execution and the internal state of that process loaded into the processor.

We will show some practical solutions to this problem that involves both an intelligent approach: using ideas from parallel queue access by Cache and Add (practical realization of Fetch and Add[Got83],[Sto85]), and brute force: using intelligent memory module as a backround processor for saving and resaving internal states. An intelligent memory module allows a processor to do usefull work, while the intelligent memory module is responsible for preparing the internal state for possible process rescheduling.

These ideas will not be presented in this paper; they belong to the MICRO level of the parallel processor complexity design, and will therefore, appear in some of the future papers.

c) synchronization cost

This cost is very similar to the scheduling cost, except that the application programmer is aware of it, if not responsible for it by issuing synchronization primitives such as **send** and **receive**.

Synchronization cost results when:

- a process issues **receive** call and the desired information is not present. Rescheduling takes place.

- process issues **send** call and the "mailbox" is full. Rescheduling takes place.

A programmer is very responsible for the synchronization cost. He must design an algorithm in such a way to require as little synchronization primitives as possible.

A system architect or system programmer can do very little to eliminate the synchronization cost. If the algorithm is designed in such a way that it does not exploit parallelism, or has "trown in" unnecessary sends and receives, even the "Paracomputer"[Sch80] would not take the advantage of its infinitely large conflict free machine.

An assumption that a programmer "deliberately" serializes the algorithm is contrary to the seconf principle in section 2.1 which says that there must be an abundant supply of usefull work. Therefore, as the system designers we must follow the principles, not the bad examples of real computation, and therefore, treat synchronization cost nothing more than as the scheduling cost.

goal 10:

intelligent memory module should incorporate all good ideas from other projects

We consider parallel processor system design both a research task (trying to develop something new), and an industry oriented product. As an industry oriented product, parallel processor should, therefore, "steal" all good ideas from other similar projects. Two of such ideas are:

- "Conflict Free Memory" from CHoPP project[Sul83]

- "Fetch and Add" from IBM RP3 project[Got83]

a) Conflict Free Memory

Conflict Free Memory is a philosophical, not the implementational feature of the parallel processing system design. The realization of the Conflict Free Memory could differ from project to project. In sections 5.2 and 5.5 we give a detailed Conflict Free Memory realization for the proposed parallel processor.

Conflict Free Memory has the following features.

- All processors may read the same location simultaneously.

- All processors may write to the same location simultaneously, however, only one processor updates the memory. It is programmer's responsibility to impose serialization for operations that require simultaneous write accesses. Hardware should be free from such tasks.

Let us first consider two simultaneous reads to the same memory location. Let us also visualize the system as in figure 3.1.3.

Shared memory controller intercepts both accesses, and finds out that they request the same memory location.

Shared memory contoller:

1) stores and forwards the first access,
2) stores but does not forward the second access.

Only one access reaches the memory bank.

On the way back, shared memory controller intercepts the requested result, finds out that both processors requested the result and therefore, sends the result to both of them.

The procedure is the same for the simultaneous writes to the same memory location. N-1 requests are stored, only the first request is forwarded to the memory.

Other possible combinations, such as simultaneous writes and reads are also covered in sections 5.2 and 5.5, where we give a detailed design of what we call the Conflict Filter, a key feature of the intelligent memory module.

b) Fetch and Add

Fetch and Add is a synchronizational mechanism used in NYU Ultra and IBM RP3 projects, and can be described as the most promissing synchronizational mechanism which permits highly parallel execution of operating system primitives.

The format of Fetch and Add is F&A(V,e), where V is a variable, and e is an expression. If V is a shared variable and many F&A's simultaneously address V, the effect of this operation is exactly the same as it would be if the access occured in some serial order. The semantics of F&A is:

Assume that processor P_i executes ANS_i = F&A(V,e_i), and that simultaneously processor P_j executes ANS_j = F&A(V,e_j).

Then

$ANS_i = V$ and $ANS_j = V+e_i$

or

$ANS_i = V+e_j$ and $ANS_j = V$,

and , in either case the value of V is $V+e_i+e_j$.

Let us consider two simultaneous Fetch and Add operation using figure 3.1.3.

Shared memory controller intercepts both F&A's and finds out that they require the same shared variable V.

Shared memory controller:

1) stores both accesses,
2) forwards F&A(V,e_i+e_j).

Let us assume that the value of variable V was R. On the way back, shared memory controller returns R to the processor that requested F&A(V,e_i), and $R+e_i$ to another processor. New value of V is $R+e_i+e_j$.

An example of an airline ticket reservation can illustrate the usefullness of the Fetch and Add operation. Let us assume that 5 processors simultaneously request F&A(tickets,1). Initial value of variable tickets is 43. Shared memory controller intercepts the requests, and forwards F&A(tickets,5) to the memory bank. Variable tickets is incremented to 48. On the way back, each processor gets a differant, but correct update of the variable tickets.

In sections 5.2, 5.5, and 8.4 we give details of Cache and Add operation, a real world implementation of Fetch and Add, using intelligent memory modules.

__goal__ 11:

the ULTIMATE goal: profit

The intelligent memory module can be used as the important building block of a large parallel processor. Therefore, we can use it as a part of a multiprocessor that we develop ourselves, and that can be used in various areas of applications: numerical calculations, data processing, artificial intelligence, vision and image processing, computer graphics and CAD, etc.

There is another very important profitable aspect of the intelligent memory module. It can be used stand-alone by other projects involved in developping parallel processors. An intelligent memory module is general and modular enough, and can be used as an add-on unit for any parallel processing system using the INHERENT design approach.

Therefore, the intelligent memory module can be sold as a separate unit to anyone interested in developping large scale parallel processor system, or can be a part of our own multiprocessing system that can be sold to various application oriented users.

4. INTERCONNECTION

In this section we implement goals 1,2,3,4,5,6 and 7.

We start with an abstarct system in figure 3.1.3 that has N processors connected with N memories using a shared memory controller. This shared memory controller is an intelligent generalized network connector (GCN).

4.1 deterministic routing

It is clear by now that concepts such as Fetch and Add and Conflict Free Memory require that packets travel the same path to and from memories.

Therefore, we must develop a mapping that supports N^N connections, and has the routing deterministic property.

In a field of cryptography, exclusive or (__EOR__) function is used for simple coding and decoding.

__Definiton:__

Exclusive OR (__EOR__) has an unique relation:

$$s \text{ EOR } d = r \quad \text{and} \quad r \text{ EOR } d = s$$

or

$$s \text{ EOR } d \text{ EOR } d \text{ EOR } = s.$$

Exclusive OR is a function that uniquely and deterministically relates r,s and d.

This notion we incorporate into our routing procedure. A shared memory controller uses __EOR__ when forwarding a packet from P_i to M_j.

Let's assume:

s = source address (processor number)
d = destination address (memory bank number)

__Definition:__

Routing Tag r, is a function result of s __EOR__ d, which tells to the shared memory controller how to route packets from processor s to memory bank d.

Let $s_0 \dots s_{n-1}$, $d_0 \dots d_{n-1}$ and $r_0 \dots r_{n-1}$ be the binary representations of r,s and d.

If there exists a path between s and d, then by the definition of __EOR__:

$s_{n-1} \dots s_0$ __EOR__ $d_{n-1} \dots d_0 = r_{n-1} \dots r_0$,

and r gives a unique path traversal between s and d.

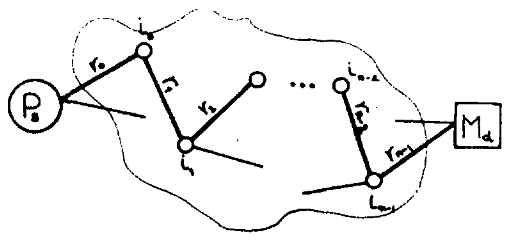


figure 4.1.1

Since r uniquely relates s and d, then a path from P_s to M_d, and a path from M_d to P_s are the same, where:

forward path is: $r_0 r_1 r_2 \dots r_{n-1}$,

backward path is: $r_{n-1} r_{n-2} \dots r_1 r_0$.

Each r_i can have two states: 0 and 1. Therefore, a contact point (represented by i's in figure 4.1.1) inside a shared memory controller must have two incoming and two outgoing links.

If a packet comes to a contact point j, it will be forwarded to one of the outgoing links. Shared memory controller decides which link to choose and the decision is based on value of r_j.

Theorem 1:

An unique path between each processor and each memory is possible through $\log_2+1$ stages of contact points, such that each contact point (switch) at level j with a binary tuple representation $(b_{n-1}b_{n-2}\ldots b_j\ldots b_0, j)$ is connected to switches $(b_{n-1}b_{n-2}\ldots c_j\ldots b_0, j+1)$ and $(b_{n-1}b_{n-2}\ldots b_j\ldots b_0, j+1)$ at level $j+1$, where c stands for bit complement.

proof:

A proof is based on induction of coupling together two processor sets, each one having 2^i processors into one set of 2^{i+1} processors.

i) simple case for $n=2^0$

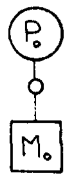


figure 4.1.2

Routing is deterministic and unique, since there exists a single processor memory connection.

ii) simple case for $n=2^1$

Let's assume that there are two independent processors from figure 4.1.2 that we want to connect. If we reenumerate them, add a new level and apply Theorem 1, so that each switch at level 0 is connected to a switch at level 1:

- switch 0 is connected to switches 0 and 1
- switch 1 is connected to switches 1 and 0

the following connections result:

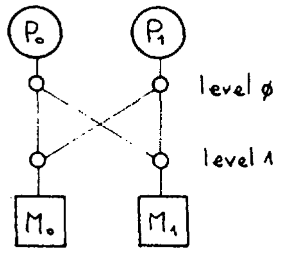


figure 4.1.4

Routing is unique and deterministic.

If P_0 wants M_1, routing tag is: 0 **EOR** 1 = 1; reverse routing is the same.

If P_1 wants M_0, routing tag is: 1 **EOR** 0 = 1; reverse routing is the same.

iii) simple case for $n=2^2$ (illustration only)

Let's assume that there are two independent sets of processors from figure 4.1.4 that we want to connect. If we reenumerate them, add a new level, and apply Theorem 1:

- switch 00 is connected to switches 10 and 00
- switch 01 is connected to switches 11 and 01
- switch 10 is connected to switches 00 and 10
- switch 11 is connected to switches 01 and 11

the following connections result:

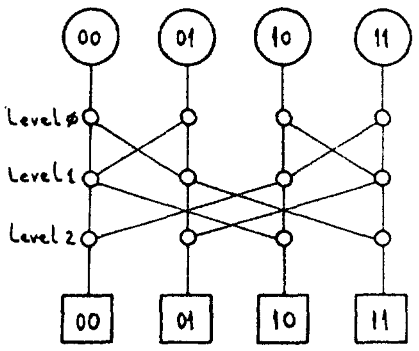


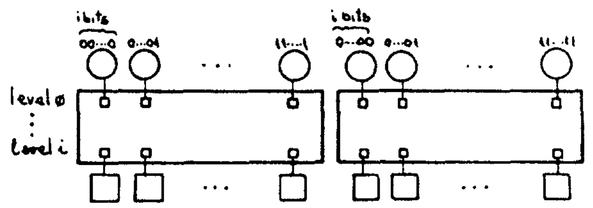
figure 4.1.6

Routing is unique and deterministic.

If processor P_{01} wants to access memory bank M_{11}, a shared memory controller generates a routing tag: 01 **EOR** 11 = 10. The rightmost bit represents a transition from level 0 to level 1; leftmost bit represents a transition from level 1 to level 2. We showed in ii) that the rightmost bit transition is unique and deterministic; here, we show that the leftmost bit transition is unique and deterministic, too.

iv) induction case for $n=2^{i+1}$

Let's assume that there are two independent sets of 2^i processors, for which Theorem 1 works. Or in other words, routing is unique and deterministic for two sets of $n=2^j$ processors.



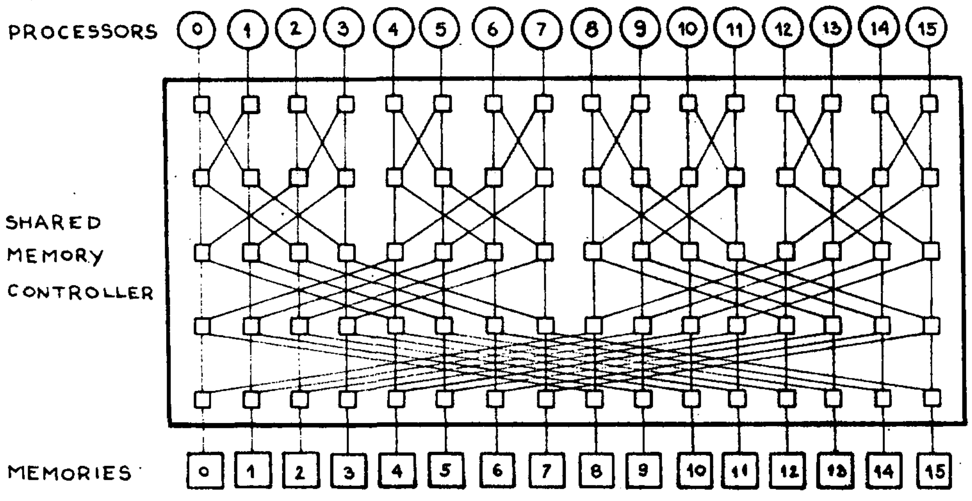


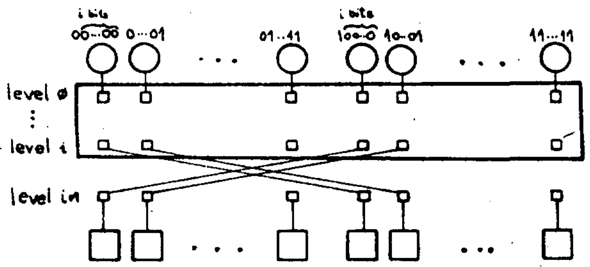
figure 4.1.9

We see that the switches of the first group have 0 as the leftmost bit, and that the switches of the second group have 1 as the leftmost bit.

Let's apply Theorem 1: each switch at level i is connected to a switch at level i+1 as following:

- switch $0x_{i-1}...x_0$ is connected to switches $1y_{i-1}...y_0$ and $0x_{i-1}...x_0$,
- switch $1x_{i-1}...x_0$ is connected to switches $0y_{i-1}...y_0$ and $1x_{i-1}...x_0$

where $x_i = y_i$ for all i.



If we assume that the routing from stage 0 to i is unique and deterministic, than it is obvious, from iii), ii) and i) that the routing from stage i to stage i+1 (leftmost bit) is deterministic and unique, too.

QED.

In figure 4.1.9 we show the graphical representation of the shared memory controller for $n=2^4$ processors.

Let's consider a simple example: P_{10} wants to send a packet to memory bank M_6. A shared memory controller generates a routing tag:

1010 **EOR** 0110 = 1100

Reading the routing tag from right to left:

$S_{10,0}$ sends a packet to $S_{10,1}$ (first bit 0),
$S_{10,1}$ sends a packet to $S_{10,2}$ (second bit 0),
$S_{10,2}$ sends a packet to $S_{14,3}$ (third bit 1),
$S_{14,3}$ sends a packet to $S_{6,4}$ (forth bit 1).

4.2 Interconnection transformation

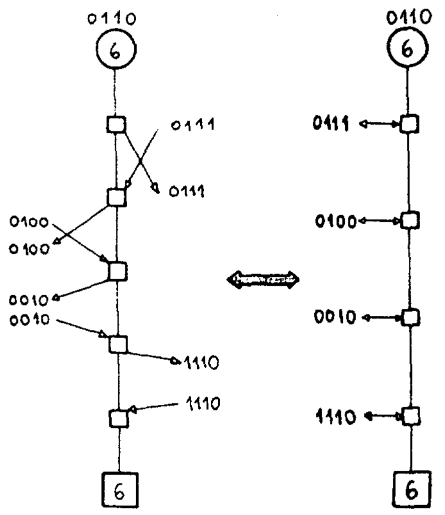
In figure 4.1.9 we see that a proposed shared memory controller consists of 80 switches ($n\log_2 n+n$), and 128 ($2n\log_2 n$) lines crossing each other in a complicated manner. Obviously, such a structure is unacceptable and too expensive for practical VLSI implementation.

__Theorem 2:__

Shared memory controller can be transformed into N independent, smaller and modular **intelligent memory modules.**

proof:

Let's isolate any processor in figure 4.1.9.

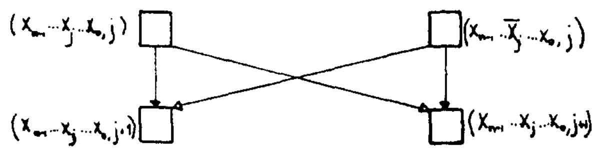


From this picture we see that:

- $switch_{i,j}$ has an outgoing link to $switch_{i+1,k}$
- $switch_{i+1,j}$ has an incoming link from $switch_{i,k}$

where $switch_{r,q}$ represents a switch at stage r, of processor q.

This oservation is an consequence of Theorem 1.


$(x_{m}\cdots x_j \cdots x_0, j)$
$(x_{m}\cdots \overline{x}_j \cdots x_0, j)$
$(x_{m}\cdots x_j \cdots x_0, j+1)$
$(x_{m}\cdots x_j \cdots x_0, j+1)$

Therefore, an outgoing and incomming links are the same.

QED.

Hence, the shared memory controller is tranformed into N module. Each module can be an add-on element to a standard available microprocessor.

collorary 1:

Transformed interconnection structure is equivalent to HyperCube.

proof:

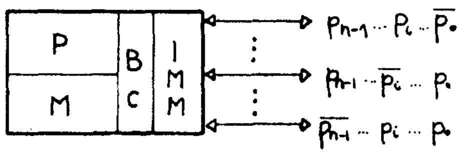
Let $P=p_{m-1}\dots p_1 p_0$ be the binary representation of an arbitrary line label.

HypeCube interconnection functions can be defined as:

$cube_i(p_{m-1}\dots p_0) = p_{m-1}\dots p_{i+1}, c_i, p_{i-1}\dots p_0$

where $0 <= i <= m$, $0 <= P <= N$, and c_i represents bit complement. Or in other words, each processor P is connected to $\log_2 N$ processors, such that these processors differ only in one bit of their binary representation.

From Theorem 1, Theorem 2 and a definition of HyperCube, it is obvious that the transformed inteconnection structure represents HypeCube.


P
M
B
C
I
M
M
$p_{n-1}\cdots p_i \cdots \overline{p_0}$
$p_{n-1}\cdots \overline{p_i} \cdots p_0$
$\overline{p_{n-1}}\cdots p_i \cdots p_0$

QED.

Figure 4.2.1 shows the way 16 processors could be connected to a parallel processor system using HyperCube connection, and the intelligent memory modules as the connection mechanism. Figure a) represents linear, brute force connections; figure b) represents the optimized Shared HyperCube connections. Note that in figure b) no lines cross each other when 8 processors are put atop of other 8 processors.

In this paper, we use a structure that is more figurative for understanding system functions, algorithms, etc. A reader should keep in mind that the structures from figures 4.1.9 and 4.2.1 are the same, and represent only a different system visualizations, not the different structures.

4.3 interconnection complexity

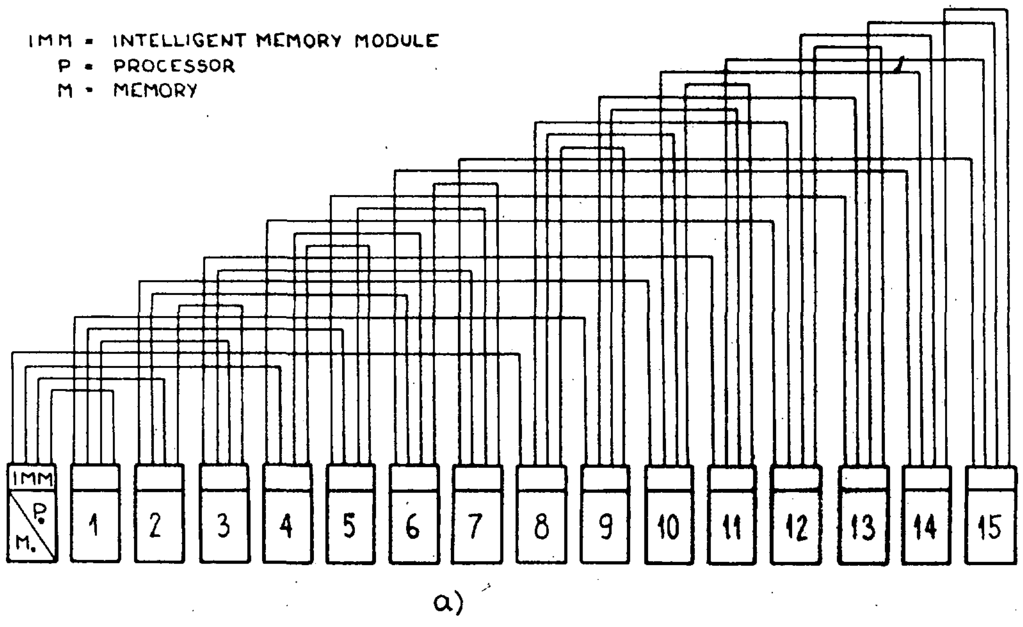
In sections 4.1 and 4.2 we showed an interesting transformation in converting a very complicated shared memory controller into N small intelligent memory module, in such a way that these intelligent memory module connect each other using HyperCube structure. There are at least three advantages of the proposed interconnection scheme compared to regular HyperCube.

1) This interconnection scheme is more powerfull, because it supports shared memory concepts (HyperCube is typical local memory parallel processing interconnection scheme).

2) It is faster than HyperCube, because routing is done by the intelligent memory module and not the processors.

3) We can use "of the shelf" processors and add to them the intelligent memory modules to design the parallel processing machine.

In comparison to other interconnection schemes, our proposed interconnection network of N intelligent memory modules is as cheap and as fast as HyperCube, and therfore, belongs to a



class of networks that are considered to be the most prosperous interconnection networks for large scale parallel processors such as: omega, baseline, indirect binary n-cube, benes, shuffle, etc.

4.3.1 time complexity

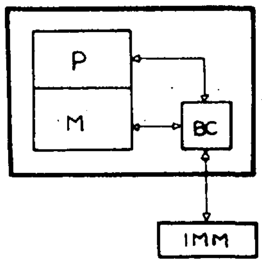
Adapting the intelligent memory module to a standard microprocessor in most examples results in the intelligent memory module connection to the microprocessor's bus controller.

Ones may argue that this connection results in a loosly coupled parallel system.

It is true that the time is increased when a bus controller is used in processor, intelligent memory module connection. We will show that the overall does not increase so much to call a whole design loosly coupled.

Let us show that the microprocessor's bus controller does not influance the memory access time delay.

In general, a processor uses a direct connection to its local memory; only when reffering to the peripheral devices, processor uses the bus controller. When a processor requires a memory location that is not in it's local domain, it uses a bus controller and through the intelligent memory modules, processor accesses a global information.



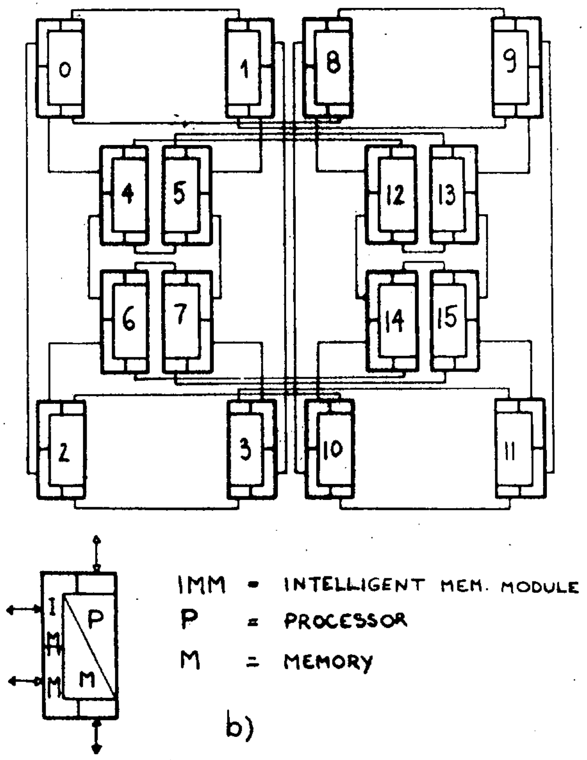
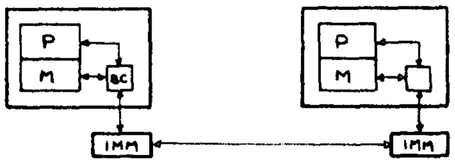


figure 4.2.1

A claim that a communication using a bus contoller is too slow and that the time overhead is too large compared to a direct processor memory connection is valid only if the number of processors is very small.

Let's consider the following situation:



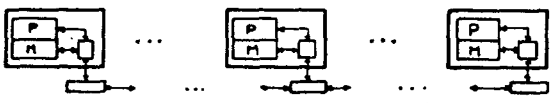
t_m - memory acces time
t_{BC} - bus controller access time
t_{IMM} - intelligent memory module time
t_{OH} - overhead time

Overall acces time $T = t_m + t_{OH}$, where $t_{OH} = 2(t_{BC} + t_{IMM})$.

A bus contoller acces time is the most dominan factor in determining the overall speed. Therefore, such a system can be considered loosly coupled.

Let's consider a system with large number of processors $(n >> 2)$.

We have the following situation:



Again, $T = t_m + t_{OH}$.

However, $t_{OH} = 2(t_{BC} + t_r) + t_q$

where $t_r = t_{IMM}*\log_2 n$ represents routing delay, adn t_q queuing delay due to interconnection congestion.

We see that t_{BC} is too small compared to t_r to be considered a dominant factor in the overall speed, and therefore, a system cannot be called loosly coupled.

What about t_r and t_q ?

A routing time t_r has been shown[Bra86] as the best price/performance time for any large scale parallel processing system. Only two interconnection schemes have better performance than t_r: crossbar switch, and fully connected network. However, both scheme are so expensive that they are unfeasable for large number of processors. Other schemes: buses, rings, stars, etc. which are cheaper, are not powerfull enough, and should be avoided for large n.

We have not measured the queuing time delay t_q, yet. Future studies and simulation runs will tell us more about it. However, we are very optimistic about t_q. At this moment, we can only speculate that using conlict filters (sections 5.2), random interleaving (section 8.3), cache and add (section 5.3), we distribute enough memory accesses that no bottlenecks create on any intelligent memory module or memory bank.

4.3.2 space complexity

In order to connect N processors, we require N intelligent memory modules. Each intelligent memory module is connected to $\log_2 N$ other equivalent modules.

Hence, there are $N/2*\log_2 N$ connections. For a 16 processor system, assuming that each connection is 16 wires wide + 4 wires for synchronization, we have 640 wires: $(m+s)n/2\log_2 n$, where m represents number of lines per connection, and s a number of lines for synchronization.

5. INTELLIGENT MEMORY MODULE

In the previous sections of the paper we defined an intelligent memory module as the main building block of the multiprocessor system.

It would be usefull to summarize again the functions that we expect that the intelligent memory moduel should do in order:

1) to increase the speed of the system,

2) to free processors from doing unessesary work,

3) to increase the modularity of the system.

There are numerous functions we should consider; some new ones we expect to appear in the course of an actual machine design. To name the few:

i) packet routing
ii) elimination of memory conflicts
iii) detecting and solving failures
iv) address translation
v) packet creation
vi) memory interleaving
vii) implementation of Cache and Add
viii) shared memory access implementation
ix) routing tag generation
x) fault tag table maintainance
xi) protection
xii) process scheduling
xiii) process synchronization

In this paper we will show how to implement first ten functions; they are part of the NANO level, and as such, they represent problems for the system arhitects.

It is important that some of the functions of the intelligent memory module take place simultaneously.

Therefore, we divide the intelligent memory module into two blocks:

1) routing units

2) global memory unit

There are $\log_2 N$ routing units in each intelligent memory module. They are responsible for packet routing, memory conflicts and fault tolerance. Global memory unit is concerned with packet creation, address translation, memory interleaving, implementation of Cache and Add, and access to the global memory.

5.1 routing units

Let's assume that we have a system with $N=2^n$ processors. Each intelligent memory module has n routing units.

Let $m_j = m_{n-1}m_{n-2}\ldots m_0$ be an intelligent memory module number of processor j.

Then, each routing unit i (RU_i) is connected to three other routing units:

1) if i <> 0 then RU_{i-1}
   else global memory unit

2) if i <> n-1 then RU_{i+1}
   else global memory unit

3) RU_i of the intelligent memory module $m_{n-1}\ldots c_i\ldots m_0$, where c stands for bit complement

Each RU_i routes a packet:

1) to RU_{i+1}, if i-th bit of the routing tag is 0 and a packet represents memory request,

2) to RU_{i-1}, if i-th bit of the routing tag is 0 and a packet represents memory reply,

3) to RU_i of an intelligent memory module $m_{n-1}\ldots c_i\ldots m_0$, if i-th bit of the routing tag is 1.

4) to global memory unit, if i=n-1 and i-th bit is 0 and a packet represents memory request,

5) to global memory unit, if i=0, i-th bit of the routing tag is 0 and a packet represent memory reply.

Figure 5.1.1 shows the block diagram of the intelligent memory module $m_3m_2m_1m_0$, with the corresponding connections to other intelligent memory modules and its local bus controller.

For example, if $r_3r_2r_1r_0$ represent a routing tag, then, if a packet is in RU_1, then RU_1 will send the packet:

1) to RU_2 if r_1=0 and packet=forward

2) to RU_0 if r_1=0 and packet=backwards

3) to RU_1 of $m_3m_2c_1m_0$ if r_1=1

5.2 conflict filter

In section 3.1, goal 8, we described the function of the Conflict Free Memory.

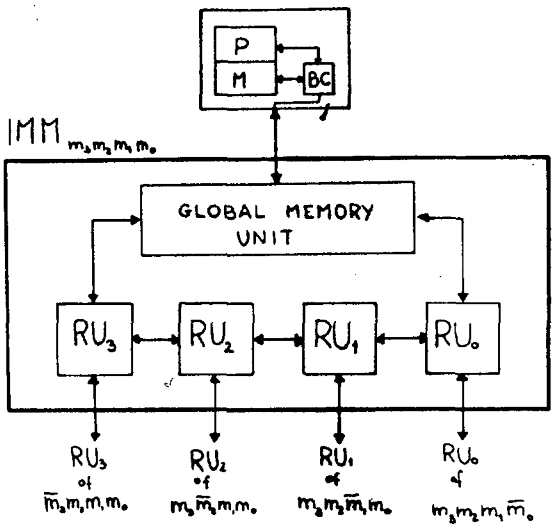

P
M
BC
IMM $m_3m_2m_1m_0$
GLOBAL MEMORY UNIT
RU_3
RU_2
RU_1
RU_0
RU_3 of $\bar{m}_3m_2m_1m_0$
RU_2 of $m_3\bar{m}_2m_1m_0$
RU_1 of $m_3m_2\bar{m}_1m_0$
RU_0 of $m_3m_2m_1\bar{m}_0$

figure 5.1.1

If we consider the interconnection structure as in figure 4.1.9 we can notice two properties of the structure, that are important for the Conflict Filter implementation.

1) Memory requests and memory replies travel the same path. This is a consequence of Theorem 1.

2) On the way to the same memory location, two packets meet each other in on of the switches.

Let's assume that processors P_0, P_2, P_5, and P_7 access the same location in M_6, as it is shown in figure 5.2.1.

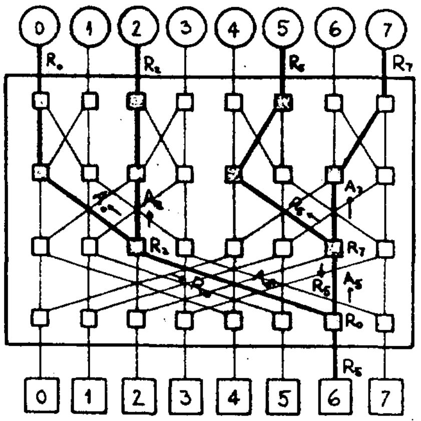


figure 5.2.1

Let's call the corresponding requests: R_0, R_2, R_5, R_7.

Then,

1) R_0 and R_2 meet at switch $S_{2,2}$; if R_0 cames first, $S_{2,2}$ sends R_0, and stores R_2 in its local buffer. Otherwise, R_0 is stored and R_2 is sent.

2) R_5 and R_7 meet at switch $S_{6,2}$; R_5 is sent, P_7 is stored.

3) R_0 and R_5 meet at switch $S_{6,3}$. R_5 is sent, R_0 is stored.

4) R_5 accesses memory, alone.

No queueing at memory bank M_6, although 4 accesses took place, simultaneously.

On the way back,

1) answer A_5 arrives at $S_{6,3}$. $S_{6,3}$ sends back both A_5 and A_0.

2) answer A_0 arrives at $S_{2,2}$. $S_{2,2}$ sends back A_0 and A_2.

3) answer A_5 arrives at $S_{6,2}$. $S_{6,2}$ sends back A_5 and A_7.

Let's consider the following scenario:

P_5 sends a packet to M_1. When A_5 is on the way back, P_4 sends a packet to the same memory location as P_5. Both requests meet at $S_{4,2}$. At that time A_5 arrives at S_4, and ignites A_4 and A_5. Processor P_4 receives a packet in much shorter time than it would be if the packet needed to go through the whole network.

This is why we consider memory conflicts sometimes <u>usefull</u>.

This is really science fiction for C.mmp, Cm*, and other multiprocessor projects.

What are the hardware requirements for the Conflict Filter?

Each routing unit should consists of:

1) logic for packet composition/decomposition,

2) queueing synchronization,

3) logic for determining routing action,

4) <u>CAM</u> - content addressable memory for implementing Conflict Filter.

5.3 content addressable memory

Content addressable memory is a memory unit addressable by content. This type of memory is accessed simultaneously and in parallel on the basis of data content rather than by specific address or location. Because of its organization, the associative memory is uniquely suited to do parallel searches by data association. Moreover, searches can be done on an entire word or on a specific field within the word.

The block diagram of a CAM is shown in figure 5.3.1.

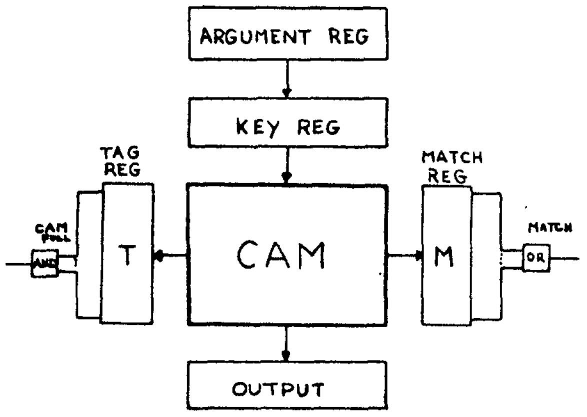


figure 5.3.1

Each word in memory is compared in parallel with the content of the argument register. The words that match the bits of the argument register set a corresponding bit in the match register.

Reading is accomplished by sequential access to memory for those words whose corresponding bits in the match register have been set.

A word is deleted from memory by clearing the bit tag.

Words are stored in memory by scanning the tag register until the first 0 bit is encountered.

In the interconnection scheme from figure 4.1.9 and the example in section 5.2, we see that only two accesses to the same location can meet in a particular RU_i. That simplifies CAM read operation.

When a packet arrives at RU_i, RU_i associatively compares the address with the other entries in the CAM. If the match occures, packet is not sent. Since two packets differ in their routing tags, only control bits are stored in CAM in a location on which the match occured.

If the match did not occur, both the address and the control bits are stored in CAM.

When a return packet arrives at RU_i, RU_i compares associtively the packet's routing tag and the routing tag of the matched CAM location. It the match occured, only one packed is sent back. This situation occurs when only one access went through RU_i. If the match did not occur, two packets are send back: one packet generarted by RU_i from the routing tag in CAM, and the other one that arrived to RU_i.

5.4 cache and add

In section 3.1, goal 8, we give a definition of a Fetch and Add operation. We also mentioned that F&A is considered the most promising synchronizational mechanism for parallel processing. Then, it is obvious that every parallel processing project incorporates Fetch and Add.

fact 1: Fetch and Add is not implemented in our intelligent memory module,

fact 2: Fetch and Add does not work for real processing.

Let's expand the philosophical definition of Fetch and Add onto our model of the interconnection network of intelligent memory modules. Then, according to the authors of F&A, each module should funtion as in figure 5.4.1.

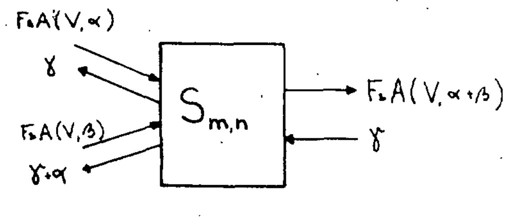


figure 5.4.1

Let's assume that switch $S_{m,n}$ receives from $S_{l,k}$ packet F&A(V,e_i) and from $S_{i,j}$ packet F&A(V,e_j). Then, switch $S_{m,n}$:

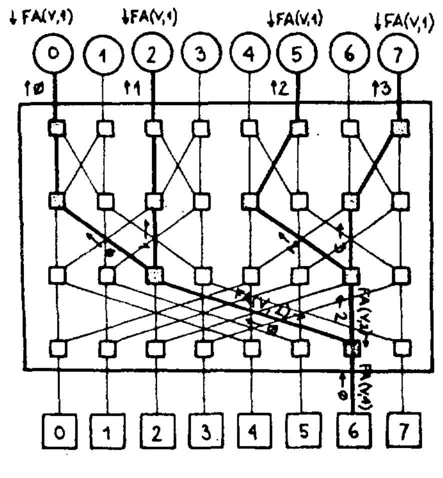
1) sends F&A(V,e_i+e_j)
2) stores in the local buffer e_i

On the way back, as the result of F&A operation, $S_{m,n}$ receives a value of V: R.

Switch $S_{m,n}$:

1) sends R+e_i back to $S_{i,j}$
2) sends R to $S_{k,l}$

Let's give an example of 4 processors simultaneously sending F&A(V,1). Initial value of V is 0.



In O($\log_2$N) steps m processors simultaneously update shared variable and receive the proper update of the variable.

Is that so?

Unfortunately, this scheme works only if the requests come to the switches trully simultaneous. However, this is not the real picture of parallel processing. Memory requests to the same shared variable may vary in time.

Let's consider the following scenario:

F&A(V,e_i) arrives at switch $S_{i,j}$ at time t_0. Some t units later F&A(V,e_j) arrives at $S_{i,j}$.

What should $S_{i,j}$ do at time t_0?

Switch $S_{i,j}$ has four options:

i) to wait for F&A(V,e_j) and then send F&A(V,e_i+e_j)

ii) to send F&A(V,e_i); later, F&A(V,e_j)

iii) to send F&A(V,e_i); later, F&A(V,e_i+e_j)

iv) to send only F&A(V,e_i)

None of these options represents the original definition of Fetch and Add.

Options iii) and iv) are incorrect. They result in improper update of V.

Options i) and ii) are correct, but are not quite what the original definition of Fetch and Add requires.

Option ii) results in memory conflicts (Conflict Filter is not valid for F&A packets).

Option i) results in unnecessary waiting. It is possible that F&A(V,e_j) does not arrive at all; hence, F&A(V,e_i) was delayed unnecessary.

It is our opinion that Fetch and Add is extremely usefull concept for operating system primitives (think of simultaneous queue updates) and that it is worth of implementing.

We introduce a new concept called Cache and Add. It uses option ii) but eliminates conflicts using CAM cache that resides in each global memory unit of the intelligent memory module. Therefore, a global memory unit is responsible for Cache and Add packets. Routing units are free from any work concerning C&A packets.

First C&A(V,e_i) request, is a request to a global memory unit to bring V from memory into cache (solely for C&A operations). Any other C&A packet results in cache update of V: adding e_i to the content of V, storing the summation into V, and returning the the original value of V. Accordingly, each routing unit simply forwards the C&A packet:

A result is:

i) access time for C&A packet is O($\log_2$N), as it is by the F&A definition,

ii) traffic among the intelligent memory modules is increased, since each C&A packet is forwarded,

iii) global memory unit logic is more complex,

iv) routing unit logic is less complex.

5.5 routing unit algorithm

We have defined all functions relevant to the routing unit. Let's assume that the system consists of $N=2^n$ processors, and that we observe RU_i of an intelligent memory module m_i. A block diagram of RU_i is shown. in figure 5.5.1.

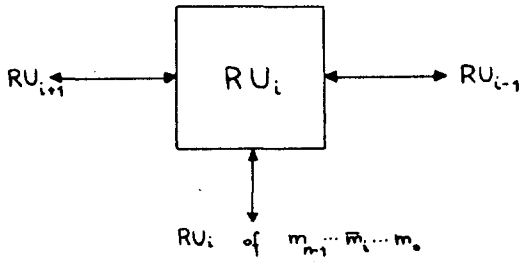


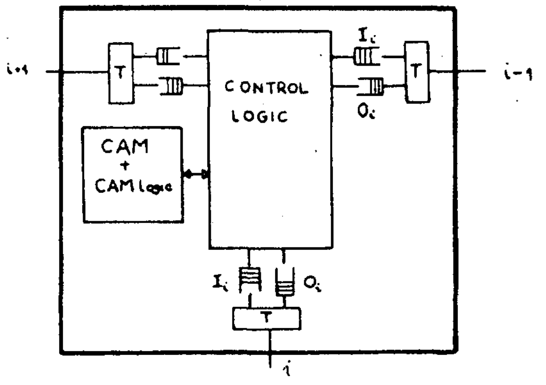
figure 5.5.1

Each RU consists of:

i) logic unit: composition and decomposition of packets, routing bit determination,

ii) CAM and CAM logic: conflict filter realization,

iii) transievers: transfer of packets between RU's (transievers work in parallel with RU's).



Let's call I_{i-1}, an input queue to RU_i from RU_{i-1}. O_{i-1} is an output queue to RU_{i-1} from RU_i. There are three pairs of such queues. Each input queue has an EMPTY line, too: queue status line.

Routing packet consists of: 16 bit control, 32 bit address, up to 128 bit data.

Control bits:

0:1 -	memory access types:
	00 - read
	01 - write
	10 - C&A
	11 - diagnostics
2 -	0 forward packet
	1 bacward packet
3 -	reconfigurable path packet
4:5 -	type of data
	00 - 16 bit data field
	01 - 32 bit data field
	10 - 64 bit data field
	11 - 128 bit data field
6:15 -	10 bit routing tag; up to 1024 processors

On the first step of operations, each module reads a packet from one of the input queues:

```
repeat
   if not empty(I_{i-1}) and not CAM_full then
                              process_packet(i-1);
   if not empty(I_i) then process_packet(i);
   if not empty(I_{i+1}) then process_packet(i+1);
forever
```

Each RU indefinitely repeats read queue operations.

```
process_packet(k)
 begin
    read_packet(k);
    if control[2:2] = 0 then forward_route(k)
         else  backward_route(k)
 end
```

A read operation involves packet decomposition and packet storing into corresponding registers: control register, address registers, data registers.

```
read_packet(k)
 begin
    control := read(k);
    if control[2:2] = 0 then
       begin
          read_address(k);
          if control[0:1] = 01 or
             control[0:1] = 10
          then read_data(k)
       end
    else
       begin
           read_address(k);
           if control[0:1] <> 11 then
              read_data(k)
       end
 end
```

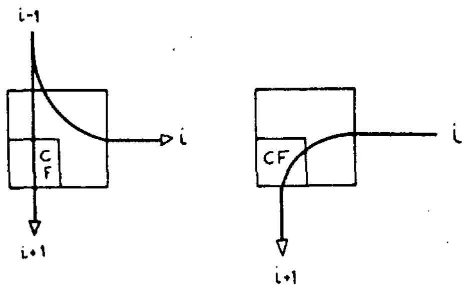
Read_address(), read() and read_data() are algorithm independant functions. They consist of selecting input and output lines and enabling the READ flag.

Write_packet() procedure is similar to read_packet() function, except that it checks if the output queue is full.

RU_i process differently memory request packets (forward), and memory replay packets (backward).

A forward packet can come to RU_i from I_i or I_{i-1}. If it comes from I_i, it can go only to O_{i+1}. RU_i does not need to make decision where to forward a packet. However, RU_i needs to execute a Conflict Filter operation.

If the packet comes from I_{i-1}, and a routing bit is 1, a packet is simple forwarded to O_i. No Conflict Filter operation is needed. RU_i of the remote intelligent memory module will do the Conflict Filter operation. Otherwise, if the routing bit is 0, RU_i executes Conflict Filter operation, and forwards a packet to O_{i+1}.



Notice that C&A packets (control[0:1]=10) simply pass through RU_i.

```
forward_route(k)
 begin
  if k = i then
   if control[0:1]=10 then write_packet(i+1)
     else
       begin
         forward_conflict_filter();
         if not match than write_packet(i+1)
       end
  else
   begin
     route_bit := shiftr(control[6:15],i);
    if route_bit = 1 then write_packet(i)
     else
      if control[0:1]=10 then
                          write_packet(i+1)
       else
        begin
          forward_conflict_filter();
          if not match then write_packet(i+1)
        end
   end
 end
```

A routing bit determination is done using the right shifter operation.

A Conflict Filter logic is the most important part of the routing unit. It consists of CAM and CAM logic that executes CAM operations: compare_CAM, insert_CAM, add_CAM, delete_CAM.

CAM logic consits of:

- CAM with m entries (16 entries we expect to be enough),
- argument register (48 bits) for storing address and control register,
- key register (48 bits) for masking parts of the argument register,
- tag_register (m bits) each bit representing if the entry is empty (0), or full (1),
- match register (m bits) representing matched CAM entries,
- output_CAM register (16 bits) representing CAM output value,
- CAM_full bit: AND(tag register),
- match bit: OR(match register),
- logic for decoding and encoding CAM addresses.

The Compare_CAM() function sets value 1 in the match register for each entry that matches an argument register. Consequently, a MATCH bit is a result of OR function on all bits in match register.

There is another invisible compare_CAM() operation that takes place when the match occurs: an output_CAM register receives a value of the stored control bits from the matched CAM entry.

```
compare_CAM()
 begin
  argument_reg[0:31] := address;
  key_reg := $FFFFFFFF0000_{16}$;
  compare_CAM_enable := 1
 end
```

These are the invisible operations of the compare_CAM() function:

```
do
  if $CAM[0:31]_i$ = argument_reg[0:31] then
       $match\_reg_i$ := 1
parallel: 0 <= i <= MAX;

MATCH := OR(match_reg);

if MATCH then output_CAM := $CAM[32:47]_k$;
```

THe Insert_CAM() function finds the first empty entry in CAM, and inserts an address and control bits in that entry:

```
insert_CAM()
 begin
  argument_reg[0:31]  := address;
  argumemt_reg[32:47] := control;
  key_reg := $FFFFFFFFFFFF_{16}$;
  l := 0;
  while tag[l] = 1 do l := l + 1;
  tag[l] := 1;
  select := CAM_decode(l);
  write_CAM_enable := 1;
  CAM_full := AND(tag)
 end
```

The Add_CAM() function, in comparison to the insert_CAM() function, adds only the control bits to CAM. This operation takes place when the first packet is in CAM, and the second packet with the same address has just arrived:

```
add_CAM()

 begin
  argument_reg[32:47] := control;
  key_reg := $00000000FFFF_{16}$;
  l := 0;
  while match_tag[l] = 0 do l := l + 1;
  select := CAM_decode(l);
  write_CAM_enable := 1
 end
```

Delete_CAM operation is very simple, and consists of tag bit invalidation:

```
delete_CAM()
 begin
  l := 0;
  while match_reg[l] = 0 do l := l + 1;
  tag[l] := 0;
  CAM_full := 0
 end
```

Now, we can describe the forward Conflict Filter logic. There are four possible situations that the Conflict Filter should process:

1) first packet is read access, the second is read access, too,

2) first packet is read access, the second is write access,

3) first packet is write access, the second is read access,

4) first packet is write access, the second is write access, too.

There are no problems with situations 1 and 4. They are described in sections 5.3 and 3.1.

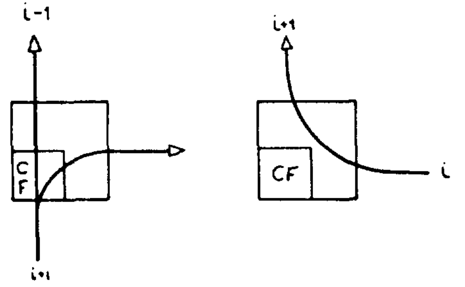
There is no problem with the third situation, either. The second packet is stored, but not forwarded. The only requirament is that the write packet, on the way back, carries the data part, too. This can produce, unfortunately, some additional traffic overhead.

The second situation could produce a problem. We solved it the following way.

We let both packets go through the Conflict Filter and through RU_i. On the way back, a read packet comes first, but it is not sent back. When a write packet comes back to RU_i, both requests are sent back. There are two problems with this solution: network traffic is increased and two requests collide at the same memory bank; memory conflict occurs. However, we do not expect, in general, this kind of nondeterministic programming, and therefore, do not expect similar situations to occur very often.

```
forward_conflict_filter()
 begin
   match := compare_CAM();
   if match = 0 then insert_CAM()
     else
       begin
         if control[0:1]=01 and
                output_CAM[0:1]=00 then
              match := 0;
         add_CAM()
       end
 end
```

A backward packet can come to RU_i from either I_i or I_{i-1}. If a packet comes from I_i, it can go only to output queue O_{i-1}. RU_i needs neither to make decision where to forward a packet, nor to execute Conflict Filter operation.



If a packet comes from I_{i+1}, a specific Conflict Filter operation takes place:

```
backward_route(k)
 begin
  if i=k then write_packet(i-1)
    else
     if control[0:1]=10 then
        begin
          route_bit:=shiftr(control[6:15],i);
          if route_bit=0 then
                write_packet(i-1)
              else write_packet(i)
        end
     else backward_conflict_filter()
 end
```

On the way back, RU_i compares the address of the returning packet with the content of CAM. If the result of comparison says that only one access took place, a packet is simply sent back to either O_{i-1} or O_i. Matched CAM entry is deleted.

Otherwise, RU_i returns both requests back. CAM entry is deleted, too.

```
backward_conflict_filter()
 begin
  match := compare_CAM();
  if control = output_CAM then
   begin
    delete_CAM();
    route_bit := shiftr(control[6:15],i);
    if route_bit=0 then write_packet(i-1)
                   else write_packet(i)
   end
  else
   if control[0:1]=00 and output_CAM[0:1]=01
      then  add_CAM()
   else
     begin
       route_bit:=shiftr(control[6:15],i);
       if route_bit=0 then write_packet(i-1)
                      else write_packet(i);
       control := output_CAM;
       control[2:2] := 1;
       route_bit:=shiftr(control[6:15],i);
       if route_bit=0 then write_packet(i-1)
                      else write_packet(i);
       delete_CAM()
     end
 end
```

5.6 summary

A routing unit is a simple but very valuable part of the intelligent memory module. In the previous sections we showed complete and correct functions of routing unit that are used in eliminating memory conflicts and achieving fast routing. We presented the functions in terms of algorithmic procedures. We think that the procedures can be improved in terms of speed and logic complexity, and that the motivated VLSI designer can find enough operations that can be parallelized either by introducing overlapping, pipelining, or distributing the logic components into more independent groups.

6. RECONFIGURABLE INTERCONNECTIONS

In a section 3.1, we defined a goal that the intelligent memory module should be capable of detecting failures and allowing bypass.

A problem with the proposed interconnection structure, as a result of Theorem 1 and goal 7 is that there exists only one path from a given processor to a given memory bank. An intelligent memory modules cannot change routing procedure, or change communication connections.

Therefore, if there is a fault on the path from P_i to M_j, no communication is possible, and M_j is permanently disconnected from P_i.

We introduce an improvement to the shared memory controller in figure 4.1.9, such that, when a failure is detected, a new path is choosen.

There are two requirements that we must follow:

1) a new path must preserve Theorem 1,

2) additional implementational hardware must be minimal.

Keeping these two requirements in mind, we propose a reconfugurable shared memory controller, as shown in figure 6.1.

A reconfigurable shared memory controller has the following additional functions:

1) it is possible to bypass switch $S_{0,i}$

2) $S_{n-1,i}$ is connected to $S_{n,j}$ in the same way as $S_{0,i}$ is connected to $S_{1,j}$

3) it is possible to bypass $S_{n,j}$

When the reconfigurable shared memory controller functions normally, a path from P_i to M_j goes through $S_{0,j}$, then through switches at levels 1 to n-1. However, the path bypasses $S_{n,j}$.

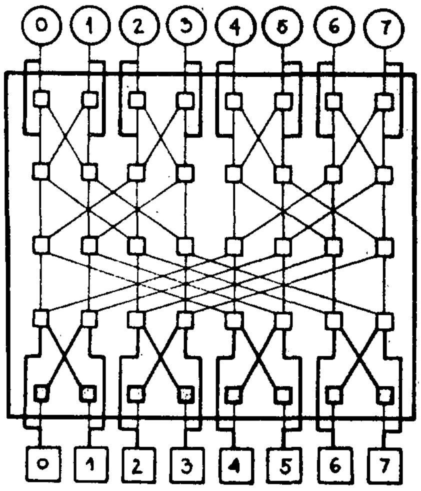


figure 6.1

There are two types of switch faults.

i) $S_{0,i}$ is faulty; $0<=i<=N$

Let's assume the routing tag: $r_{n-1}...r_0$. If $r_0=0$, then a new routing tag is $r_{n-1}...r_1$, $S_{0,i}$ bypass is enabled. $S_{n,k}$ bypass is enabled, too.

If $r_0=1$, then a new routing tag is also $r_{n-1}...r_1$. $S_{0,i}$ bypass is enabled. $S_{n,j}$ bypass is disabled.

ii) $S_{i,j}$ is faulty; $1<=i<=N$, $0<=j<=N$

$S_{n,k}$ bypass is disabled. A new routing tag is $r_{n-1}...r_1c_0$. $S_{0,i}$ bypass is disabled, too.

Let's assume that P_6 wants to access M_3, and that a switch $S_{7,2}$ failed. The initial routing tag is 101. $S_{0,6}$ bypass is disabled and the routing tag is changed to 100, as it is shown in figure 6.2.

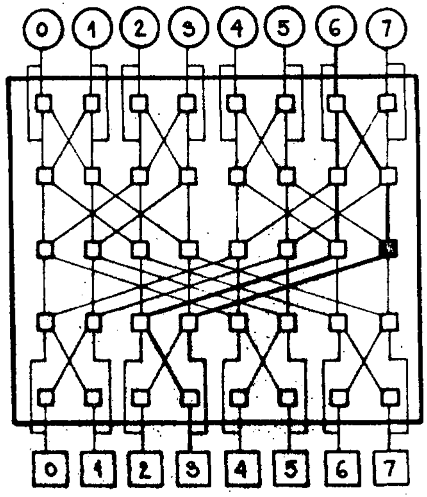


figure 6.2

6.1 single switch failure

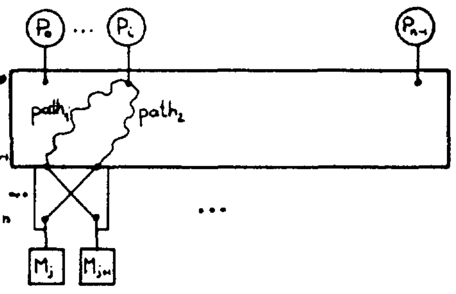
Fault tolerance could be acheived by having redundant paths from P_i to M_j.

Theorem 3:

In the reconfigurable shared memory controller, there exit exactly two **distinct** paths between any P_i and any M_j.

proof:

Theorem 1 states that there exists an unique path from a given processor to a given memory bank. A new stage in the recofigurable shared memory controller enables two paths to share the same destination.



The proof is sufficient for showing the existance of two paths. However, we must show also, that two paths that share the same destination, do not have links or switches in common.

lemma 1:

Two paths from P_i to M_j have no links and no switches in common.

proof:

Let's consider figure 4.1.9.

Let's also assume the following source-destination connections:

P_i to M_{2j} and P_i to M_{2j+1}; $j = \{0,1...n/2\}$.

From Theorem 1, routing tags from P_i to M_{2j} and M_{2j+1} differ only in the rightmost significant bit. The rest of the routing bits are the same.

Let's consider now figure 6.1.

Each $S_{2j,n-1}$ is connected to $S_{2j+1,n}$.
Also, each $S_{2j+1,n-1}$ is connected to $S_{2j,n}$.

Since the routing tags differ only in the rightmost significacnt bit, then the path from P_i to M_{2j} called the **primary** path, and path from P_i to M_{2j+1}, called then **secondary** path, have no links or switches in common.

QED.

Theorem 4:

There exists at least one fault free path between any P_i and any M_j.

proof:

There are two possible faults to consider:

i) $S_{0,i}$ is faulty; $0<=i<=N$

$S_{0,i}$ must be bypassed. Lemma 1 showed that the secondary and primary paths have no links or switches in common.

Since $S_{0,i}$ is bypassed, there are still n switches from level 1 to level n to enable fault free path.

ii) $S_{i,j}$ is faulty; $1<=i<=N$, $0<=j<=N$

Since one switch failure results only in one path disconnection for a given P_i and M_j, then by lemma 1, at most one of the paths is faulty. Therefore, fault free path exists.

QED.

Therefore, as single switch failure does not result in path disconnection.

6.2 finding fault free path

It is important that each switch can dynamically diagnose a failure of the neighbouring switches and links.

For example, switch $S_{i,j}$ can broadcast back a possible failure of either $S_{i+1,j}$ or $S_{i+1,k}$ to $S_{i-1,j}$ and $S_{i,j-1}$. At the end, each processor that uses either $S_{i+1,j}$ or $S_{i+1,k}$ receives an information about switch failure. Accordingly, the reconfigurable shared memory controller updates the routing tags by using the secondary path instead of the primary.

Theorem 5:

Each processor uses the primary path:

i) $r_{n-1}...r_0$; $S_{0,i}$ bypass disabled; $S_{n,j}$ bypass enabled

or the secondary path in a case of a switch failure:

ii) $r_{n-1}...r_1c_0$; $S_{0,i}$ and $S_{n,j}$ bypass disabled, for $S_{k,l}$ fault and $k>0$

iii) $r_{n-1}...r_1$, $S_{n,j}$ bypass enabled; $S_{0,i}$ bypass enabled, for $S_{0,i}$ fault and $r=0$

iv) $r_{n-1}...r_1$, $S_{n,j}$ bypass disabled; $S_{0,i}$ bypass enabled, for $S_{0,i}$ fault and $r_0=1$

proof:

i) Proven by Theorem 1.

ii) All bits of the routing tag are equal except r_0. Since path $r_{n-1}...r_1 1$ and path $r_{n-1}...r_1 0$ have no links or switches in common, by Lemma 1, and since there exists a transition from level n-1 to level n, the secondary path routing tag is correct.

iii) A routing tag $r_{n-1}...r_1 0$ and a routing tag $r_{n-1}...r_1$ with $S_{0,j}$ bypass enabled are identical. Therfore, the secondary path is same as the primary path, from level 1 to level n-1.

iv) It is the same as for ii) if we consider that the bypass of $S_{0,i}$ means the same as having $r_0=0$.

QED.

Each switch dynamically receives and forwards a fault label of the failed switch. At the end, each processor receives coordinates of the failed switch.

Let's assume that $S_{k,l}$ failed.

Definition:

Fault Tag f, is a function result of **EOR**, and k (failed switch level), which tells to the reconfigurable shared memory controller where the location of the failed switch is.

Fault Tag = P_i **EOR** M_l = $xx \ldots xf_k \ldots f_0$

where, x = don't care.

This means that for a path from P_i to cross $S_{k,l}$, a routing tag must not be equal to the fault tag: $xxx \ldots xf_k f_{k-1} \ldots f_0$. This is summarized in the following theorem.

Theorem 6:

If the routing tag, $r_{n-1} \ldots r_i r_{i-1} \ldots r_0$, and the fault tag $xxx \ldots xf_i f_{i-1} \ldots f_0$ agree in i+1 low order bits, the routing path uses a fault switch.

proof:

The proof is by the induction on i, number of low order bits.

Let's consider only $r_i \ldots r_0$ and $f_i \ldots f_0$; let's also assume that they meet at stage i, but have different paths.

i) case 0 (two paths meet at level 0)

If r_0 <> f_0, two paths cannot meet at level 0. Therefore, $r_0 = f_0$.

ii) case i (two paths meet at level i)

Let assume that $r_j \ldots r_0 = f_j \ldots f_0$ for $j < i$.

For two paths to meet at stage j+1, $r_{j+1}=f_{j+1}$.

Hence, for two paths to meet at stage i, they must have all i bits the same:

$r_{n-1} \ldots r_i \ldots r_0 = xx \ldots xf_i \ldots f_0$.

QED.

In order to implement the fault tolerant procedures, a reconfigurable shared memory controller must maintain a fault tag table. When a new routing tag r is generated, r is associatively compared with a fault tag table. If the match occurs, a secondary path must be used.

6.3 multiple fault tolerance

In the previous section we showed that a single switch failure does not result in processor memory disconnection. Moreover, for certain instances of multiple switch faults, the reconfigurable shared memory controller should also maintain fault free capability.

Multiple fault tolerance is price/performance feature. The neccessary and sufficient condition for our approach is that the primary and the secondary paths are not both faulty.

An extreme example of an expensive multiple free fault reconfigurable shared memory controller would contain n-1 secondary paths, or n extra levels.

Our approach is different. We allow multiple switch faults. However, if both the primary and the secondary paths are faulty, rescheduling must be used. For a processor that has both paths to M_j faulty, it is equivalent as having M_j not operational.

Theorem 7:

Let $L_1=x \ldots xf_i \ldots f_0$ and $L_2=x..xg_j \ldots g_0$ be two fault routing tags, j <> i.

The interconnection structure is fault free if there are no two fault tags that have:

i) f_0 <> g_0
ii) $f_i \ldots f_1 = g_i \ldots g_1$
iii) else don't care

proof:

The proof is obvious, since we have proven that there exist two paths, such that the first rightmost significant bit of the primary path is different from the most significant bit of the secondary path; the rest of the bits are the same.

Therefore, the interconnection is fault free if either the primary or the secondary path is fault free.

QED.

6.4 interconnection transformation

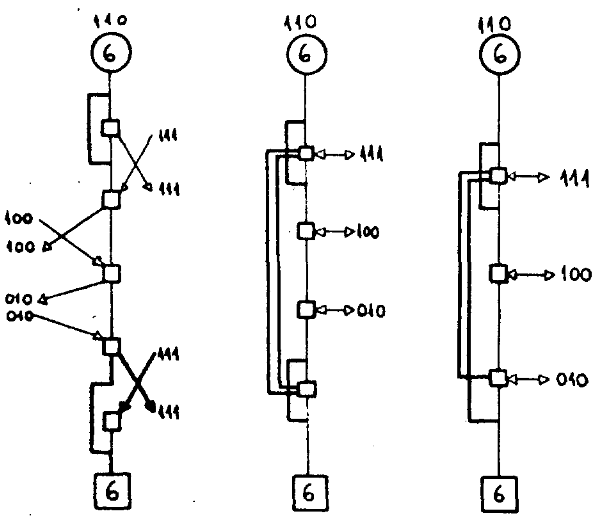
For the very same reasons as in section 4.2, it is important that a reconfigurable shared memory controller can be transformed into N simpler and less expensive reconfigurable intelligent memory modules.

Theorem 8:

Reconfigurable shared memory controller can be transformed into N independent, smaller and modular reconfigurable intelligent memory modules.

proof:

We will use the same procedure as in Theorem 2. Let's isolate any processor from figure 6.1.



Since the connection between level n-1 and level n is the same as the connection between level 0 and level 1, then by applying Theorem 2, we prove that the reconfigurable shared memory controller can be transformed into N modules, too.

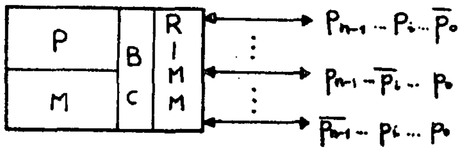
QED.

Collorary 2:

Reconfigurable shared memory controller from figure 6.1 can be transformed into **reconfigurable HyperCube.**

proof:

The proof is the same as for Collorary 1.


P
M
B
C
R
I
M
M
$P_{n-1} \cdots P_i \cdots \overline{P_0}$
$P_{n-1} - \overline{P_i} \cdots P_0$
$\overline{P_{n-1}} - P_i \cdots P_0$

QED.

The most important feature of the presented metamorphisis is, that all the properties of the powerfull, general and complicated reconfigurable shared memory controller have not been lost in a process transformation to a simple structure such as HyperCube.

7. RECONFIGURABLE ROUTING UNIT

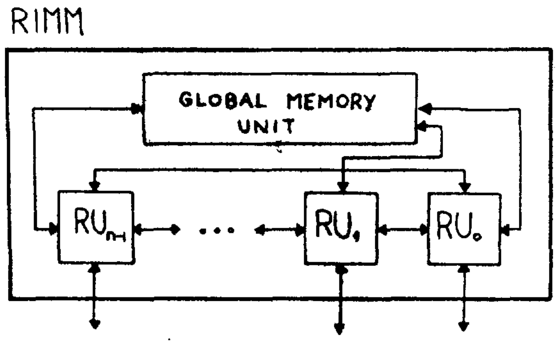
In this section, we will expand the design of the routing unit that we started in section 5.1, by adding to the routing unit the reconfigurable functions defined in section 6. Unfortunately, in the process of adding the reconfigurable functions to the intelligent memory module hardware some modularity was lost:

1) routing units RU_0, RU_1 and RU_{n-1} are different from other units. They can forward packets not only to O_{n-1}, O_{n-2}, O_n, but also to an output queue that we call O_0. Furthermore, they can receive packets from I_0.

2) each routing unit has additional logic such as: logic for generating and checking failures, logic for broadcasting diagnostics packets.

3) extended hardware in the global memory unit: fault tag table, logic for choosing primary and secondary path.

Since modularity is the key principle of a successfull VLSI design, we beleive that all routing units should have the same hardware complexity and functions.



At the time of system implementation, RU_{n-1} will be connected to RU_0 using port 0. RU_1 is connected to the global memory unit using port 0. The rest of the connections is the same as described in 5.1.

7.1 error checking generator

There are two approaches to detect routing unit failure:

i) checking errors dynamically,

ii) performing diagnostics prior to system operation.

To identify a fault's location, an error checking hardware must be build into each module. We will assume the second approach, just for the presentation simplicity, although the algorithm that we present is general enough to give insight into approach using dynamic error checking.

We suggest method called cyclic reduction code or CRC code. The basic idea is to append a checksum to the end of the packet in such a way that the polynomial represented by the checksumed packet is divisible by a generator polinomial known to each RU_i.

Each RU_i can check a failure of RU_i or RU_{i+1} dynamically and independently. RU_i is fault free if RU_{i-1} announces to R_i that RU_i is functional and eligable for checking failure of RU_i and RU_{i+1}. Initially, the global memory unit checks if RU_0 is operational. If so, RU_0 can check a status of RU_1 or RU_0 of another intelligent memory module.

A diagnostics packet has different control bits:

0:1	-	11 represents a diagnostic packet
2:3	-	RU_i state

```
        01: receive-return state
        11: check state
        00: wait state
        10: operational; can broadcast
            failures back,
```

4:15	-	number of failed RU

An operational RU_i, sends a packet to O_i and O_{i+1} and waits. If the returning packet does not arrive, or it arrives with the changed checksum, a failure is reported back to all operational units. If not, the unit is acknowledged to be operational.

```
diagnostics(k)
 begin
   if control[2:3]=01 then
    begin
      control[2:3]:=11;
      write_packet(k)
    end
   else if control[2:3]=11 then
    if check_parity() <> 0000000$_{16}$ then
     begin
      control[2:3]:=10;
      report_failure()
     end
    else if k=i-1 then
      begin
        control[2:3]:=00;
        write_packet(k)
      end
   else if control[2:3]=00 then
    begin
     write_packet(i); write_packet(i+1);
     $t_1$:=wait($I_i$); $t_2$:=wait($I_{i+1}$);
     if $T_{max}$ > $t_1$ then report_failure(i);
     if $T_{max}$ > $t_2$ then report_failure(i+1)
    end
   else broadcast_back_failure(k)
 end
```

Each entry in a fault tag table is of the form: $x...xf_i...f_0$, which means that there exists RU_i, which is not operational. For a packet to reach RU_i, a processor needs to generate a following routing tag: $x...xr_i...r_0$. A fault tag can be generated dinamically, so that each RU that is on the way to RU_i, adds the corresponding bit of the fault tag.

For example, if RU_i finds out that RU_{i+1} is faulty, then the fault tag is xxx..x0. If RU_i receives a diagnostic packet with x...x1011 from RU_{i+1}, it generates x..x10110 diagnostics packet, and sends it to RU_{i-1} and RU_i.

```
report_failure(k)
 begin
   address:=k;
   control[6:15]:=1111111111;
   broadcast_back_failure(k)
 end

brodcast_back_failure(k)
 begin
  if k=i then
   begin
    control[15:15]:=1;
    write_packet(i-1)
   end
  else
   begin
    shiftl(control[15:15],1);
    control[15:15]:=0;
    write_packet(i);
    write_packet(i-1)
   end
 end
```

7.2 reconfigurability algorithm

The main idea in designing a reconfigurable routing unit is to use as little additional hardware as possible, compared to the standard routing unit described in section 5.

A block diagram of the reconfigurable routing unit is given in figure 7.2.1. In this section we present all additional features of the routing unit that involve fault tolerance and reconfigurable routing.

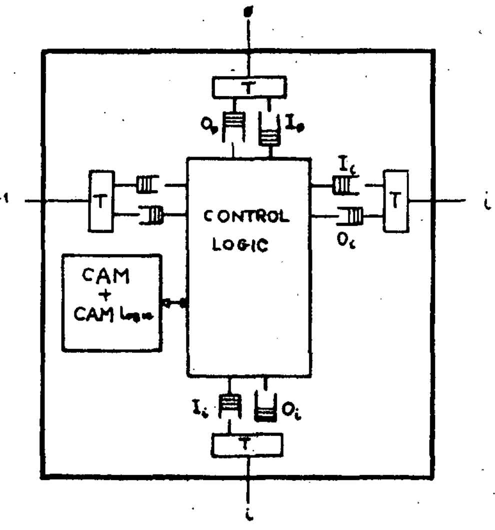


figure 7.2.1

Each unit reads a packet from one of the input queues. Only RU_0, RU_1 and RU_{n-1} can read from I_0. For other modules I_0 is always empty.

```
repeat
   if not empty($I_{i-1}$) and not CAM_full then
                           process_packet(i-1);
   if not empty($I_i$) then process_packet(i);
   if not empty($I_{i+1}$) then process_packet(i+1);
   if not empty($I_0$) then process_packet(0)
forever
```

```
process_packet(k)
 begin
    read_packet(k);
    if control[0:1] = 11 then diagnostics(k)
    else
     if control[2:2] = 0 then
               process_forward_packet(k)
         else  process_backward_packet(k)
 end
```

When a processor is using the secondary path, as the result of some RU failure, RU_0 of the last intelligent memory module is used as an extra stage. We call RU_0 a detour routing unit. If a packet represents a memory request, then RU_{n-1} of the last intelligent memory module sends a packet to its RU_0 through O_0.

```
write_packet_0()
 begin
   if control[3:3]=1 and i=n-1 then
     write_packet(0)
   else write_packet(i+1)
 end
```

RU_0 will send a packet to RU_0 of the corresponding intelligent memory module. That RU_0 will access his global memory unit.

```
process_forward_packet(k)
 begin
  if control[3:3]=1 and i=0 then
    forward_detour(k)
  else forward_route(k)
 end
```

```
forward_detout(k)
 begin
   if k=0 then write_packet(i)
   else write_packet(i-1)
 end
```

A forward_route() procedure is the same as in section 5.5.

On the way back, a packet will travel the same path. It first arrives at the detour RU_0, than at RU_0 of the intelligent memory module that has RU_{n-1} and so on all the way back.

```
process_backward_packet(k)
 begin
  if control[3:3]=1 and (i=0 or i=1) then
     backward_detour(k)
  else backward_route(k)
 end
```

```
backward_detour(k)
 begin
  if i=0 then
   if k=m-1 then write_packet(i)
     else write_packet(0)
  else
   if control[15:15]=0 then write_packet(0)
   else write_packet(i-1)
 end
```

7.3 summary

It is possible with very little extra hardware to achieve a reconfigurable, fault tolerant intelligent memory module.

The routing unit functions are the lowest level functions of a large scale, general purpouse parallel system. We developped extremely powerfull and general, but simple and fast routing unit that supports fault tolerance. This routing unit is the key element in the overall system performance.

8 GLOBAL MEMORY UNIT

In section 5 we defined 13 functions that we consider key functions in achieving fast and general purouse parallel processing. We decided to implement these functions using the intelligent memory module. Ten of these functions we grouped into NANO level design; the rest of them to MICRO level.

We can divide the intelligent memory module functions into 3 groups:

i) routing unit functions (we described these functions in previous sections),

ii) global memory unit functions that support routing unit functions,

iii) global memory functions that support MICRO level.

Third group of functions need not to be implemememented into the intelligent memory module hardware. Functions, such as process sceduling, memory protection, etc., can be done on a processor level, too.

In this section we are concerned only with the second group of functions. In other words, we want to describe the overall routing algorithm, from address encoding to global memory access. We devide global memory functions that support routing (group ii) into two classes:

i) pre routing unit operations,

ii) post routing unit operations.

A flow diagram of pre routing unit operations is as following:

1) get address, data and status

2) determine memory bank number

3) generate routing tag

4) compare routing tag and Fault Tag Table

5) generate access packet

6) send packet to RU_0 or RU_1

A flow diagram of post routing operations is as following:

1) read packet from RU_{n-1}

2) decompose packet

3) execute C&A

4) access Global Memory

5) generate return packet

6) send packet to RU_{n-1} or RU_0

These functions will not be described in details, as it was for the routing unit functions. It is obvious, even from the flow diagrams, that these functions are simpler and easier to implement. In figure 8.1 we give a block diagram of the intelligent memory module with some emphasis on global memory unit.

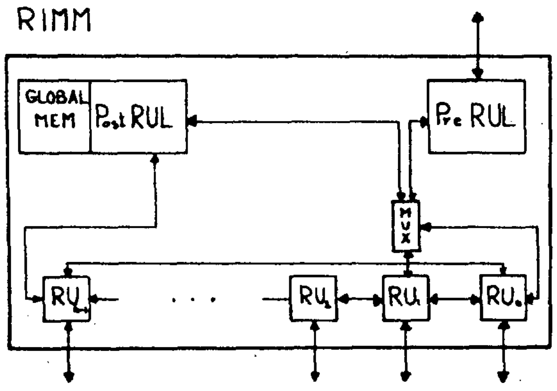


figure 8.1

8.1 pre routing unit logic

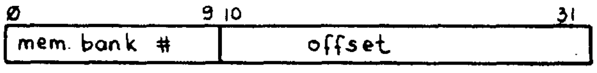
8.1.1 address translation

In general, when a system consists of a number of memory banks, a memory controller must select which bank to access. Since we transformed the memory controller into N intelligent memory units, the pre routing unit logic (preRUL) of each intelligent memory controller must select which bank to access.

In section 3.1, goal 9, we defined two types of memory conflicts: conflicts that occur at the same memory location and conflicts that occur at the same memory bank. We solved problems concerning conflicts that occur when N processors access the same memory location.

- How do we solve memory access conflicts that occur when N processors access different locations in the same memory bank?

Most of today's general purpouse computers, and almost all special purpouse computers use what is call "low order interleaving", or simply interleaving. Memory banks are selected by decoding the low order bits (the least significant bits) of the memory address:



or analitically:

memory bank number = address MOD N
offset = address / N.

In general, memory banks are arranged so that N **sequental** memory addresses: i, i+1, ... i+N-1 fall in N distinct banks. A key word in defining and implementing interleaving effectively is: sequential. In other words, as long as the generation of the next memory access is not dependant on data received from the previous access then sequential accessing can keep all banks busy simultaneously.

It is therefore, important to arrange data in such ways that the memory access patterns take advantage of the interleaving scheme, effectively. However, this is contrary to our goal 1. We don't want and don't need this kind of interleaving.

The following theorem shows the degradation of interleaved memories with respect to nonsequential memory accesses.

<u>Theorem</u>[Kog81]

Assuming that a memory design consists of N individual memory banks, and the access time for any bank is T, then for an address sequence with uniform spacing of S the average time per element access is:

i) $\frac{ST}{N}$ for $S <= N$

ii) T for $S > N$

<u>proof:</u>

i) $S <= N$

Let's assume spacing S. Then, each memory access involves k memory banks: S, 2S, 3S, ... kS, such that k = N/S. Hence, the avarage access time is: T/k, or ST/N.

ii) $S > N$

On the first memory access one memory bank is busy. On the second memory access, also one bank is busy, etc. Hence, the average access time is T/1 = T.

QED.

Various different schemes have been proposed to minimize Theorem 9, such as having prime number of banks, skewing, etc. However, they all assume, either vector or SIMD (single instruction multiple data) mode of operation. In other words, they assume uniform spacing. In the multiprocessor systems, where each processor can execute independent and different task, memory access patterns are, in general, nonuniform.

We propose <u>random</u> interleaving for eliminating memory conflicts that occur when N processors access different locations in the same memory bank.

8.1.1.1 random interleaving

It has been shown that the uniform interleaving is ineffective even for uniform memory access spacing.

Let's assume figure 3.1.2 and the following shared memory controller operation.

Shared memory controller intercepts an address and sends it to memory bank 1; second address it sends to bank 5, third to bank 15, ... and so on. Memory accesses are distributing by some pseudo random, but fixed function. We call this scheme random interleaving.

- How good is random interleaving ?.

On the average, there are less than N requests entering the shared memory controller. Since these requests are randomly distributed, intuitively, there is on the average slightly less than one request going to each memory.

Let's define P_i^{NB}, a probability that there are no memory conflicts at memory bank M_i.

In other words, $P_i^{NB} = P_i(m=0) + P_i(m=1)$, is a probability that at most one request arrives at M_i.

Then,

$$P_i(m=0) = \left(\frac{N-1}{N}\right)^M$$

$$P_i(m=1) = \frac{1}{N}\binom{M}{1}\left(\frac{N-1}{N}\right)^{M-1}$$

for $N=M$, $N \to \infty$,

$$P_i^{NB} = 2e^{-(N-1)/N}.$$

If we define $P_i^B(m=1)$, a blocking probability with one memory access waiting, then

$$P_i^B(m=1) = 2.5e^{-(N-2)/N}.$$

Unfortunately, these probabilities offer no practical visualization of the whole system, since these probabilities concentrate only on a particular memory bank M_i. They say, for example that a choosen memory bank M_i has $P_i^{NB}=0.26$, or $P_i^B(m=1)=0.08$. However, other M_j's are not involved in the calculation.

Hence, we are more interested in probability that is a function of all accesses and all memory banks.

Let's define $P(k_1=K_1, k_2=K_2, \ldots, k_i=K_i)$, a probability which says what the probability is that K_i memory banks have k_i memory accesses, for $1 <= i <= N$.

For example, let's assume 4 memory banks, and 4 memory accesses. Then,

- $P(k_1=4, k_2=0, k_3=0, k_4=0)$ is probability that 4 banks received 1 access.
- $P(k_1=2, k_2=1, k_3=0, k_4=0)$ is probability that 2 banks received 2 accesses, and 1 bank 2 accesses.
- $P(k_1=0, k_2=0, k_3=0, k_4=1)$ is probability that 1 bank received 4 accesses.

$$P(k_1=K_1, k_2=K_2, \ldots) = \frac{\prod_{i=1}^{i=S}\binom{M-\sum_{j=1}^{i-1}K_j}{K_i}\prod_{r=0}^{r=K_i-1}\binom{N-\sum_{j=1}^{i-1}jK_j-ir}{i}}{M^N}$$

$$\prod_{i=1}^{i=0} = 1 \qquad \sum_{i=1}^{i=0} = \emptyset$$

Probability $P(k_i=K_i)$ calculations involves all possible combinations of K_i's that satisfy: iK_i summation. For example, for a 16X16 processor memory system, there are 136 such combinations. We developped a program for calculating $P(k_i=K_i)$. We will give only an illustration analysis for 16X16 system. More thorough analysis is left for some future papers.

There are three interesting results from the analysis of 16X16 system:

1) access distribution with the highest probability:

k=1	k=2	k=3	k=4	k=5	k=6	
7	3	1	0	0	0	14%

14% of time there are 7 banks with one memory access, 3 banks with 2 accesses, and 1 bank with 1 access.

2) expected number of memory banks with k accesses:

k=1	k=2	k=3	k=4	k=5	k=6
6.08	3.04	0.95	0.20	0.03	0.00

It is to expect, on the average 6 banks with 1 acccess, 3 banks with 2 accesses, 1 bank with 1 access.

3) probability at least one memory bank has more than k accesses

k>1	k>2	k>3	k>4	k>5	k>6	
100	80	23	3	0.4	0.0	%

It is 80% that there exists a memory bank that has more that 2 accesses.

These results are very encouriging. The queues that generate in front of the memory banks are very short. Furthermore, the results are obtaines under the most conservative constraints: N accesses entering N banks. We expect even better results for smaller number of request.

Let's assume low order interleaving. Ten low order bits represent memory bank number. The rest of the bits represent an offset inside the memory bank.

We have to build a hashing function that will randomize low order 10 bits. Let' call them S bits. A sufficient constraint is that the function is fast and simple on S. For example, multiplication of S by itself and using middle 10 bits.

8.1.1.2 routing tag generator

From newly generated low order interleaving bits, S bits, we generate a routing tag by simply:

R-tag = S <u>EOR</u> P-number

where P-number represents processor or intelligent memory module number.

8.1.2 fault tag table

In Theorem 6, we state that if the routing tag and the fault tag agree in i+1 low order bits, the routing path crosses a fault switch. We also mentioned that a reconfigurable intelligent memory module must maintain a fault tag table. The pre routing unit hardware consists and executes functions concerning reconfigurable, tault tolerant routing.

A fault tag table is a content addressable memory. Its organization and functions are the same as for the Conflict Filter.

During diagnostics, a pre routing unit logic of each intelligent memory module receives the coresponding fault tags of not operational routing units.

During normal intelligent memory module operation, a pre routing unit checks if the generating routing tag matches the fault tag table. If so, a reconfigurable path is chosen using operations defined in Theorem 5.

8.1.3 packet creation

A pre routing unit hardware, generates a packet that consists of:

1) 16 control bits from status bits that came from processor and from generated routing tag,

2) memory bank offset as address,

3) data if write or C&A access.

A packet is sent to an input queue I_{i-1} of RU_0 for normal oerations, or to I_0 queue of RU_1 if RU_0 is not operational.

8.2 post routing unit logic

PostRU reads a packet either from RU_{n-1} for normal requests, or from RU_0 for bypass requests. There are two main operations of PostRU: C&A operations and Global Memory reads or writes.

8.2.1 Cache and Add Table

As we described in sections 3.1, 5.5 and 5.5, we changed Fetch and Add synchronizational primitive into Cache and Add.

PostRUL process each C&A request as following. Let's assume the following C&A format: C&A(V,e).

PostRUL compares V with C&A table. C&A table is content addressable memory, too.

If match occurs, PostRUL increments V by e.

If match does not occur, PostRUL brings V from global memory, increments V and inserts V into C&A table.

In both instances, PostRUL returns a non incremented value of V to either I_{i+1} queue of RU_{n-1} for normal opeartion, or to I_{i-1} of RU_0 for bypass opeartion.

8.2.2 global memory

One of the characteristics of the INHERENT design approach, is a tight connection between a processor and its local memory. In general, processor works locally, using data and instructions from its local memory. These operations are fast. Only when a processor requires a shared information, it generates a global memory access. That global access arrives at PosRUL of some remote intelligent memory controller.

- What should PostRU do?

An access to a local memory of the processor from an intelligent memory module is inefficient and expensive. It is inefficient, because it has to interrupt processor's operation, and go through a bus controller to access local memory. It is expensive, because the intelligent memory module looses its easy to add modularity.

fact: memory elements are the cheapest and the most compact digital components

Furthermore, the trends show that the memory capacities will increase while their price decrease, both absolutely and relatively. Typical examples are large supercomputers with hundreds megawords of memory.

We decided to follow the trend. Intelligent memory module contains a bank of a global memory.

This is the reason why we call our contribution to the parallel processing intelligent <u>memory</u> module.

8.3 summary

The intelligent memory module can be considered as an add on memory to a standard computer system, which connected with the other intelligent memory modules can create a large scale, general purpouse parallel processor.

Even without MICRO level functions, that we also plan to incorporate into an intelligent memory module hardware, an intelligent memory module can be advertized and sold as **the memory for the future.** Since it incorporates simple and modular logic of the routing units, PreRUL and PostRUL, a price of the intelligent memory module is bounded by the price of the memory chips that it incorporates. Therefore, for a slightly higher price of an ordinary memory, and additional price for noiseless fiber optics cables, a user can buy and incorporate into his system a powerfull structure that supports parallel processing in the highest possible form with:

i) fast routing,
ii) fault tolerance, reconfigurability,
iii) memory conflict elimination,
iv) fast global memory access,
v) mechanisms for fast scheduling and synchronization.

It is known that with good hardware must also come good software for hardware to be effective.

The intelligent memory module cannot be sold successfully without adequatelly well written compiler or part of the operating system. The whole idea of the INHERENT design approach is to use as much available standard products as possible. That means software, too. Unfortunately, we cannot say that we succeeded in that respect. The intelligent memory module that we designed, differentiates local processor memories from global memories. It requires from a compiler to associates different memory allocation patterns for local and for shared variables. This requirement results in more changes in sotware, particlarly, memory management, then that we expect for the INHERENT design approach. Therefore, it is our goal, in the future studies to transpass the gap between the local and the global memories and make the intelligent memory transparent to the compiler writers.

Even when the system software is done "from the scratch", it is better for the system programmer to visualize the whole memory space as one contigent unit. In the next section we offer some ideas how to eliminate the difference between the local and the global memories without performance degradation. In other words, we show some ideas how to design a better system using the NOVEL design approach.

9. NOVEL DESIGN APPROACH

There are few deficiencies of the reconfigurable intelligent memory module that result in either speed or generality degradation. One of these deficiencies is a remote memory access latency time. Or in other words, a time it takes a processor to receive data from the remote global memory. In section 4.3.1 we showed that the latency time is a function of the routing unit delays plus the queuing delay. In this section we briefly show how it is possible to eliminate the memory latency time.

The other deficiencies of the intelligent memory module result in system generality degradations more than in speed. In general, these deficiences come from the fact that the system differentiates local from global memories. We will also briefly show how to increase system generality by eliminating the difference between local and global memories. This will enable system prgrammers to visualize memory as one huge bank and enable them to write more effective memory management functions.

9.1 parallel processor

The key solution to this problem is stated in the principle number two in section 2.1 which says that there must be an abundant supply of usefull work. What does it mean?

The algorithms that programmers develop must be organized in terms of individual processes which are designed to generally operate asynchronously in parallel. For most of the algorithms, number of generated processes is much larger than number of processors.

The trick now is not to concieve of the processor as being assigned to a single process until its completion. Rather think of the processes as the set of activities to be done and the processors as a set of objects which can perform the activities.

Let's consider the following scenario. A processor takes a process from the set of processes and begins working on it. It continues to work on this process until the process generates a memory requests which must be sent to the remote global memory. The processsor then sends the request to the intelligent memory unit, but does not sit and wait for the result to come. The processor then sets that process aside and picks up another and proceeds to operate on this new process until this process makes a global memory request. The processor continues to work on different processes until one of the requests arrives from the global memory. At that time a processor can proceeed with the process that required global memory access.

The number of processes necessary to keep the processor busy is bounded by $O(\log_2 n)$.

- How do we implement processor multiplexing?

Current VLSI technological trends, with RISC architectures as the processor design examples, incorporate, in general, large number of registers. These registers are used by compilers to optimize overlapping and pipelining. We can use these large sets of registers inside each processor for eliminating global memory latency time, too.

Let's call processor registers: PC (program counter), SP (stack pointer), IR (instruction register), MAR (memory address register), MBR (memory buffer register) S (status register) and other general purpouse registers, processor's dynamic context (DPC). Let's also assume that a processor can have $O(\log_2 n)$ such DPC's. This is possible using current technology, even for large n. Hence, we see that we can effectively and quickly execute process multiplexing or what we call DPC exchange.

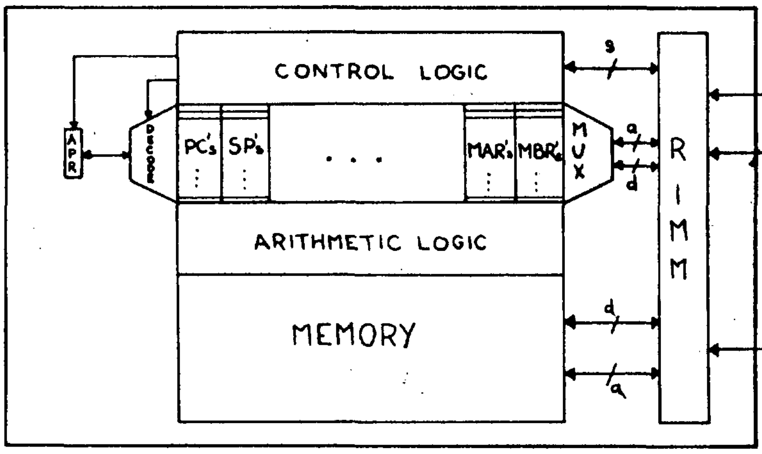


figure 9.1

There are some requirements in order to implement DPC exchange effectively. Addressing must be indirect, through the active process register (APR). For example, APR = 0, represents first set of registers, APR = 1, represents a second set of registers, and so on. A load R_5 instrustion is actually a load $R_5[APR]$ instruction. APR is incremented when a DPC exchange occurs:

APR := (APR + 1) MOD n

where n is a number of register sets.

It is very important to execute DPC exchange fast. This can be achieved only if the processor and its intelligent memory module effectively complement each other.

Another requirement to the effective processor utilization is that the number of processes is larger than the number of processors. Let's explain this statement.

Note that the processor multiplexing does not decrease memory latency. Processor multiplexing results in increased throughput and increased processor utilization. Let's assume that a rate of a processor is P units. Then the processor multiplexing with k sets of registers is equivalent of kP processors each one with rate of P/k. Therfore, it is important to have at least Pk processes to take advantage of processor multiplexing.

Let's assume standard fetch, decode, fetch operands, execute memory cycles. Let's also assume that we have k virtual processors. That means that the degree of processor multiplexing is k. When a virtual processor generates an address, that address comes to an intelligent memory module. The intelligent memory module quickly decodes the memory bank number corresponding to that address:

1) new_address := random_HASH(address)

2) bank_number := new_address MOD N

It is important that the random hashing function in 1) is fast and simple.

If the bank_number is the same as the processor number, than the address is in processor's local memory. The intelligent memory module signals a processor to wait for the memory access completion.

If the bank_number differs from the intelligent memory module number, than the address reside in some remote memory bank. In order to access that memory bank it is necessary to travel through $O(\log_2 n)$ routing units. The intelligent memory module signals to the virtual processor not to wait for the memory access completion, but to execute DPC exchange.

Each intelligent memory module is capable of storing data and status into each virtual processor's MBR or status register. This is necessary since virtual processors and intelligent memory modules work asynchronously. For example, if a virtual processor V request memory address A, and A resides in some remote memory bank, A is sent to the routing units. When the request returns, it's information is stored into V's MBR or status register. Next time when the virtual processor V becomes active, it will have the requested information available. If the system is designed such that a minimum multiplexing time is larger than the remote memory latency, V needs not to check if the information is present or not. It can go directly on with the next microinstruction.

- What happens with a processor when the intelligent memory module signals that the memory access is remote?

There are two things that a processor must accomplish It must save the status of the current virtual processor and it must start the next virtual processor. Both operations can be done very fast. In order to save the status of the current virtual processor, a control logic must save the current microinstruction into a Virtual Processor Table, indexed by APR.

In order to start a new virtual processor, the control logic must increment APR, and must set the microinstruction program counter to value of VPT[APT].

A block diagram of the proposed parallel processor is given in figure 9.1.

Besides the operations concerning virtual processors, all other performance enhancement operations such as pipelining, overlapping, multi functional units, reduce instructions, etc., can be implemented in the parallel processor.

The gratest advantage of the parallel processor is that it nullifies the routing time delay. Therefore, the intelligent memory modules could become even more complex and more time consuming elements of the system. We can add more power and more intelligence to them, without expecting speed degradation. This is very important when we have in mind that the intelligent memory module should process functions such as process scheduling, synchronization scheduling and other operating system primitives that result in large network traffic.

9.2 address distribution

Using an intelligent memory controller as the address translator, memory bacomes more transparent to the system software, particulary to the memory management routines. In the INHERENT design approach, a compiler must determine which variables are local, and which are global, and accordingly must preceed with the memory allocation. One of the problems of the separate local and global memory design is the program code duplication. If N processes execute the same process code, then N multiple copies of the process code reside in respective local memories. The problem is also with process scheduling. For example, if a process runs on a processor P_i, and synchronization takes place, than the process is blocked. Next time when the process is reactivated, it could be allocated to some other processor P_j, which needs not to have process code in its local memory. Unnecessary overhead takes place.

Let's consider the address distribution scheme. In this scheme there exists a single process code copy, or in other words a single copy of the whole program. Local program variables are allocated on the stack; therefore, they are also part of this huge bank of memory. Since all addresses are randomly distributed, so are the local variables and program code instructions. In the INHERENT design approach only global variables are randomly distributed.

A single code in memory could not be implemented without the Conflict Filter operations. Too many memory conflicts would occur.

In section 8.3 we mentioned that each intelligent memory module contains it's own memory. For the INHERENT design approach we called that memory a global memory. This memory can be used as cache for memory addresses that are not present in the processor's local memory. This could eliminate frequent virtual processor DPC exchanges, which occur when the code is distributed randomly among the memory banks.

9.3 summary

The NOVEL desing approach leads to a processor, which in combination with the intelligent memory module can acomplish all the postulates that we defined for the large scale parallel processing.

Most of the functions of the proposed parallel processor are to be discovered, yet. The principles are known, but the practical implementational solutions or even simulational solutions are still beyond current research state. The NOVEL design apprach has been the least studied area of the system design, because we do not expect to build the parallel processor in close future. However, it is important to keep in mind functions of the parallel processor and build the intelligent memory module in such a way that the processor could be easily incorporated with the intelligent memory module into a powerfull and fast parallel system.

10. CONCLUSION

The primary goal of this project is profit. If we, in the process of achieving this* goal succedded in finding some new ideas, or developped tools, mechanisms or algorithms that are novel to the computer science community, we consider these tools and algorithms jusfied and usefull only if they help us in approaching this goal, faster and more effective.

There are very few computer firms that can compete, for example, with IBM, in building a parallel machine, such as their RP3. Therefore, we offer a different course in taking over the parallel processing oriented community. We propose multistage project developpment.

On the first stage, which we call the **intelligent memory** development stage, we offer a reconfigurable intelligent memory module, without MICRO functions. This module is marginally more expensive than memory that it contains, but with all the inovations and parallel concepts incorporated, it is attractive enough to be sold profitable. With cleverly written cross-compiler, and cleverly modified operating system, the personal computer oriented community interested in parallel processing could be taken over.

On the next stage, MICRO level operations could be added to the **reconfigurable** intelligent memory module hardware, creating even more attractive and more powerfull building block for a parallel processing system using standard personal computer. First two stages represent the INHERENT design approach.

In the last stage, that we call the **parallel processor** development stage, we could use the knowledge, experiance and profit from the previous stages to built a custom made processor, optimized compiler and custom made operating system to utilize the intelligent memory module functions. With a machine designed using this NOVEL approach, we could enter the market of large scale parallel processors and compete sucessfully with the giants.

ACKNOWLEDGEMENT

The efforts of a number of individuals are reflected in this work. In particular, the author wishes to acknowledge Marjan Miletić, for the "real world" expertise, Silvo Bizjak for more thorough mathematical insight to the random interleaving problem, Estera Brajak, for fine drawings, moral support and understanding. But above all, author wishes to thank DO Iskra Delta, and prof. dr. A. P. Železnikar, for supporting the work, and making possible that the ideas, some of them born even in the early 80's could finally be presented in more formal and more organized way.

REFERENCES

[Bra86] Brajak P., Parallel Processing: Architecture of the 90's (A Survay of Recent Developments), Informatica 3, 1986, 3-10

[Bra86b] Brajak P., On techniques and conceptions of writing effective programs for large scale parallel processors, in preparation

[Ost83] Ostlund N., Hibbard P., Whiteside R., A Case Study in the Application of a Tightly Coupled Multiprocessor to Scientific Computations, in Parallel Computations, ed. G. Rodrigue, AP 1983

[Sul83] Sullivan H., Cohn L., US Pattent 254583, private communications

[Kog81] Kogge P., The Architecture of Pipelined Computers, page 43, McGraw-Hill 1981

[Hha85] Hwang K., Briggs F.A., Computer Architectures and Parallel Processing, McGraw-Hill 1985

[Sch80] Schwartz T. J., Ultracomputer, ACM Transactions on Programming Languages, 1980

[Got83] Gottlieb at al., The NYU Ultracomputer - Designing an MIMD Shared Memory Parallel Computer, Transaction on Computers, 1983

METODOLOŠKI PRISTUP DEFINISANJU INFORMACIONIH SADRŽAJA BAZE PODATAKA

INFORMATICA 1/87

Mirko Smilevski
Visoke vojnotehničke škole, Zagreb

UDK 681.327.016

U ovom radu opisan je jedan pristup definisanju informacionih sadrzaja baze podataka, zasnovan na Nijssenovoj metodologiji projektovanja baze podataka. Predlozeni postupak obuhvaca pocetne faze u razvoju informacionog sistema - analizu aktivnosti i informacionu analizu. Konceptualni model, dobijen predlozenim postupkom, osnova je za dalje projektovanje logicke strukture baze podataka. Cilj postupka je da pomogne analiticarima sistema i projektantima baza podataka u izgradnji konceptualnog modela baze podataka, a korisnicima u definisanju svojih informacionih potreba.

Kljucne reci: aktivnosti, analiza, baza podataka, informacioni sistem, informacioni tok, konceptualizacija, konceptualni model, realni sistem

A METHODOLOGICAL APPROACH TO DEFINING THE INFORMATION CONTENTS OF DATA BASE

In this paper a methodological approach to defining the information contents of data base is presented. The proposed approach deals with initial information system development phases - activity and information analysis. This method intends to help the system analysts and data base designers in conceptual development phase of data base and to help the users in defining their information needs.

Keywords: activity, analysis, conceptualization, conceptual model, data base, information flow, information system, real system

1. U V O D

Iskustva pokazuju da je projektovanje baze podataka kriticna faza u razvoju informacionog sistema [3, 4, 8, 14]. Naime, dvosmislenosti, netacnosti i nepotpunosti u specifikacijama informacionih zahteva (to je etapa na koju se zasniva konceptualno modeliranje baze podataka) prouzrokuju dalekosezne posledice, ne samo u fazi projektovanja strukture baze podataka, vec i u kasnijim fazama razvoja informacionog sistema i na njegovu upotrebljivost u celini. Uspesno projektovana baza podataka rezultat je usvojenog metodoloskog pristupa definisanju informacionih zahteva korisnika i usvojene metodologije konceptualnog modeliranja njene strukture.

U cilju definisanja jedinstvenog metodoloskog pristupa, koji bi vodio uspesnom razvoju informacionog sistema i njegove baze podataka, nacinjen je pokusaj da se sistematizuje redosled postupaka projektovanja baze podataka i razvoja informacionog sistema uopste i nazvan je imenom informaciono inzenjerstvo [6]. Informaciono inzenjerstvo, u okviru postupaka za razvoj informacionog sistema, predvidja cetiri faze projektovanja baze podataka:

(1) strategsko planiranje potreba (analiza aktivnosti),

(2) informaciona analiza (formulacija i analiza informacionih zahteva),

(3) modeliranje podataka i

(4) fizicko projektovanje baze podataka.

Kolika je paznja usmerena projektovanju baze podataka moze se oceniti i po tome sto se prve tri faze projektovanja preporucuju u samom pocetku razvoja informacionog sistema. Iz ovih razmatranja postaje takodje jasno zasto se sve vise naglasava vaznost temeljite analize realnog sistema i informacionih zahteva kao bitnih pretpostavki za razvoj uspesnog informacionog sistema i zasto se u literaturi razmatra i predlaze sve vise metoda i tehnika za obuhvatanje svih informacija koje bi sadrzavala njegova baza podataka [1, 3, 9, 11, 12]

Analiza podataka, kako se u [17] jednim imenom nazivaju informaciona analiza i konceptualno modeliranje, je proces obuhvatanja informacija i njihove transformacije u konceptualni model podataka. Analiza podataka je, prema tome, proces uzlazne (bottom-up) analize informacionih zahteva korisnika, koja rezultira u konceptualni model

Medjutim, kako se informacioni sistem gradi za realni sistem koji se obicno sastoji od vise grupa korisnika, zavisno od njegove organizacione podele, potrebno je pre analize podataka izvrsiti analizu aktivnosti realnog sistema ili, kako se u nekim izvorima naziva, strukturnu funkcionalnu analizu. To je s-vrha (top-down) analiza realnog sistema s ciljem da se otkriju njegove aktivnosti odnosno funkcije koje ce biti informaciono podrzane i informacioni tokovi kojima se (ili ce se) odabrane aktivnosti opskrbljivati trazenim informacijama.

Analiza aktivnosti rezultira u takozvanu shemu informacionih tokova realnog sistema. Time je olaksana analiza podataka koja konceptualni model gradi na osnovu analize informacionih tokova.

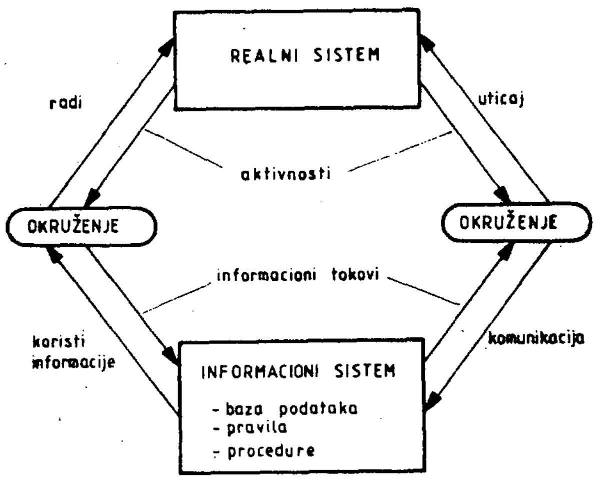
Definisanje informacionih sadrzaja baze podataka je skup postupaka kojima treba odrediti njen sadrzaj i strukturu, radi zadovoljavanja informacionih zahteva korisnika, postujuci pri tome organizacione aspekte realnog sistema, s jedne, i odnose i veze izmedju podataka, s druge strane. Taj skup postupaka u opstem slucaju podrazumeva primenu razlicitih metodologija u nedostatku jedinstvene integralne metodologije projektovanja baze podataka. Tako se, na primer, za funkcionalnu analizu koristi HIPO ili SADT metodologija, a za informacionu analizu i konceptualno modeliranje PSL/PSA ili Chen-ova E-R metodologija [16]. U ovom radu prikazujemo metodoloski pristup definisanju informacionih sadrzaja baze podataka informacionog sistema, zasnovan na Nijssenovoj metodologiji projektovanja baza podataka [13,15] koja u sebi integrise prve tri pocetne faze u razvoju informacionog sistema: analizu aktivnosti, informacionu analizu i modeliranje logicke strukture baze podataka.

Sredisnja paznja u ovom radu posvecena je analizi aktivnosti i informacionoj analizi. Pored uvoda, rad sadrzi jos cetiri poglavlja i zakljucak. Za razumevanje ovde predlozenog metodoloskog pristupa uvodimo prvo, u poglavljima 2 i 3, osnove odnosa izmedju realnog i informacionog sitema na kojima se gradi Nijssenova metodologija, osnovne pretpostavke usvojene metodologije i globalne korake u pristupu definisanju informacionih sadrzaja i konceptualnog modela baze podataka.

U cetvrtom poglavlju detaljno je razradjena pocetna faza u razvoju informacionog sistema - analiza aktivnosti, koja obuhvaca prva tri koraka metodoloskog pristupa. Peto poglavlje opisuje postupak informacione analize, koji obuhvaca preostala dva koraka metodoloskog pristupa. Rezultat ove faze je konceptualni model baze podataka, cime smo postigli postavljeni cilj. Konceptualni model, naime, upravo definise informacije koje treba sadrzavati baza podataka i koje treba ugraditi u njenu logicku strukturu.

2. ASPEKTI INFORMACIONOG SISTEMA

Nijssenova metodologija projektovanja baze podataka zasniva se na opstoj postavci informacionog sistema u odnosu na realni sistem (prikazano na slici 1).



Slika 1: Odnos i medusobni uticaj realnog sistema, informacionog sistema i okruženja

Informacioni sistem, najkrace receno, je sredstvo za informacionu podrsku realnog sistema. U ovim razmatranjima pod realnim sistemom se podrazumeva onaj deo celokupne posmatrane realnosti za koga se informacije prikupljaju, obradjuju i isporucuju. Realni sistem, u krajnjem obliku, je odredjen aktivnostima koje obavlja, njegovo okruzenje. Uz svaku aktivnost su pridruzene informacije. Neke od njih su potrebne radi izvrsenja same aktivnosti, a druge pak izviru iz aktivnosti pokazujuci stanje ili rezultate nakon zavrsetka aktivnosti. Sve te informacije obuhvata, obradjuje i isporucuje okruzenju informacioni sistem.

Iz opisanog polozaja informacionog sistema u odnosu na realni sistem mozemo usvojiti njegovu opstu arhitekturu: informacioni sistem se sastoji od baze podataka koja odrazava stanje realnog sistema u nekom vremenskom trenutku, od skupa pravila postavljenih nad podacima i od procedura za obradu podataka, sve to organizovano tako da se mogu ispuniti informacioni zahtevi okruzenja. Okruzenje realnog i pridruzenog informacionog sistema sacinjavaju ljudi koji izvrsavaju aktivnosti realnog sistema, s jedne, a koriste informacije informacionog sistema, s druge strane.

Iz prethodne diskusije nazire se osnovna ideja Nijssenove metodologije u informacionoj analizi i konceptualnom modeliranju, koja se sastoji u: odredjenju aktivnosti realnog sistema, odredjenju tokova informacija, usmeravanju informacionih tokova na pravcu korisnici-informacioni sistem, otkrivanju sadrzaja informacionih tokova, razlaganju informacionih sadrzaja na elementarne nosioce vrednosti informacija - atribute, te u strukturiranju tih atributa u model baze podataka, postujuci pripadnost atributa objektima iz realnog sistema i postojece relacije izmedju njih. Samo na ovaj nacin moze se garantovati da ce baza podataka odgovoriti na sve unapred postavljene informacione zahteve.

3. ETAPE KONCEPTUALNOG MODELIRANJA

Osnovne pretpostavke u metodolskom pristupu definisanju konceptualne strukture baze podataka o kome je ovde rec su:

(1) realni sistem, za koga se informacioni sistem gradi, sastoji se od skupa aktivnosti;

(2) informacioni sistem se sastoji od skupa funkcija koje informaciono podrzavaju aktivnosti realnog sistema i

(3) informacije u realnom sistemu teku - preko informacionog sistema - informacionim tokovima, a nosioci informacija su dokumenti razlicitih vrsta i formi (izvestaji, tabele, pisma i sl.).

Uzevsi u obzir navedene pretpostavke i aspekte na koje se gradi Nijssenova metodologija, pristup konceptualnom modeliranju baze podataka mozemo podeliti u 5 koraka:

(1) odredjenje aktivnosti realnog sistema koje ocekuju informacionu podrsku;

(2) odredjenje funkcija informacionog sistema s kojima ce informacioni sistem informaciono podrzati aktivnosti realnog sistema;

(3) odredjenje informacionih potreba korisnika, odnosno, odredjenje informacionih tokova i dokumenata - nosilaca informacija;

(4) analiza informacionih tokova i

(5) konstruisanje konceptualnog modela baze podataka.

Za prikaz shema informacionih tokova, informacionih struktura i konceptualnog modela usvojicemo graficku notaciju koja se uobicajeno koristi u vec poznatim postupcima informacione analize i konceptualnog modeliranja [5, 12, 13].

4. ODREDJENJE I ANALIZA AKTIVNOSTI

Pocetna faza u razvoju informacionog sistema i njegove baze podataka je funkcionalna analiza odnosno analiza aktivnosti realnog sistema. Odredjenje i analiza aktivnosti je s-vrha pristup realnom sistemu radi:

(1) odredjenja aktivnosti ili funkcija realnog sistema i selekcije onih aktivnosti koje treba informaciono podrzati,

(2) odredjenja ulaznih i izlaznih informacionih tokova selektiranih aktivnosti i

(3) konstruisanja globalnog modela informacionog sistema.

Pazljivim i celovitim posmatranjem realnog sistema i intervjuisanjem njegovog okruzenja na svim nivoima (od najvisih rukovodilaca do krajnjih korisnika) mozemo stvoriti opstu sliku o zadacima i aktivnostima realnog sistema i o vitalnim informacionim tokovima. Nakon toga, posebna paznja se posvecuje upravo informacionim tokovima, zato sto oni sadrze podatke od kojih treba saciniti bazu podataka.

4.1 Selekcija aktivnosti realnog sistema

Za identifikaciju i izbor aktivnosti realnog sistema moze posluziti shema organizacione strukture realnog sistema [7, 10]. Teskoce u ovom procesu javljaju se kada se informacioni sistem razvija za realni sistem koji se takodje razvija, pa se njegove aktivnosti ne mogu u potpunosti odrediti, ili kada se informacioni sistem dogradjuje zbog promena aktivnostu realnog sistema. U svakom slucaju, za uspesnu analizu aktivnosti potrebno je potpuno poznavanje realnog sistema i aktivno ucesce okruzenja. U ovoj fazi okruzenje najbolje zna koje aktivnosti treba informaciono podrzati i kakve efekte od toga ocekuje.

Odredjenje i analiza aktivnosti uobicajeno pocinje identifikacijom globalnih aktivnosti realnog sistema i globalnih funkcija informacionog sistema. Na jednom nivou, medjutim, aktivnosti i funkcije su preslozene za razumevanje, obuhvatanje informacionih tokova i konceptualno modeliranje baze podataka. Zato se one dele na manje slozene aktivnosti i funkcije koje se nazivaju zadacima ili podfunkcijama. Uporedo s tim, dokumenti kao nosioci informacija se takodje cepaju na komponente ili podskupove podataka ili na podskupove recenica. U ovom procesu, koji je nazvan dekompozicijom [12, 13], slozene funkcije se razlazu na jednostavnije sve dok ne postanu potpuno jasne tako da je moguce:

(1) detaljno opisati njihove transformacije

(2) odrediti dokumente - neposredne nosioce informacija.

4.2 Odredjenje informacionih tokova

Nakon odredjenja aktivnosti potrebno je da obuhvatimo i specificiramo informacione potrebe okruzenja za njihovo obavljanje. Usvojili smo, naime, pretpostavku da su u okruzenju uspostavljeni informacioni tokovi koji omogucavaju komuniciranje izmedju njegovih elemenata. Svaki informacioni tok u tom smislu oznacava strujanje poruka, a prikazuje komunikaciju izmedju dva elementa iz okruzenja.

Ovde cemo usvojiti jos jednu pretpostavku, a to je da poruke informacionim tokovima struje samo u jednom smeru - od izvorista ka odredistu (izvoriste i olrediste su, jasno, elementi ili delovi okruzenja). Svaki informacioni tok je na ovaj nacin odredjen smerom i sadrzajem. Smer informacionog toka pokazuje kretanje informacija od izvorista ka odredistu, a sadrzaj otkriva prirodu komunikacije izmedju njih.

Korisnici informacionog sistema cesto ne znaju kako da specificiraju svoje informacione zahteve, ne zato sto ne poznaju svoj deo aktivnosti, vec vise zbog toga sto ne nalaze zajednicki (razumljiv) jezik sa analiticarima sistema i projektantima baze podataka. Stoga je zadatak analiticara i projektanata da u ovoj fazi, sluzeci se uobicajenim jezikom, rade zajedno s korisnicima na definisanju informacionih zahteva, da postignu saglasnost o znacenju i smislu podataka i da konstruisu pravila za zastitu integriteta informacija. Oni takodje treba zajedno da razrese sve dvosmislenosti u specifikaciji informacionih sadrzaja i da informacione zahteve opisu u formi dokumenata.

4.3 Konstruisanje sheme informacionih tokova

Sledeci korak u ovoj fazi je konstruisanje globalnog modela informacionog sistema kao skup funkcija koje podrzavaju odredjene aktivnosti realnog sistema. Svaka funkcija se postavlja sa zadatkom da informaciono podrzi pridruzene aktivnosti tako da:

(1) ostvaruje komunikaciju izmedju elemenata okruzenja;

(2) obradjuje informacije sa ulaznih informacionih tokova i osvezava bazu podataka novim informacionim sadrzajima i

(3) generise potrebne informacije i izlaznim informacionim tokovima isporucuje ih okruzenju.

Tako postavljene, funkcije informacionog sistema mozemo shvatiti kao transformatore informacionih tokova.

Sa odredjenjem informacionih tokova javlja se novi problem: kako obuhvatiti sadrzaje poruka koje njima struje. Ovde cemo taj problem resiti usvajanjem pretpostavke da su nosioci informacija u informacionim tokovima dokumenti koji se mogu pojaviti u razlicitim oblicima: izvestaji, pisma, tabele i sl. Jedan deo tih dokumenata prihvacamo iz realnog sistema, drugi deo dokumenata pak formiramo u procesu odredjenja informacionih potreba okruzenja.

Dokumenti se mogu shvatiti kao skupovi podataka ili, slobodnije, kao skupovi specificno konstruisanih recenica kojima se iskazuju odredjene informacije. Uopsteno receno, kao dokumenti se mogu prihvatiti sve vrste pisanih poruka. Dokumenti kao nosioci informacija su poznati i iz sire literature, kao prirodan i siroko prihvacen nacin komuniciranja okruzenja s bazom podataka informacionog sistema [1, 2, 11].

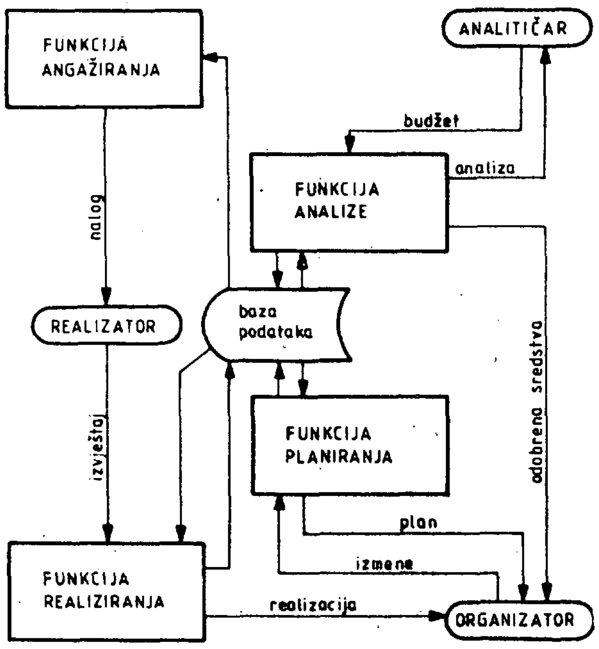
Odredjenjem i analizom aktivnosti realnog sistema mi razvijamo funkcionalni model sistema identifikacijom globalnih aktivnosti ili funkcija i njihovih odnosa. Funkcionalni model, koji se sastoji od glavnih aktivnosti realnog (i informacionog sistema), informacionih tokova i veza izmedju funkcija i korisnika, moze se opisati pomocu takozvane sheme informacionih tokova [13], kao na slici 2. Ova shema sadrzi:

(1) odabrane aktivnosti realnog sistema i pridruzene funkcije informacionog sistema - prikazane pravokutnicima,

(2) okruzenje - prikazano elipsama i

(3) dokumente koji kao nosioci informacija kolaju informacionim tokovima, posredstvom funkcija, izmedju okruzenja i baze podataka.

Informacioni tokovi prikazani su strelicama, a baza podataka poluelipsom. Dokumenti su ovde jedini informacioni mediji za transfer informacija. Strelice pokazuju smerove kretanja dokumenata informacionim tokovima.



Slika 2. Globalna shema informacionih tokova sistema upravljanja projektom

Shema informacionih tokova predstavlja mrezni model realnog i pridruzenog informacionog sistema jer oba sistema prikazuje preko njihovih komponenti i informacionih veza izmedju komponenti. On je dinamicki model, jer prikazuje kretanje informacija kroz komponente oba sistema. On je osnova za sledece dve faze u razvoju informacionog sistema: analizu podataka (analizu informacionih tokova) a analizu procesa (analizu funkcija informacionog sistema).

Faza odredjenja i analize aktivnosti je kljucna za razumevanje realnog sistema i za odredjenje njegovih informacionih potreba. Informacione potrebe, odredjene ovom analizom i sintetizovane u obliku dokumenata, bice nam polazna osnova za projektovanje konceptualnog modela baze podataka.

5. INFORMACIONA ANALIZA

Proces specificiranja dopustenih informacionih sadrzaja baze podataka smatra se nezaobilaznom fazom u razvoju informacionog sistema [4,14]. Ova se faza, u okviru informacionog inzenjerstva, razvila do nivoa discipline i nazvana je imenom "informaciona analiza" [4,12,13]. Informaciona analiza se bavi problemom potpunog i nedvosmislenog prikaza semanticke strukture informacija koje okruzenje zahteva od informacionog sistema.

Informacionoj analizi se, u razvoju informacionog sistema, pristupa nakon analize aktivnosti, odnosno nakon odredjenja informacionih tokova i dokumenata - nosilaca informacija. Analizom dokumenata otkrivaju se strukture informacija koje struje informacionim tokovima. Upravo te informacije su osnova za odredjenje informacionih sadrzaja baze podataka.
Cilj ove faze je prenos znanja o svakoj aktivnosti, prikupljenog u prethodnoj fazi, u formalni prikaz koji se naziva konceptualna shema (informacioni model). U fazi konceptualnog modeliranja podataka, nasa paznja je usmerena samo na konceptualne objekte koji se prate iz realnog sistema, na njihova svojstva, opisana njihovim atributima, i na relacije koje su uspostavljene izmedju objekata. Fizikalna ili racunarska reprezentacija objekata je irelevantna na ovom mestu.

Da bi prikazali konceptualne objekte potreban nam je racunarsko-nezavisan semanticki model koji:

- moze obuhvatiti znacenje konceptualnih objekata,
- ima dobro definisana pravila interpretiranja tako da i drugi ljudi, osim projektanata i korisnika sistema, mogu razumeti znacenje podataka, i
- ima dobru dokumentacionu podrsku (tekstualnu i graficku) i operatore visokog nivoa za njegovu transformaciju.

5.1 Konceptualni model

Rezultat informacione analize naziva se informacioni ili konceptualni model [3,11,12,13,14] - sredstvo za komuniciranje izmedju okruzenja i projektanta baze podataka. Informacioni model je binarni relacioni model [13], u kojem se razlikuju sledeci apstraktni pojmovi: objekti, atributi, relacije i ogranicenja. Svaki apstraktni pojam je u informacionom modelu jednoznacno identificiran imenom (ime objekta, ime atributa, ime relacije i ime ogranicenja). Svaki od njih moze takodje imati pridruzeni skup sinonima i opis u prirodnom jeziku.

Objekti su realni ili apstraktni pojmovi koji postoje u realnom sistemu i poseduju svojstva pomocu kojih se mogu medjusobno razlikovati, a prikazuju klase elemenata iz realnog sistema za koga se projektuje baza podataka i informacioni sistem. Tako, na primer, objekti: projekt, realizator, organizator i sl., postoje u realnom sistemu i poseduju svojstva pomocu kojih se medjusobno razlikuju. Svaki element klase elemenata naziva se pojava objekta.

Atributi su svojstva koja pripadaju objektima i omogucavaju im da se identifikuju, karakterisu, klasifikuju i da se izmedju objekata uspostave odredjeni odnosi. Svaki atribut je matematicka funkcija: domena je skup pojava objekata, a kodomena je skup vrednosti. Svaka elementarna vrednost atributa naziva se pojava atributa. Atributi su, na primer: ime projekta, broj izvestaja i sl. Svakoj pojavi objekata se pomocu atributa pridruzuju vrednosti iz odredjenih skupova vrednosti.

Relacije prikazuju odnos izmedju razlicitih klasa (skupova) objekata i izmedju objekata i atributa. Relacije su binarne, a kod njihovog uspostavljanja moguce su samo dve kombinacije odnosa: mzmedju dva objekta i izmedju atributa i objekta. Relacije izmedju dva atributa nemaju smisla.

Ogranicenja su pravila ukljucena u konceptualni model koja opisuju ponasanje realnog sistema. Svrha im je sprecavanje protivrecnosti izmedju sadrzaja baze podataka i aktivnosti u realnom sistemu. Ona omogucavaju precizniji opis odnosa izmedju objekata, atributa i relacija u realnom sistemu, prenoseci te odnose na nivo konceptualnog modela.

Osnovno ogranicenje koje se postavlja nad ulogama koje odredjeni objekti/atributi igraju u relacijama sa drugim objektima/atributima je jedinstvenost. Tako, na primer, u recenici "RADNIK ima IME-RADNIKA" uloga "ima" je jedinstvena jer svakom radniku pripada samo jedno ime iz skupa imena. Pri tome u opstem slucaju obrnuto ne vazi, jer jedno isto ime moze pripadati vise nego jednom radniku. U tom smislu je ogranicenje jedinstvenosti preslikavanje tipa 1:n.

Svaki objekt, medjutim, mora imati jedinstvenu identifikaciju. Ona se ostvaruje s jednim ili vise atributa, tako da je preslikavanje izmedju skupa objekata i Kartezijevog produkta skupova vrednosti atributa jednako 1:1. Skup od jednog ili vise atributa, pridruzen odredjenom skupu objekata, koji jednoznacno identifikuje njegove elemente zove se kljuc objekta. Ako uopsteno postoji vise skupova atributa koji jednoznacno odredjuju elemente skupa objekata, onda se jedan od tih skupova atributa bira za kljuc i naziva se primarnim kljucem.

U konceptualnom modeliranju znacajno je jos jedno ogranicenje, koje se moze postaviti nad skupovima objekata, i naziva se ogranicenje podskupa. Skup objekata o1 je podskup skupa objekata o2 ako se svi elementi iz o1 pojavljuju u o2, dok obrnuto u opstem slucaju ne vazi.

Ogranicenjima se u konceptualnom modelu moze naglasiti da eventualno usvojeni sistem ili druga specificna programska podrska za upravljanje bazom podataka treba da odrzava samo dozvoljena stanja baze podataka. Zato se ogranicenja cesto nazivaju pravilima za ocuvanje integriteta baze podataka.

5.2 Odredjenje elemenata konceptualnog modela

Osnovni problem informacione analize je kako iz dokumenata izvuci elemente konceptualnog modela. I u ovoj fazi, isto kao u prethodnoj, naglasava se uloga okruzenja u otkrivanju trazenih elemenata. U dokumentima, koji sluze kao medij za transfer informacija, najcesce dominiraju samo vrednosti atributa, dok se ostali elementi (objekti, relacije, ogranicenja) neposredno ne pominju. Duznost projektanta je da, u saradnji sa korisnicima, analizom dokumenata i poznavanjem realnog sistema izvuce pripadnost atributa objektima i odnose izmedju njih.

Ako se prihvati tvrdnja da se dokumenti uopsteno sastoje od skupa recenica, kao specijalni slucaj recenica u prirodnom jeziku, onda se njihovom analizom mogu otkriti trazeni elementi. Neki opsti algoritam za otkrivanje ovih elemenata ne postoji, ali zato mozemo usvojiti dva pravila pomocu kojih je to otkrivanje znatno olaksano, premda i dalje treba racunati na saradnju korisnika.

Prvo pravilo je otkrivanje imenica, glagola i prideva u recenicama dokumenata, na osnovu kojih se elementi konceptualnog modela otkrivaju tako sto se imenice prihvacaju kao objekti, glagoli kao relacije, a pridevi kao atributi. Prednost ove tehnike je sto objekti, atributi i relacije otkriveni na ovaj nacin imaju smisao za korisnika zato sto ukazuju na pojmove iz realnog sveta. Nedostatak je, medjutim, sto je ovaj proces imenovanja veoma subjektivan i razliciti projektanti mogu dati razlicita imena istim objektima, atributima ili relacijama.

Drugo pravilo, na koje se upravo oslanja Nijssenova metodologija konceptualnog projektovanja, naziva se "konceptualizacija" [15]. To je proces u kome se recenice iz dokumenata transformisu u elementarne recenice, a elementarne recenice u objekt-uloga parove. Naime, sve ono sto moze uci ili se vec nalazi u informacioni sistem moze biti potpuno, formalno i jednostavno opisano specifikacijom tipova objekata, specifikacijom uloga koje tipovi objekata mogu igrati u elementarnim recenicama i specifikacijom ogranicenja koja su nametnuta odredjenim tipovima objekata u odredjenim ulogama.

Formalno, recenica se moze prikazati kao skup jedne ili vise elementarnih recenica, koje u smislu konceptualizacije predstavljaju binarne relacije:

```
R = { <(OBJ-i, UL-j), (OBJ-k/ATR-l, UL-m)>. . . }

gde su

     OBJ     - objekt
     ATR     - atribut
     UL      - uloga

     <. . .> - binarna relacija
     (. . .) - objekt-uloga/atribut-uloga
               par
     /       - operator "ili"

     0 < i, j, k, l, m < n
```

Formalni opis recenice pokazuje da je elementarna recenica binarna relacija izmedju objekta i atributa ili izmedju dva objekta, odnosno, da se u elementarnoj recenici mora pojaviti najmanje jedan objekt

Ogranicenja se postavljaju nad ulogama koje objekti igraju u relacijama s drugim objektima ili atributima, a neka ogranicenja se namecu nad skupovima objekata Ona su semanticka znacajka konceptualnog modela i cesto se ne mogu prepoznati neposredno iz dokumenata i njihovih recenica, vec tek iz odnosa objekata sa drugim objektima ili atributima

5.3 Graficka reprezentacija konceptualnog modela

U fazi informacione analize neophodno je usvojiti prikladnu notacionu tehniku ili jezik za opis konceptualnog modela, koji bi bio razumljiv i za projektanta baze podataka i za okruzenje i koji bi olaksao komuniciranje izmedju njih.

Razmotrimo, na primer, sledecu recenicu:

RADNIK sa brojem 123456

radi na

PROJEKTU sa brojem 654321.

Ovo je slozena recenica koja se, upotrebom pravila za analizu recenice i odredjenje elemenata konceptualnog modela, opisanih u tacki 5.2, moze rastaviti na cak 6 elementarnih recenica:

(1) RADNIK ima BROJ-RADNIKA
(2) BROJ-RADNIKA pripada RADNIKU
(3) PROJEKT ima BROJ-PROJEKTA
(4) BROJ-PROJEKTA pripada PROJEKTU
(5) RADNIK radi na PROJEKTU
(6) PROJEKT izvrsava RADNIK.

K tome jos treba dodati sledeca ogranicenja:

(1) RADNIK ima samo jedan BROJ-RADNIKA
(2) BROJ-RADNIKA pripada samo jednom RADNIKU
(3) PROJEKT ima samo jedan BROJ-PROJEKTA
(4) BROJ-PROJEKTA pripada samo jednom PROJEKTU
(5) RADNIK radi na vise PROJEKATA
(6) PROJEKT izvrsava vise RADNIKA.

Daljom analizom recenica i ogranicenja dolazimo do sledecih zakljucaka:

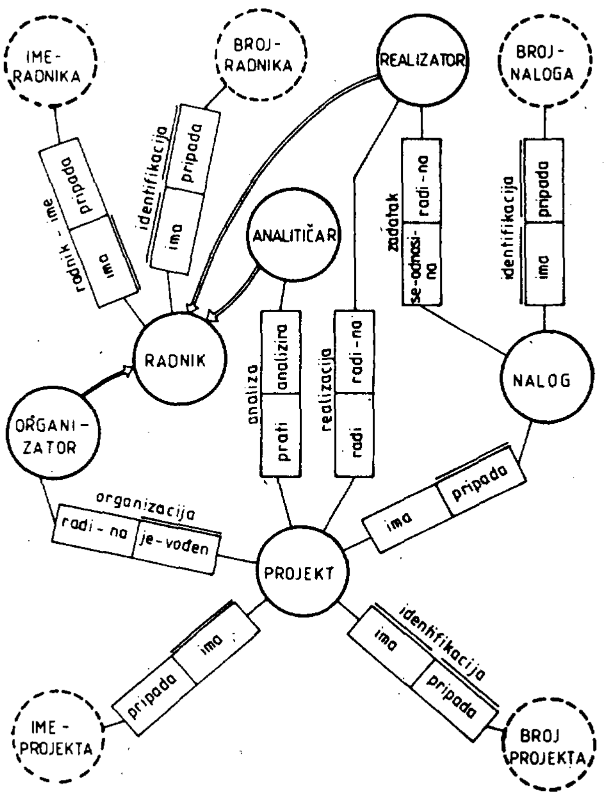
(1) izmedju RADNIK i PROJEKT uspostavljena je relacija, koju mozemo nazvati ZADATAK
(2) izmedju RADNIK i BROJ-RADNIKA uspostavljena je relacija, koju mozemo nazvati KLJUC-RADNIKA
(3) izmedju PROJEKT i BROJ-PROJEKTA uspostavljena je relacija, koju mozemo nazvati KLJUC-PROJEKTA

I, na kraju, zakljucujemo da su u slozenoj recenici RADNIK i PROJEKT objekti, a BROJ-RADNIKA i BROJ-PROJEKTA atributi koji pripadaju odgovarajucim objektima.

Konceptualni model sacinjen je od prikazanih rezultata analize recenice koji su predstavljeni u tekstualnoj formi. Iz primera se, medjutim, vidi koliko je takav opis kompleksan i nedovoljno razumljiv. Zato je u opisanom pristupu usvojena graficka reprezentacija konceptualnog modela [5,13], kao na sl. 3.

Usvojena notacija omogucava jasan i sazet prikaz objekata (puni krugovi), atributa (iscrtkani krugovi) i relacija (dvodelni pravougaonici). Uloge, koje odredjeni objekti igraju u relacijama sa drugim objektima ili atributima, prikazane su u odgovarajucim delovima pravougaonika koji pripadaju odredjenim relacijama.

Ogranicenja jedinstvenosti i jednoznacne identifikacije prikazane su crticama koje se postavljaju iznad uloga koje objekti/atributi igraju u odgovarajucim relacijama. Tako je jednoznacna identifikacija nekog objekta prikazana sa po jednom crticom iznad obe uloge u relaciji sa drugim objektom/atributom. Relacija 1:n prikazana je crticom iznad uloge objekta/atributa koja je jedinstvena. Na primer, u relaciji RADNIK-IME, RADNIK ima samo jedno IME, pa je crtica postavljena iznad uloge "ima". IME, medjutim, moze pripadati vise nego jednom RADNIKU, pa zbog toga iznad uloge "pripada" nije postavljeno ogranicenje jedinstvenosti. Odgovarajuce tome, u relaciji n:m nema jedinstvenosti, kao sto je to na primer relacija ZADATAK, pa se iznad uloga objekata/atributa ne postavljaju crtice.



Slika 3. Grafički prikaz dela konceptualnog modela sistema upravljanja projektom

Ogranicenje podskupa se, po definiciji, postavlja izmedju objekata. U grafickoj notaciji se klasa objekata 01 kao podskup klase objekata 02 oznacava tako da se od 01 prema 02 nacrta dvostruka crta sa strelicom prema 02. Tako je, na primer, REALIZATOR podskup skupa RADNIK, pa su u grafickom prikazu na slici 3 odgovarajuci objekti povezani dvostrukom crtom i strelicom prema objektu RADNIK.

6. ZAKLJUCAK

Projektovanje informacionog sistema je slozen proces koji traje odredjeno vreme i zahteva odgovoran i metodoloski pristup, s jedne strane, i aktivno ucesce velikog broja strucnjaka (analiticara, projektanata, programera i dr.) i, jasno, korisnika, i to u svim fazama projektovanja, s druge strane. Premda jos uvek ne postoji jedinstven metodoloski pristup koji bi obuhvatio celokupan proces razvoja i projektovanja informacionog sistema, nacinjen je pokusaj da se sistematizuje redosled postupaka koji vodi uspesnom razvoju informacionog sistema i nazvan je imenom "informaciono inzenjerstvo". U poslednje vreme, a posebno zbog narastanja primene sistema za upravljanje bazama podataka, razvoju metoda, formalnih jezika i alata za podrsku pojedinih faza u razvoju informacionih sistema poklanja se sve veca paznja.

Baza podataka je kljuc uspesnosti informacionog sistema [5,6]. Struktura baze podataka je odraz informacionih potreba okruzenja, stoga je jasno zasto se u njenom definisanju i projektovanju pocinje od realnog sistema i zasto se nastoji da se okruzenje aktivno ukljuci u njenom razvoju, kao i u svim fazama razvoja informacionog sistema uopste [6,11,12].

Analiza informacionih zahteva je realno gledano polazna osnova za uspesno projektovanje informacionog sistema. Jedna od faza u procesu projektovanja baze podataka koja je od najveceg zanimanja u ovom radu je obrada informacionih zahteva i konstrukcija konceptualnog modela koji specificira te zahteve.

Osnovna pretpostavka u ovom pristupu projektovanja informacionog modela je da se konceptualno modeliranje moze zasnovati na to kako informacije teku izmedju sistema i njegovog okruzenja i izmedju komponenti sistema. Mi posmatramo konceptualno modeliranje kao tehniku za specificiranje objekata i relacija, u formalnom jeziku, koja doprinosi razumevanju informacionog modela i od strane korisnika i od strane projektanata.

Predmet analize informacionih zahteva je razumevanje aktualnog stanja realnog sistema i skupljanje svih informacionih zahteva. Informacioni zahtevi ce nakon toga biti upotrebljeni u konceptualnom modeliranju podataka za ugradnju kod generisanja konceptualnog modela. U fazi modeliranja sva paznja projektanta je usmerena na karakteristike objekata i relacije medju njima. Naravno, na ovom nivou se ne razmislja o nacinu na koji ce ti objekti biti prikazani na racunaru.

Opisani metodoloski pristup definisanju informacionih sadrzaja baze podataka upravo podrzava takva nastojanja. Osnova ovog pristupa sastoji se u odredjenju aktivnosti realnog sistema koje treba informaciono podrzati, odredjenju tokova informacija, usmeravanju informacionih tokova na pravcu okruzenje - informacioni sistem, otkrivanju sadrzaja informacionih tokova, razlaganju informacionih sadrzaja na objekte, atribute, relacije i ogranicenja, te u strukturiranju tih elemenata u konceptualni model, postujuci pripadnost atributa objektima iz realnog sistema i postojece odnose (relacije i ogranicenja) izmedju njih.

Konceptualni model baze podataka, dobijen opisanim pristupom, osnova je za dalje projektovanje logickog modela baze podataka.

Literatura:

1. C. Batini, B. Demo, A Di Leva A methodology for conceptual design of office data bases. Information systems 9(3/4) 1984. (251-263)
2. C. Holsapple, S Sher A Whinston A consulting system for data base design. Information systems 7(3), 1982. (281-296)
3. N. Leveson, A. Wasserman, D Berry BASIS A behavioral approach to the specification of information systems. Information systems 8(1), 1983. (15-27)
4. M. Lundeberg: Information systems development. Prentice-Hall, 1981
5. J. Martin: Computer data-base organization Prentice-Hall, 1975
6. J. Martin, C McClure Software maintnance. Prentice-Hall, 1983
7. K. Mensah: Identifying subsystems ininformation systems analysis Information systems 9(2), 1984, (181-190)
8. A. Olive: Information derivability analysis in logical information systems Communications of the ACM 26(11) 1983, (933-938)
9. A. Olive, F. Saltor Formal verification of information derivability in databases using precedence analysis Information systems 7(3), 1982, (209-215)
10. G.-C. Roman: A taxonomy of current issues in requirements engineering Computer 18(4), 1985, (14-23)
11. N. Roussopoulos, R T Yeh An adaptable methodology for database design Computer 17(5), 1984, (64-80)
12. G Verheijen, J Bekkum NIAM. An information analysis method. Accepted for the IFIP Conference "Comparative Review of Information System Design Methodologies". North Holland, 1982.
13. D. Vermeir, G M. Nijssen: A procedure to define the object type structure of a conceptual schema. Information systems 7(4), 1982, (329-336)
14. M. Vetter, R. Maddison Database design methodology. Prentice-Hall, 1981
15. G. M. Nijssen, Ed : Architecture and Models in Data Base Management Systems. North Holland, 1977
16. S. Bing Yao, Ed: Principles of Database Design, vol I. Prentice-Hall, 1985.
17. R. Perkinson: Data Analysis the Key to Data Base Design North Holland, 1984

THE REVIEW OF SOME DATA FLOW COMPUTER ARCHITECTURES

INFORMATICA 1/87

Jurij Šilc and Borut Robič
Jožef Stefan Institute
Department of Computer Science and Informatics
Ljubljana

UDK 681.32.02

The article reviews some selected data flow computer arhitectures. All the architectures are designed for VLSI implementation to provide large throughput, low power consumption, and reduced size and weight. While some are in the phase of simulation and VLSI chip floor-plan contruction the others already exhibit real VLSI implementation.

PREGLED IZBRANIH PODATKOVNO PRETOKOVNIH RAČUNALNIŠKIH ARHITEKTUR - V članku podajava pregled izbranih podatkovno pretokovnih računalniških arhitektur. Nekatere arhitekure so bodisi v fazi simuliranja oziroma izdelovanja logičnih načrtov za VLSI vezja, druge pa so že implementirane v VLSI tehnologiji.

Introduction

In spite of the conceptual break with previous computers, the hardware of the fifth generation computers will be based on VLSI of semiconductor components. Yet it is to be expected, that the hardware of each type of the fifth generation computer will be much more closely tailored to the application area than it is the case at present.

For a number of reasons, one of the most promising architectural models is data flow architecture. It is flexible and extensible, it has the potential for very high data throughputs, and it reflects, at hardware level, the inherent parallelism of the processing. Thus, the potential realm of use includes problem solving & inference machine and intelligent interface machine as it was proposed in the JIPDEC project for fifth generation computer systems.

The presence of some real data flow computers indicates that the state of the art in data flow computing has already passed initial, purely academic discussions.

In the article we review some existing data flow computers from the architectural view point. The presentation is not intended to be thorough. Instead, we concentrate on similarities and differences among the selected architectures.

Manchester data flow computer

The machine organization of the **Manchester data flow computer** [Gurd-85] is a packet communication organization with token matching [Robič-86] and is shown in Fig.1. The basic structure is a ring of four modules connected to a host system via an **I/O switch** module. The modules operate independently in a pipelined fashion with packets transferred at a maximum rate of 4.37 M packets/second. Packets destined for the same instruction are paired together in the **matching unit**. This has limited storage capacity, so that an **overflow unit** is required for programs with large data sets. Paired packets and those destined for unary instructions, fetch the appropriate instruction from the **instruction store**, which contains the machine-code for the executing program. The instruction is forwarded together with its input data to the **processing unit**, where it is executed. Output packets are eventually produced and transmitted back to toward to the matching unit to enable subsequent instructions. The return path passes through the I/O switch module, which connects the system to a host processor, and to the **token queue**, which is FIFO buffer for smoothing out an even rates of generation and consumption of packets.

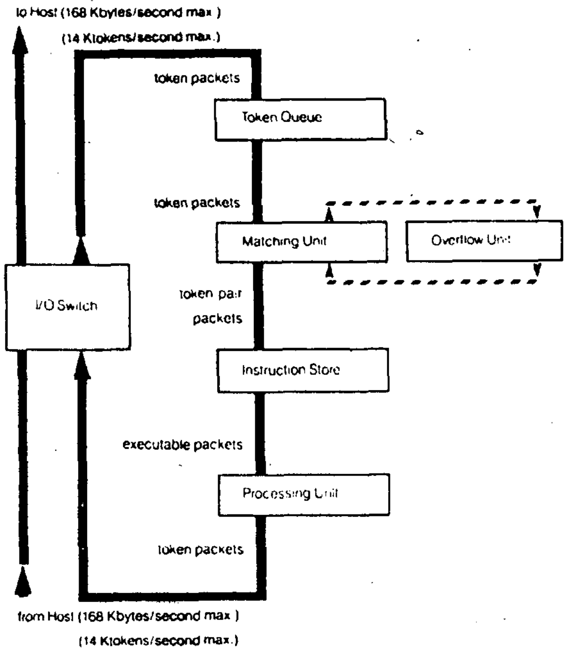


Fig.1. Manchester dataflow system structure.

The I/O switch module gives priority to input from the ring and selects the output route by performing a decode of certain marker bits. It is organized as a simple two-by-two common bus switch.

The token queue comprises three pipeline buffer registers and a circular buffer memory. The later has a capacity of 32K packets, each beeing 96-bit wide.

The matching unit is based on a 1.25 MW pseudoassociative memory with six pipeline registers in the main ring and two buffers interfacing with the overflow unit. The memory is used to store matched packets while awaiting their partners. Its associative operation is achieved by accessing a parallel store using an appropriate hash function. Recall, that packets destined for unary instructions do not need to match with partners; instead, they pass straight through the unit. The overflow unit handles packets that cannot be placed in the parallel hash table becouse they encounter a full hash entry. Overflow packets are stored in linked lists in the overflow unit, which contains a microcoded processor together with data and pointer memories.

The instruction store comprises two pipeline buffer registers, a segment lookup table, and a random-access instruction store to hold the program. The segment field of the incoming packet is used to access the instruction from the store. The instruction is 70 bits wide. The instruction is combined with a destination field and the data field of the incoming packet and is sent to the perocessing unit as a 166-bit executable instruction packet.

The processing unit comprises five pipelined buffer registers, a special purpose preprocessor, and a parallel array of up to 20 homogeneous microcoded function units with local buffer registers and common buses for input and output. A small number of instructions are executed in the preprocessor but the majority are passed into one of the function units via the distribution bus. Each function unit contains a microcoded bit-slice processor with input and output buffering, 51 internal registers, and 4 KW of writable microcode memory. Instructions are execued independently in their allotted function unit, and the output is merged onto the arbitration bus and thence out of the processing unit toward the I/O switch.

In the Manchester architecture a harware hashing scheme is used to simulate the associative memory which turns out to be less exepencive. Unfortunately, this scheme does not produce very good results in terms of waiting time. In order to reduce the waiting time, a multiple matching units scheme is incorporated in the **EXMAN** - EXtended MANchester data flow computer [Patnaik-86].

MIT data flow computer

The **MIT data flow computer** bases on a static concept of data flow architecture [Dennis-80] in which the instructions of machine level program are loaded into specific mermory location in the machine before computation begins, and only one instance of an instruction is active at a time.

Instructions are held in the local memories of the **processing element** PE. Each instruction includes an ooeration code, spaces for operand values, and destination fields that specify where resuls should be sent. Each PE is equipped to recognise which of the instructions it holds have been enabled for execution by arrival of needed operand values. If an enabled instruction calls for a scalar arithmetic operation, the instruction, including its operands, is sent to a **functional unit** FU capable of performing that operation. The **array memory units** AM are provided to hold arrays of data

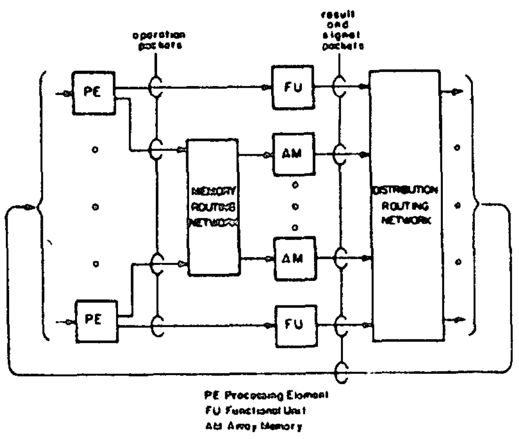


Fig.2. The MIT data flow computer.

making up the data base of computation, and are accessible through the **memory routing network**. Instruction execution in FU or AM yields result packets each of which consists of a data value and a destination field that specifies the target instruction for the result packet. The result packets are sent to PEs that hold the target instruction through the **distribution routing network**. Other instructions, such as those calling for duplicate data values, for boolean operations, and for simple tests, are performed within the PE.

The current status of the MIT data flow project is that hardware for the above computer architecture is under development. For this sake, a data flow engineering model [Dennis-83] consisting of eight processing units coupled by a paket communiation network built of two-by-two routers is designed for emulating the described architecture.

Data flow computer SIGMA-1

SIGMA-1 is a data flow multiprocessor system for scientific computations [Shimada-86]. The configuration of the system is depicted in Fig.3. Four processing elements PE and four structure elements SE are connected by local network and called a group. Groups are connected by global network. The purpose of using hierarchical network is to execute programs efficiently by utilizing principles of locality.

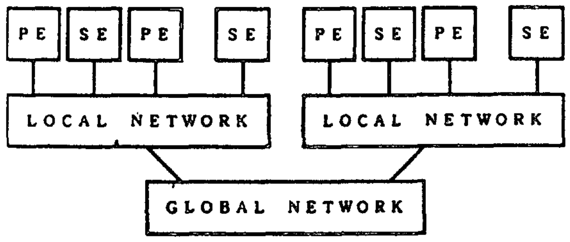


Fig.3. Global configuration of the SIGMA-1.

The **processing element** consists of five units, with the units organized as a two-stage pipeline as shown in Fig.4. PE executes all instructions except those that manipulate structure memory. The **buffer unit** (8 KW of 40 bits) is an interface between the network and the PE. The length of the incoming packet is 88 bits. It is divided into two parts (top

48-bit and bottom 40-bit) and passes through the network as consecutive parts. The most signifficant 8 bits are a network address, next 40 bits are tag, and the remaining 40 bits are data type and value. When there is no waiting packet in the buffer memory and the next units are not dealing with an other packet, the incoming packet bypasses this unit and proceeds to the subsequent units. The **fetch unit** is 16 KW, 40-bit-wide program memory. The link number carried by an incoming packet is used to access the address of an instruction to be fetched. The operation field of the fetched instruction indicates an operation code and is sent to the execution unit. The destination field of the fetched instruction gives addresses of destination instructions (waiting for the result) and is sent to the destination unit. The matching flags from the destination field are sent to the matching unit. The **matching unit** is a 16KW, 80-bit-wide associative memory to find a partner packet of an incoming packet. The matching-flag indicates whether the operation is unary or binary. When it is a unary, the incoming data packet is bypassed to the execution unit. If the instruction is binary operation the incoming packet is stored in the associative memory if it is a first arrived packet of the two operands. Otherwise, the matching unit succeeds to find a partner packet in the matching memory and sends both data of packet pair to the execution unit. The **execution unit** consists of an ALU, a shift unit and a floating point arithmetic unit. The word length is 32 bits. It receives an operation code from the fetch unit and data from matching unit. The **destination unit** makes output packets by combining the destination addresses and results from the execution unit.

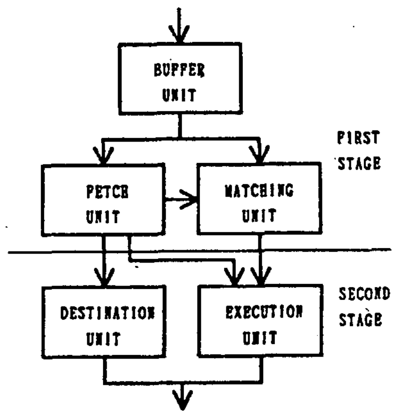


Fig.4. Structure of the processing element.

The **structure element** comprises 64KW, 35-bit-wide memory to store array data and a control unit to manage free memory words and waiting queues. When an array is declared in a program, a contiguous area corresponding to the array size is allocated in the structure memory. Once the word is allocated, the used bit of each word in the area is turned on. Each word has two other special bits. The presence bit means that data has already been written in the word. The waiting bit indicates that at least one read request packet exists in the waiting queue. When data is written in the word the data is sent to the instructions indicated by the waiting packets.

A 10 by 10 crossbar is adopted for a **local network**. This is realized by bit slice chip. The **global network** is organized as a multistage network [Mavriĉ-86]. The same crossbar chip is used for the module at each stage of the global network.

Judging from the performance of 1.35 MIPS of the prototype hardware for the benchmark programs, the performance of the next version of a processor with CMOS LSI technology should be about 1.9 MIPS.

μPD7281-based data flow architecture

The **μPD7281** is the first VLSI device on silicon using data flow architecture [NEC-85]. The μPD7281 image pipeline processor is designed to be used as a peripheral processor with a mini- or microcomputer serving as the host. Fig.5 shows a general system configuration example of which four μPD7281s are used connected to the memory in a ring shape with the entire ring interfacing with the host computer via a standard bus.

For the above architecture, NEC is developing a support chip **MAGIC**, Memory Access and General bus Interface Chip. It handles all packet flow between the μPD7281s, the image memory, and the host processor.

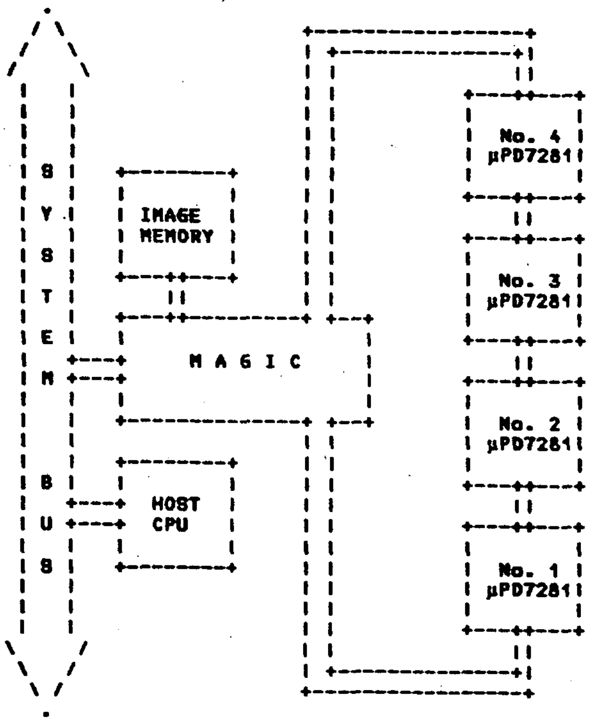


Fig.5. μPD7281-based data flow architecture.

The μPD7281 uses an internal circular pipeline and the powerful instruction set [Silc-86] to allow high end immage processing. A data flow architecture allows the processor to maximize efficiency in a variety of multiprocessing applications. As shown in the block diagram in Fig.6, the μPD7281 is formed by ten functional blocks: the **input controller** IC, the **link table** LT, the **function table** FT, the **address generator and flow controller** AG&FC, the **data memory** DM, the **queue** Q, the **processing unit** PU, the **output queue** OQ, the **output controller** OC, and the **refresh controller** RC.

Before any processing occurs, the host processor down-loads the object code into the LT and FT by using specially formated input packets. The contents of the LT and FT are closely related to a data flow graph. The arcs represent the entries in the LT while the nodes represent the entries in the FT.

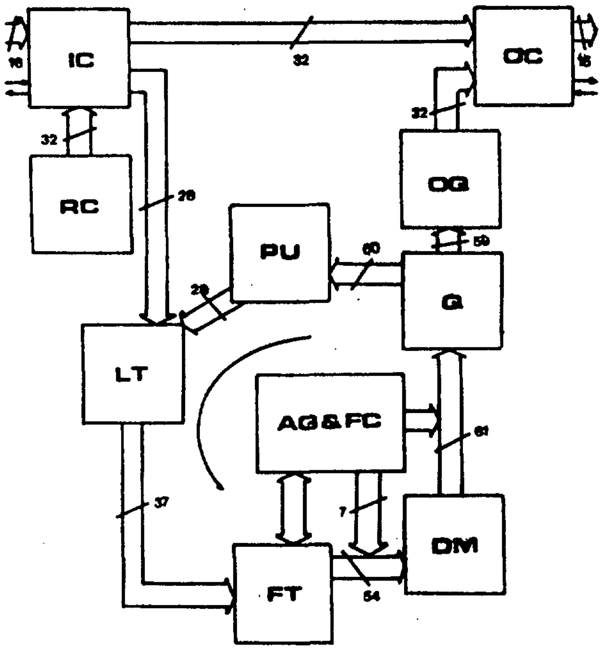


Fig.6. Block diagram of the µPD7281.

When a data packet enters the µPD7281, it fetches from the LT the address of the instruction in FT, waiting for the incoming data. After the destination instruction has been fetched, the AGFC unit determines whether the instruction is unary or binary. If it is unary, the operation packet, consisting of the instruction and the data is composed and sent via Q to the PU. If it is binary, the AGFC stores the incoming data to the DM if it is the first arrived operand for the instruction. Otherwise, it fetches the first operand from the DM and sends it together with the incoming packet and the instruction to PU via Q. The result packet from PU can either be sent out of the µPD7281 (via LT, Q, OQ, and OC) or can be used for further execution of the program graph in the same processor.

The applications of the µPD7281-based data flow architecture include digital image restoration, data compression, and enhancement, pattern recognition, radar and sonar processing, FFTs, digital filtering, speech processing, and numeric proessing.

DFSP - a data flow signal processor architecture

A block diagram of the DFSP architecture [Hartimo-86] is shown in Fig.7. A bank of processing elements constitutes the execution unit, which performs the actual digital signal processing computations and I/O operations. Other parts of the architecture form a control section, which is essentially a data flow instruction execution pipeline. In orther to increase communication bandwidth, data transfers are separated physically from execution control using a double bus architecture. Signal data is transferred via the shaded buses of the figure. The unshaded buses are used for operation and results packets, which do not contain operand and result values, respectively.

A host computer is required to load the application programs of the DFSP. Programs are coded as separate high level operations, which are copied into the local memories of the processing elements (PEs). The PEs are allowed to be functionally nonidentical in order to capitalize the existing high speed architectures for fixed signal processing algorithms. Frequently used operations may be executed in dedicated PEs having the appropriate hardware structure. The I/O functions take place in special PEs called I/O processors. This is convenient in signal processing applications, becouse signal sources and sinks tend to introduce specialized requirements.

The control section schedules instructions for the PEs using fixed-format messages. Recall, that initially all the information about the data flow graph of the application program resides in the local memories of the PEs. Each result packet carries the neccessary part of this information to the control section where it is temporarily stored in the activity store until the destination operation may be scheduled for the execution. The execution is performed by sending an operation packet to one of the PEs.

The **activity store** contains the activity templates of those operations which have received at least one of the operands, but which are not scheduled for the execution. Conceptually, the activity store contains a representation of the active part of the data flow graph.

The contains of the result packet are used by the **update unit** for locating the activity template (of the destination operation). It also contains the the address of a block in the

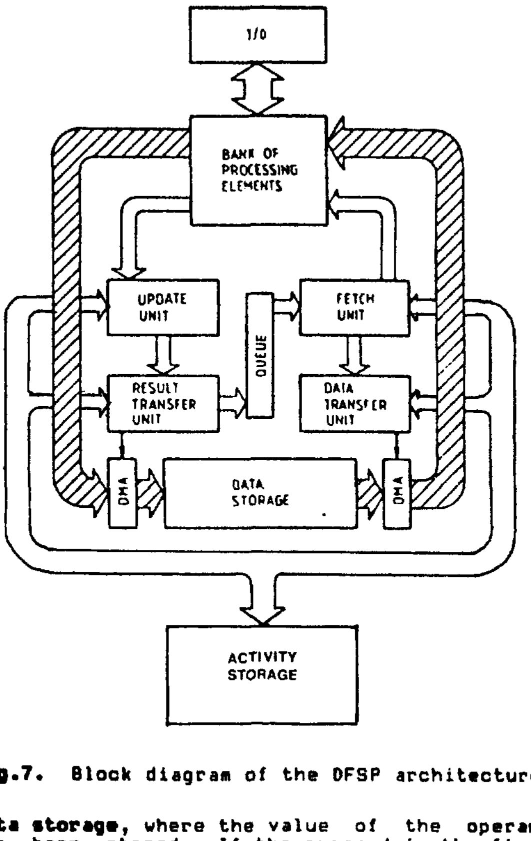


Fig.7. Block diagram of the DFSP architecture.

data storage, where the value of the operand has been stored. If the operand is the first one, the update unit creates a new activity template and stores the result packet into it. Otherwise, the result packet is stored in the located activity template. Finally, it puts a transfer command into the result queue.

After the **result transfer unit** detects the command from the queue it transfers the updated activity template. Each activity template contains a TRIGGER field whose value indicates the number of the arrived operands. The result transfer unit decrements the TRIGGER field of the destination template and checks for the resulting value. If TRIGGER equals zero the template address is put into the queue, since the operation is exectable.

After the **fetch unit** gets a template adress from the queue, sends the operation packet to an idle PE, and puts a data transfer command into the data queue.

Finally, the **data transfer unit** initiates the transfer of the operand data block from the data store to the PE.

The performance of the DFSP architecture was evaluated on a deterministic discrete event simulator. The update unit has been the major bottleneck in the simulated control section. However, considerably uniform utilization of the (four) processors has generally been achieved. Also, a VLSI implementation of the control section is under development.

HDFM - Huges data flow multiprocessor

The **HDFM** is a proposed high performance, foult tolerant, high level language programmable processor targeted for embedded signal and data processing applications [Gaudiot-85]. The HDFM consists of one to hundreds identical **processing elements** PEs connected by global packet-switching network. This network is a three-dimensional bused-cube network as shawn in Fig.8. Packet transmission proceeds via a store-and-forward protocol, which allows any PE to transfer data to any other PE.

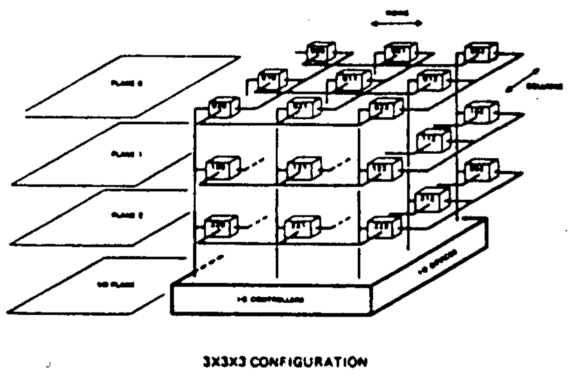


Fig.8. The HDFM cubic bus interconnetion network.

Each PE can execute the instruction set and perform the data flow sequencing and addressing. Each PE has its own local memory for both program and data storage. There is no global memory. The program, which consists of data flow graph nodes, is allocated to the local memories of the PEs at compile time. When nodes are implemented in an architecture they are called templates. A template consists of an opcode, slots for operands and destination pointers, which indicate the nodes to which the results of the operation should be sent. Each PE has a unique 9 bit address corresponding to its position in the cube (plane, column, row). This allows up to 8 PEs per bus for a maximum configuration of 512 PEs. Within each PE are multiple templates, each of which has unique template address. To implement the data flow model, the results of one template are sent to another in the form of packets. Each packet consists of a type field, a destination address, and data. The destination address indicates the PE address and the template address of the template to which the data is to be sent. Packets travelling from one PE

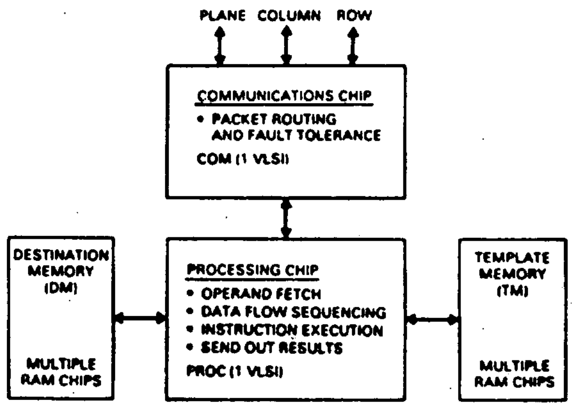


Fig.9. The structure of the PE.

to another allways take the same path. This principle, called single path routing, is requred to preserve the order of packets in time-ordered data groups such as strings. Each PE consists of four parts: **communication chip** COM, **processing chip** PROC, **destination memory** DM, and **template memory** TM. The COM receives packets from the routing network and either forwards them onto other PEs or sends them to its attached PROC. When the PROC receives a packet, it determinates whether the packet has enabled a template to fire. If so, the template opcode and the operands stored in the TM are fetched, combined with the incoming packet, which enabled the template to fire, and sent to ALU. Simultaneously, the destination addresses to which the result should be sent are fetched. The ALU performs the operation and the results are matched with their destination address to form packets. The packets are sent either back to the same PE or out into the routing network. If the template is not ready to fire, the arriving packet is stored in the TM. Since a template result may need to be sent to multiple destinations, there is additional destination overflow storage in the DM to accomodate lists of destinations for a node.

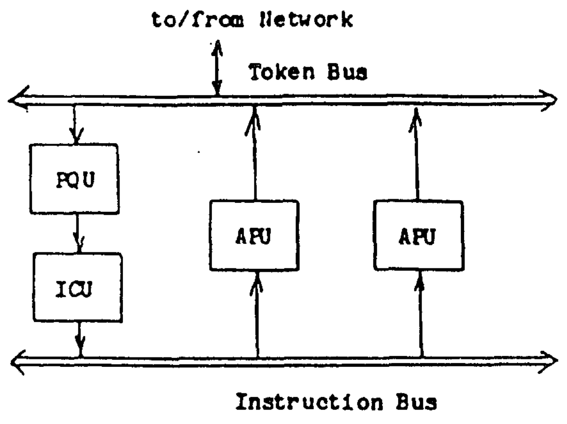
Simulation results demonstrate high-performance operation with high-level language programmability. For example, the results of the deterministic simulation of the machine show that a 64 processing element machine may provide real throughput of 64 MIPS.

PIM-D - the dataflow-based parallel inference machine

The research and development of the parallel inference machine includes the data flow mechanism to rapidly execute inference operations. The data flow model has also similarity to the logic programming languages. Execution of logic programs is performed in a goal driven manner; a clause in the program is initiated when a goal is given and returns the results to the goal. The logic programs are compiled into data flow graphs [Bic-84]. **PIM-D** is an example arhitecture to support parallel version KL1 of the kernal language for ICOT's fifth generation computers [Ito-86].

PIM-D is, similarly to SIGMA-1, constructed from multiple **processing elements** PEs and **structure memories** SMs interconnected by a network.

Each PE has several stages. The packets transferred between these stages include result packets and executable instruction packets. A result packet consists of three fields: identifier, destination and the data. The identifier



PQU: Packet Queue Unit
ICU: Instruction Control Unit
APU: Atomic Processing Unit

Fig.10. Configuration of the processing element.

specifies the invoked procedure instance to which the result packet belongs. The destination specifies the destination instruction address of the result packet. It also specifies whether the destined instruction receives a single operand or two operands. The data field contains the operand data to be sent to the instructuion. Fig.10 depicts the configuration of each PE. **Packet queue unit** PQU is a FIFO queue memory to store the result packets from **the token bus. Instruction control unit** ICU receives the result packets from PQU and checks if the destination instructions are executable or not. An instruction is executable if it receives a single operand, or if the partner operand is already in the **operand memory** in the ICU when it receives two operands. In the later case, the ICU searches in its operand memory whether the partner operand exists or not. If it does, the partner is removed from the memory; otherwise, the result packet is stored in the operand memory. This searching is performed associatively by hardware hash using the identifier and the destination address as the key field. If the instruction is executable, the ICU fetches the instruction code in its **instruction memory** and constructs an executable instruction packet and sends the packet to the next stage, one of **atomic processing units** APUs via the **instruction bus**. The APU interprets the instruction packets and sends result packets to the PQU in its PE or other PEs, or sends structure access comand packets to SMs via the token bus. The SMs are responsible for the structure access commands, perform structure manipulation operations, and return results to the destination specified by the commands.

Actual implementation of the experimental machine is currently beeing developed. The machine includes 8 PEs, 7 SMs, and one host computer used to monitor or debug the system. The APUs and SMs are implemented as microprogram control units using bit-slice microprocessors or special hardware to recognize the data tag. The ICUs are also microprogram controlled to implement hashing hardware. A software simulator for OR-parallel and Concurrent Prolog was developed. Performance evaluation results from the software simulator show that about one million head unifications per second can be achieved by exploiting parallelism.

Conclusions

Since about 1970 there has been a growing and widespread research interest in data flow computer architecture. This interest has culminated in many designs for data-driven computer systems, several of which have been or are in the process of being implemented in hardware.

The major long-term interest in dataflow techniques will be in the construction and performance of multiprocessor systems. It is particulary important to know how dataflow systems should be designed for implementation in VLSI, and to be certain that effictive software techniques are avaiable for utilizing the hardware.

References

[Bic-84] L.Bic, A data-driven model for parallel interpretation of logic programs, Proc. Int'l Conf. Fifth Gen. Comp. Systems, ICOT (1984) 517-523.

[Dennis-80] J.B.Dennis, Data flow supercomputers, Computer **13** (11) (1980) 48-56.

[Dennis-83] J.B.Dennis, W.Y.-P.Lim, and W.B.Ackerman, The MIT data flow engineering model, in: R.E.A.Mason, Ed. Information Processing 83, (Elsevier Science Publishers B.V., North-Holland, 1983) 553-560.

[Gaudiot-85] J.-L.Gaudiot, R.W.Vedder, G.K.Tucker, D.Finn, and M.L.Campbell, A distributed VLSI architecture for efficient signal and data processing, IEEE Trans. Comp. **34** (12) (1985) 1072-1087.

[Gurd-85] J.R.Gurd, C.C.Kirkham, and I.Watson, The Manchester prototype dataflow computer, Comm. ACM **28** (1) (1985) 34-52.

[Hartimo-86] I.Hartimo, K.Kronlöf, O.Simula, and J.Skyttä, DFSP: A data flow signal processor, IEEE Trans. Comp. **35** (1) (1986) 23-33.

[Ito-86] N.Ito, M.Kishi, E.Kuno, and K.Rokusawa, The dataflow-based parallel inference machine to support two basic languages in KL1, in: J.V.Woods, Ed. Fifth Generation Computer Architectures, (Elsevier Science Publishers B.V., North-Holland, 1986) 123-145.

[NEC-85] NEC Electronics, Image pipeline processor μPD7281D, Product Description (1985).

[Mavrič-86] S.Mavrič, B.Mihovilovič, and P.Kolbezen, The interconnection network in a multiprocessor system, Informatica **10** (4) (1986) 44-50 (in Slovene).

[Patnaik-86] L.M.Patnaik, R.Govindarajan, and N.S.Ramadoss, Design and performance evaluation of EXMAN: an EXtended MANchester data flow computer, IEEE Trans. Comp. **35** (3) (1986) 229-244.

[Robič-86] B.Robič and J.Šilc, Classification of new generation computer architectures, Informatica **10** (4) (1986) 18-32 (in Slovene).

[Shimada-86] T.Shimada, K.Hiraki, K.Nishida, and S.Sekiguchi, Evaluation of prototype data flow processor of the SIGMA-1 for scientific computation, Proc. 13th Int'l Symp. Comp. Arch., IEEE (1986) 226-234.

[Šilc-86] J.Šilc and B.Robič, Data flow architecture based processor, Informatica **10** (4) (1986) 74-80 (in Slovene).

GEOMETRIJSKO MODELIRANJE UPOTREBOM EULEROVIH FORMULA

INFORMATICA 1/87

Petar Kočović
SOUR Dvadeset prvi maj – Beograd
RO Fabrika turbomotora
Grupa za CAD/CAM

UDK 681.3:519.173

SAŽETAK. Ovaj članak prikazuje , ali ne dokazuje , koliko se različitih E ulerovih formula može prihvatiti . Dalje će se posmatrati neke praktične primene u modeliranju graničnom geometrijom.

ABSTRACT. In this article would be demonstrate , but do not prove , how different versions of the Euler formula can be obtained . We then consider some practical applications in boundary geometric modelling.

Kompjuterska grafika koristi mnoge različite tipove modeliranja čvrstih tela (Solid Modeler) kao što su CSG (Constructive Solid Geometry), Octree , poluprostor (Half-space) itd /1/.

CSG modelari opisuju kompleksno telo putem Booleove kombinacije jednostavnih zapremina koje se nazivaju primitivama. Modelari sa graničnom reprezentacijom (Boundary Modelers) koriste graničnu geometriju (površi , krive , tačke) i najčešće veze izmedju tih geometrijskih suseda. Ove veze se proučavaju u topologiji . Takođe je moguće CSG modele prevesti na granične modele.

Za ocenu gore pomenutih veza i odnosa koristi se u literaturi poznata Eulerova ili Euler-Poincareova formula , odnosno Eulerovi operatori/2, 3/. Ovaj rad će diskutovati problematiku vezanu za granične modele i pokazati kako modelari koji baziraju na žičanoj ili čvrstoj predstavi mogu da koriste Eulerove operatore.

TOPOLOGIJA

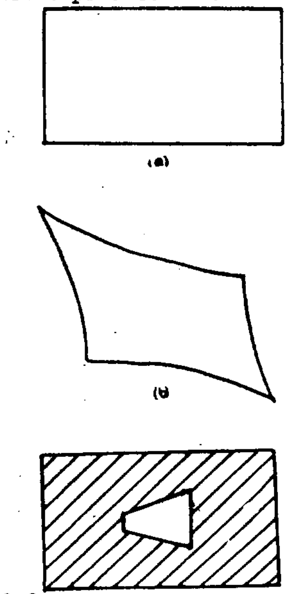
Postoje razne grane topologije . Rad će se baviti topologijom susedstva (Adjacency Topology)/2/. Geometrija počiva na vezama izmedju geometrijskih entiteta : površi , krivih , tačaka. Topologija počiva na vezama između odgovarajućih topoloških entiteta koji se nazivaju licima , ivicama i temenima . Veza izmedju osnovnog i proširenog skupa topoloških i geometrijskih entiteta je data na sl 1.

Za razliku od geometrijskog topološki prostor predstavlja "rastegljiv" prostor , odnosno topologija ne barata sa dimenzijama već sa oblicima. Za dalje , lakše , praćenje možemo smatrati da su figure koje će biti opisane na narednim slikama napravljene od hartije , sem u slučaju kada bude bilo reči o žičanim modelima

$$\begin{tikzcd}
\text{GRANIČNI OBJEKAT} \arrow[d, dash] & \\
\text{OPNA} \arrow[d, dash] & \\
\text{LICE} \arrow[d, dash] \arrow[r] & \text{POVRŠ} \\
\text{PETLJA} \arrow[d, dash] & \\
\text{IVICA} \arrow[d, dash] \arrow[r] & \text{KRIVA} \\
\text{TEME} \arrow[r] \arrow[uu, dashed, bend left=60, <->] & \text{TAČKA}
\end{tikzcd}$$

Sl 1 Hijerarhija topoloških entiteta

Razmotrimo prvo lik dat na sl 2a. Važne

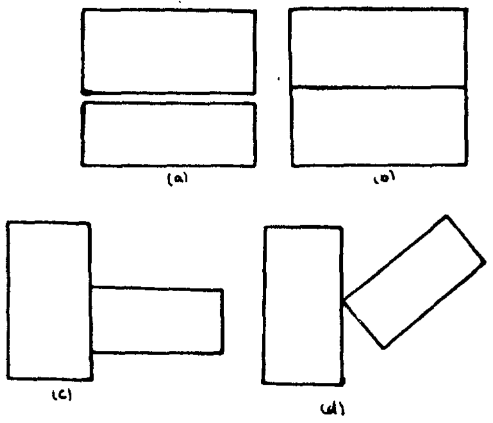


Sl 2 Primeri pojedinih likova

osobine ovog lika su da ima ograničenu površinu i da 4 linije ne dopuštaju njegovo širenje . Ako bi ovaj lik bio napravljen od gume mogao bi da se izvitoperi onako kako je dato na sl 2b. Ova dva lika imaju po 4 ugla , 4 temena i 4 ivice . Govoreći jezikom geometrije objekti 2a i 2b su različiti objekti , ali u topološkom smislu to su isti objekti . Na sl 2c je dat objekat koji se u obnosu na prethodne objekte razlikuje i u topološkom i u geometrijskom smislu: on ima 8 temena , osam ivica i jedan otvor. Kao što se sa slike 1 vidi može se uvesti analogija izmedju topološkoh i geometrijskih entiteta : teme (topološki) je tačka (geometrijski) i ivica (topološki) je kriva (geometrijski) , uz napomenu da su prave specijalni slučajevi krivih .

POJAM OTVORENIH I ZATVORENIH OBJEKATA

Ako se objekat sa slike 2a "preseče" na dva dela dobiće se objekat kao na slici 3a, tako da sada postoje dve figure. Ova dva objekta

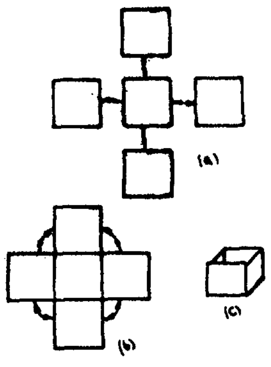


Sl 3 Razdvajanje i spajanje likova

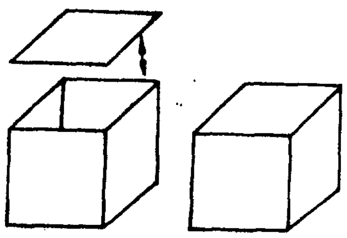
mogu da se spoje pa da daju isti objekat kao i pre sečenja (sl. 3b). Žalost , ova dva lika mogu da se spoje na različite načine , kao što je prikazano na sl. 3. Na slici 3c je prikazan spoj po ivici , a na sl. 3d po temenu . Ovaj poslednji ne može da se izvede kao "čvrsti spoj" , tj on "visi" . Modeli 3b i 3c će se zato smatrati ispravnim , a 3d neispravnim.

Znajući ovo , od lika na slici 3a možemo spajanjem napraviti krst 4b ili kutiju (sl 4c) Modeli ovakvog tipa se nazivaju "otvorenim" objektima . Ako se takvom liku doda šesti lik, objekat postaje kocka , kako je prikazano na slici 5. Ovakvi objekti se nazivaju zatvorenim objektima .

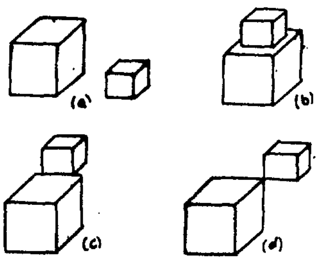
Na isti način kao što se spajaju likovi da bi se dobio zatvoreni objekat isto tako mogu da se spajaju i zatvoreni objekti . Na sl. 6 je



Sl 4 Spajanje likova



Sl 5 Dodavanje šestog lika (a) da bi se napravila kocka



Sl 6 Spajanje dva zatvorena objekta

je prikazan spoj dve kocke. Ne sme se dopustiti da objekti budu spojeni po temenima i ivicama (sl 6c , d).

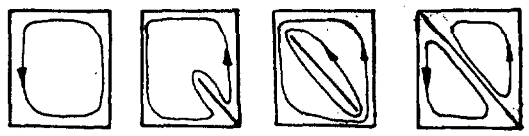
PETLJA

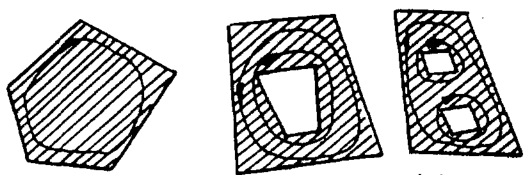
Petlja je zatvoren uredjen skup ivica i temena koji se ne presecaju medju sobom . U graničnom slučaju petlja može da sadrži jednu tačku. Primer petlje je dat na sl. 7.

LICE

Lice je deo površi objekta . Čini ga skup petlji a površinu objekta definišu u geometrijskom smislu . Samo jedna petlja definiše spoljašnju

Sl 7 Primeri petlji





Sl 8 Lice sa jednom petljom bez rupe (a), sa dve petlje i jednom rupom (b) i sa tri petlje i dve rupe (c)

granicu lica . Unutrašnja granica se može sastojati od više petlji , sl 8.

EULEROVE FORMULE

Postoji više verzija ove formule . Matematičar Euler je sugerisao pravilo koje se odnosi na broj elemenata poliedra :

$$T - I + L = 2 \qquad (1)$$

gde su T,I,L brojevi temena , ivica i likova Jednačina (1) predstavlja dobro poznati Eulerov zakon :

" U svakom jednostavnom poliedru broj lica (L) , ivica (I) i temena (T) moraju da zadovoljavaju gornju jednačinu"

U nastavku rada će biti prikazane razni tipovi ove formule koje su proširene uvođenjem dodatnih entiteta.

POLIGONALNI OBJEKTI

Poligonalni objekti su ravanski likovi sastavljeni od pravih linija . Svi dosadašnji objekti su poligonalni . U tabeli 1 su dati parametri ovih objekata . Vidi se da jednačina (1) ne važi u svakom slučaju , pa se ona mora modifikovati . Za otvorene objekte važi sledeća jednačina :

$$X1 = T - I + L - S_o = 0 \qquad (2)$$

Za zatvorene objekte važi relacija :

$$X2 = T - I + L - 2S_z = 0 \qquad (3)$$

gde su :

S_o - broj otvorenih objekata

S_z - broj zatvorenih objekata

OBJEKTI SA RUPAMA

Posmatraćemo objekte na slikama 9 i 10. Rupa se definiše kao prolaz koji zahvata ceo objekat . To nije udubljenje u stranici objekta

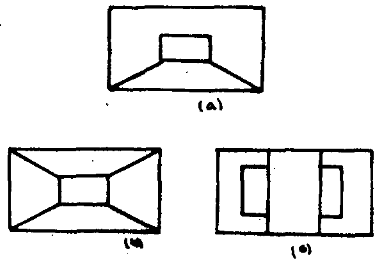
TABELA 1

Poligonalni objekti bez otvora

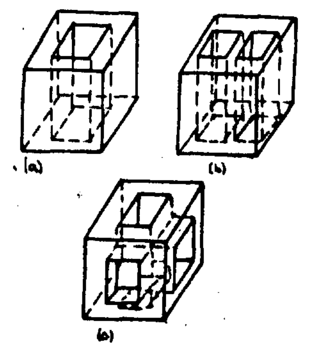
Slika	T	I	L	S_o/S_z	X1	X2
2a	4	4	1	1	0	
3a	8	8	2	2	0	
3b	6	7	2	1	0	
3c	8	9	2	1	0	
4a	20	20	5	5	0	
4b	12	16	5	1	0	
4c	8	12	5	1	0	
5b	8	12	6	1		0
6a	16	24	12	2		0

$X1 = T - I + L - S_o$

$X2 = T - I + L - 2S_z$



Sl 9 Primeri otvorenih objekata sa rupama : (a) jedna rupa , dva lica , (b) jedna rupa i 4 lica , (c) dve rupe tri lica



Sl 10 Primeri zatvorenih objekata : (a) sa jednom rupom , (b) sa dve rupe i (c) sa tri rupe

koje odgovara rupi u ravanskoj figuri . Za dalji rad su nam potrebne takodje i informacije o petljama (P). Ako se sa R označi broj rupa dobija se sledeći par jednačina :

- za otvorene objekte

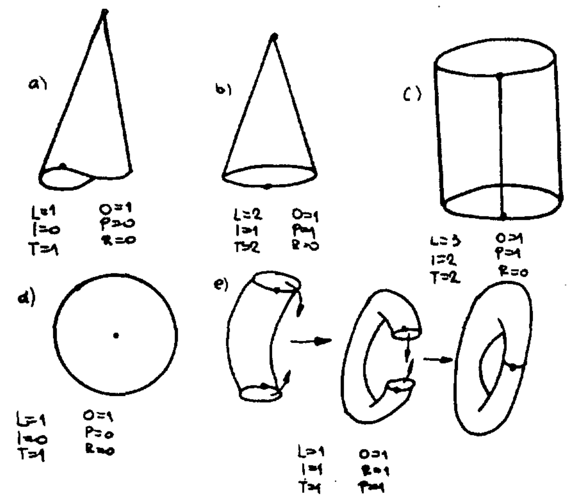
$$X3 = T - L + 2L - P - (S_o - R) = 0 \qquad (4)$$

- za zatvorene objekte

$$X4 = T - I + 2L - P - 2(S_z - R) = 0 \qquad (5)$$

OBJEKTI KOJI SE SASTOJE OD "KRIVIH" STRANICA

Prvobitne studije topologije vezane za tela tretirale su poliedre sa ravnim stranicama . U zavisnosti od toga kako su zakrivljene površi s egmentišemo u stranice , nije potrebno da re zultujuća topologija zadovolji sve uslove koje zahtevaju ravni poliedri . Na primer , moguće je da posmatramo konusnu površ kao izolovano teme bez ivica , sl 11a, a konus kao skup teme na i ivice . Medjutim , najsvrsishodnije je ra zdeliti konus u segmente tako da konus razmatramo kao poliedar .



Sl 11 Topološka analiza nekih objekata sa zakriv ljenim stranicama : a) konusna površ , b)konus , c)valjak , d)lopta i e)torus

Cilindar , sl 11b, posmatramo kao telo koje ima 3 lica (2bazisa + omotač = 3lica), 2 ivice (po jedna za granice svakog od bazisa sa jedinstvenim omotačem), dva temena (uzima se da svaka kružnica mora da ima po jedno teme , tj. tačku iz koje počinje i u kojoj se završava kružnica). Cilindar ima jednu petlju i nema rupa.

Lopta se može razmatrati na primeru pravljenja mehura od sapunice . Mehur nastaje iz diska koji je "razapet" preko žičanog prstena . Kada je mehur naduvan disk se transformiše u sferu , a teme pokazuje poziciju zatvaranja sfere. Ovo teme je , ustvari , teme kada se mehurić odvojio od prstena . Ostali podaci su dati na sl 11d

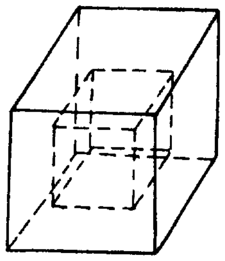
Torus je dat na sl 11e. Rigoroznija definicija torusa je data u /9/.

OPNE I RODOVI

Ranije su rupe opisane kao prolazi kroz objekat. Topološki pojam za rupu je rod (genus) .

Opna je skup lica koja konstituišu jednu topološku površinu objekta . Jedna opna je potrebna da se definiše spoljašnja granica objekta , a po jedna opna opna za svaku prazninu u objektu. Opna deli prostor na dva dele .

Može da egzistira objekat prikazan slikom 12



Sl 12 Kocka sa mehurom u obliku kocke

Sada se mogu postaviti jednačine za otvorene i zatvorene objekte :

$$X5 = T - I + 2L - P - (O - R) = 0 \quad (6)$$

$$X6 = T - I + 2L - P - 2(O - R) = 0 \quad (7)$$

Zbog potreba da se otvoreni i zatvoreni objekti posmatraju jedinstveno opšta jednačina dobija oblik :

$$X7 = T - I + L - P - 2(O - R) = 0 \quad (8)$$

i važi samo za 3D objekte .

ŽIČANI MODELI

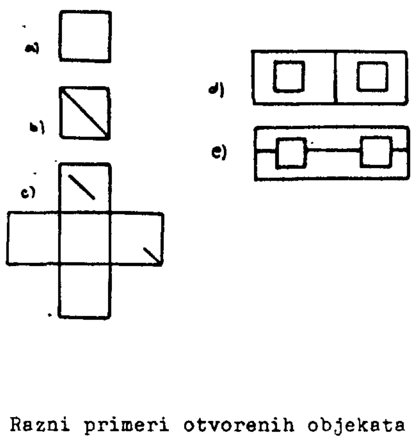
Druga klasa modela , poznata kao žičani model (wire -frame) model , sadrži informacije o temenima i ivicama , ali ne i o stranicama . Za ovakve modele se može postaviti jednačina :

$$Xž = T - I + P - 2 = 0 \quad (9)$$

gde P predstavlja broj petlji . Ako postoji više od jednog žičanog modela izraz (9) se generalizuje

$$Xž = T - I + P - 2M = 0 \quad (10)$$

gde je M broj modela .



Sl 13 Razni primeri otvorenih objekata

SUMARNI PREGLED

Sumarni pregled ovih objekata , ako bi se posmatrali kao čvrsta tela je dat u tabeli 2.

Eulerovi operatori koji pomoću vektora tranzicije omogućuju i verifikaciju modeliranja .

TABELA 2
Svi modeli posmatrani kao čvrsta tela

Slika	T	I	L	P	O	R	LL	LP	X1	X2	X5	X6	Napomena
2a	4	4	1	1	1	0	1	1	0		0	0	
2c	8	8	1	2	1	1	1	2	0		0	0	
3a	8	8	2	2	2	0	2	2	0		0	0	
3b	6	7	2	2	1	0	1	1	0		0	0	
3c	8	9	2	2	1	0	1	1	0		0	0	
3d	8	9	2	2	1	0	1	2	0		0	-1	*
4a	20	20	5	5	5	0	5	5	0		0	0	
4b	12	16	5	5	1	0	1	1	0		0	0	
4c	8	12	5	5	1	0	1	1	0		0	0	
5a	12	16	6	6	2	0	2	2	0		0	0	
5b	8	12	6	6	1	0				0		0	
6a	16	24	12	12	2	0				0		0	
6b	16	24	11	12	1	0				1		0	
6c	16	25	12	12	1	0				1		1	*
6d	15	24	12	12	1	0				1		1	*
9a	8	10	2	2	1	1	1	2	-1		0	0	
9b	8	12	4	4	1	1	1	2	-1		0	0	
9c	16	20	3	3	1	2	1	3	-2		0	0	
10a	16	24	10	12	1	1				0		0	
10b	24	36	14	18	1	2				0		0	
10c	30	48	18	22	1	3				-2		0	
12	16	24	12	12	2	0				0		0	
13b	4	5	2	2	1	0	1	1	0		0	0	
13c	15	18	5	6	1	0	1	1	1		0	0	
13d	14	15	2	4	1	2	1	3	0		0	0	
13e	18	21	2	2	1	2	1	3	-2		0	0	

1. Figure označene zvezdicom su nepravilni objekti
2. X1 = T - I + L - S
   X2 = T - I + L - 2O
   X5 = T - I + 2L - P - (O - R)
   X6 = T - I + 2L - P - 2(O - R)
3. Kolone označene sa LL i DP su "lažno lice" i "dodatna petlja" za otvorene objekte da bi mogli da se računaju po jednačini X6

ZAKLJUČAK

Na osnovu pokazanih formula može se suditi o topološko geometrijskoj ispravnosti objekta . Ova analiza ima velikog značaja pri projektovanju CAD sistema , a naročito modela ra koji baziraju na čvrstim telima . Eulerova relacija koja se odnosi na entitete koji uključuju lica , ivice i temena ne zadovoljava sve objekte . To se postiže uvodjenjem dodatnih entiteta. U radovima /3,8/ su dati osnovni

DODATAK

Svrha ovog dodatka je da omogući skup definicija termina koji su u radu bili manje ili više precizno diskutovani .

TEME (Vertex)	je jedinstvena tačka u prostoru. Teme pripada jednom ili više lica .
IVICA(Edge)	je skup od dva temena koji se nalazi na preseku dve ravni . Ne sme da preseca sama sebe .

TABELA 3
Žičani modeli

Slika	T	I	P	M	Xž
2a	4	4	2	1	0
2c	8	8	4	2	0
3a	8	8	4	2	0
3b	6	7	3	1	0
3c	8	9	3	1	0
3d	8	9	3	1	0
4a	20	20	10	5	0
4b	12	16	6	1	0
4c	8	12	6	1	0
5a	12	16	8	2	0
5b	8	12	6	1	0
6a	16	24	12	2	0
6b	16	24	12	2	0
6c	16	25	11	1	0
9a	8	10	4	1	0
9b	8	12	6	1	0
9c	16	20	6	1	0
10a	16	24	12	2	0
10b	24	36	18	3	0
10c	30	48	22	2	0
12	16	24	12	2	0
13b	4	5	3	1	0
13c	15	18	7	2	0
13d	14	15	7	3	0
13c	18	21	5	1	0

$Xž = T - I + P - 2M$

LICE (Face) je deo površi objekta . To je jedinstvena petlja i istovremeno jedinstvena opna . Površinu objekta definiše u geometrijskom smislu . Ne sme da seče samo sebe .

PETLJA (Loop) je uredjen naizmeničan skup temena i ivica koji se ne presecaju medju sobom . U graničnom slučaju petlja može da sadrži samo tačku.

PRSTEN(Ring). Dok se za petlju smatra da je spoljašnja petlja , prstenom se definišu sve petlje , i one nastale od granica lica i one koje nastaju usled postojanja rupa (Ovaj rad poistovećuje petlju i prsten)

OPNA(Shell) je skup lica koja konstituišu jednu topološku površinu objekta. Jedna opna je potrebna da definiše spoljašnju granicu objekta , a po edna opna za svaku prazninu u objektu.

ČVRSTA GRANIČNA REPREZENTACIJA(Solid Boundary Representation) je skup entiteta , topoloških i geometrijskih, koji nedvosmisleno definišu zatvorene zapremine koje ne seku same sebe .

REPREZENTACIJA ŽIČANIM MODELOM (Wire Boundary Representation) je skup topoloških i geometrijskih entiteta koji definišu žičani model.

LITERATURA

1. A.A.G. Requicha , H.B. Voelcker : "Solid Modelling : Current Status and Research Directions " , IEEE Computer Graphics and Applications", Vol 3, No 7 , Oct. 1983, str 25+37
2. P. Wilson:"Euler formulas and geometric modeling",IEEE CG&A,Vol 5, No 8, Aug 1985, str 24 + 36
3. M. Mäntylä : "Solid Modeling : Theory and Applications", EUROGRAPHICS '83 , Pre-conference Tutorial, Zagreb , August 1983
4. Baer A., Eastman C., Henrion M.:"Geometric Modelling : a survey" , CAD , Vol 11, No 5, September 1979 , str 253 + 272
5. J. Zagajac :"Edvis Solid Modeler and solid body modelling", Druga Austro-Jugoslovenska konferencija o kompjuterskoj grafici , Maribor , Jun 1986
6. K. Weiler :" Edge-based data structures for solid modeling in curved surface environments IEEE CG&A , Jan. 1985 , str 21 + 40
7. P.R. Wilson , I.D. Faux , M.C. Ostrowski , K.G. Pasquill :" Interfaces for data transfers between solid modeling systems",IEEE CG&A , Jan 1985 , str 41 + 51
8. M. Mäntylä :" Set operations of GWB",Computer graphics forum", Vol 2 , No 2/3 , Aug. 1983
9. M. Mäntylä :" Topological analysis of polygon meshes", CAD , Vol 15, No 4 , July 83
10. M.Milačić :" Razvoj i primena matematičke metode dekompozicije na višenivojske hijerarhijske sisteme sa inform.bazom",Doktorska disertacija , Mašinski fak, Beograd , 1980

OBČASNA NOTRANJA DIAGNOSTIKA MIKRORAČUNALNIKA EPM – 850

INFORMATICA 1/87

A. Temeljotov, D. Mrdaković,
D. Pavšelj, D. Čuk
Institut »Jožef Stefan« Ljubljana

UDK 681.3-181.4.001.4

Članek opisuje sistem za občasno notranjo diagnostiko 8-bitnega enokartičnega mikroračunalnika EPM-850 zasnovanega na mikroprocesorju INTEL 8085. Diagnostika je izvedena popolnoma programsko, zato se posamezni pristopi in testni postopki lahko enostavno priredijo za preizkušanje mikroračunalnikov drugih tipov.

SELF DIAGNOSTICS FOR EPM-850 MICROCOMPUTER

In the following article the self diagnostic system for single board 8-bit microcomputer EPM-850, based on the INTEL 8085 is described. The diagnostic program tests the computer hardware upon each reset or power-up and ensures detecting majority of possible faults in the system. Diagnostic procedures run without special hardware support, therefore the described testing approaches can be easely used for other computer types.

1. UVOD

Zanesljivost mikroračunalniških sistemov postaja lastnost, ki pri odločitvah o nakupu sistemov za vodenje industrijskih procesov prihaja vedno bolj v ospredje. Drugi razlog, ki načrtovalce računalniške opreme sili v bolj sistematično obravnavanje te problematike je, da pravočasno odkrivanje in lokalizacija okvar bistveno zmanjšujeta stroške vzdrževanja računalniške opreme.

Po nekaterih ocenah stroški vzdrževanja računalniških sistemov v industriji dosegajo do 25 % cene vgrajene aparaturne opreme. Če upoštevamo škodo zaradi daljše prekinitve proizvodnega procesa, postanejo ti stroški še večji. Z uvajanjem enostavnih diagnostičnih postopkov in ob podpori priučenih vzdrževalcev, ki bi znali izvršiti manjša popravila, kot so menjava modula računalnika ali menjava integriranega vezja, bi lahko te stroške bistveno zmanjšali, hkrati pa bi povečali razpoložljivost računalniškega sistema.

Za spremljanje pravilnosti delovanja računalniškega sistema potrebujemo učinkovit sistem za odkrivanje napak in okvar v njem. Napake v računalnikih lahko odkrivamo na več načinov: od "ročnega" (s pomočjo osciloskopa, logičnega analizatorja ipd.), ki je nujen v fazi razvoja in delno v fazi proizvodnje, do popolnoma avtomatiziranega odkrivanja napak, ki ga vodi računalnik sam.

V tem delu je predstavljen sistem za občasno notranjo diagnostiko 8-bitnega enokartičnega mikroračunalnika EPM-850, vendar se večina pristopov k testiranju lahko uporabi tudi za preizkušanje mikroračunalnikov drugih tipov.

2. DIAGNOSITKA MIKRORAČUNALNIKA EPM-850

2.1 Izbira diagnostične metode.

Diagnostika računalniškega sistema pomeni odkrivanje in lokalizacijo napak v njem. Znanih je več pristopov k diagnostiki računalniških sistemov glede na čas izvajanja, realizacijska sredstva in diagnostično strategijo. Pri razvoju diagnostičnih postopkov je pomemben tudi način izbire testnih vektorjev.

V našem primeru smo se lotili izdelave diagnostičnega sistema za mikroračunalnik, ki je že v uporabi. S stališča aparaturne opreme je torej dokončan in se serijsko proizvaja. Zato pri zasnovi testnih postopkov nismo mogli zahtevati spremembe v vezju računalnika, izdelava zunanje testne naprave pa bi bila predraga. Zaradi tega smo se odločili za popolnoma programsko diagnostiko, ki bi bila prisotna v sistemu samem.

Pri preizkušanju računalniških sistemov ima poseben pomen definiranje diagnostičnega jedra, to je tistega dela računalnika, ki pošilja testne vzorce na vhode posameznih delov sistema in opazuje njihove izhode. V sistemih z notranjo diagnostiko v praksi običajno težko zagotovimo, da je del računalnika, ki služi kot diagnostično jedro, bistveno zanesljivejše od ostalih delov sistema. Ker pa diagnostično jedro mora delovati, če želimo da se diagnostična procedura izvede, si pomagamo z metodo širjenja diagnostičnega jedra. Po tej metodi se najprej preizkusi zelo majhen del diagnostičnega jedra - elementi prvega nivoja. Če ta del deluje nam v nadaljevanju služi za preizkušanje elementov drugega nivoja, ki se nato pridružijo diagnostičnemu jedru. Ta postopek se nadaljuje dokler se ne ustvari jedro, ki je preizkušeno in ki lahko samostojno preizkusi vse ostale dele sistema.

Testne vektorje smo določali vnaprej na osnovi proučevanja posameznih komponent in napak, ki bi v njih lahko nastopile. Program je zgrajen tako, da lahko večino programov za preizkušanje posameznih delov sistema kličemo tudi med izvajanjem uporabniških programov, ob ustreznem dodeljevanju procesorja.

2.2 Zgradba mikroračunalnika EPM-850

Mikroračunalnik EPM-850, izdelan na Inštitutu Jožef Stefan, se lahko uporablja kot samostojni računalnik za vodenje industrijskih procesov, ali kot periferna mikroračunalniška postaja v distribuiranem sistemu za vodenje kompleksnejših industrijskih procesov. Realizirane ima naslednje funkcije:

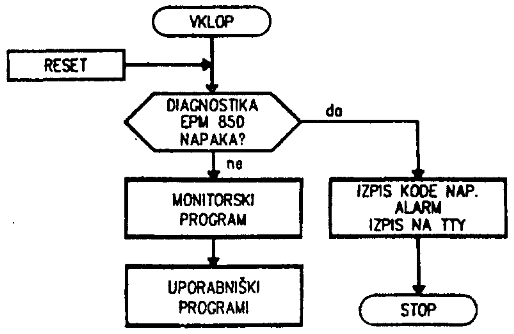
- mikroprocesorska enota INTEL 8085,
- 64K besed pomnilnika, ki ga lahko poljubno konfiguriramo glede na dolžino RAM in EPROM,
- 2 serijska sinhrona komunikacijska kanala, za povezavo v računalniško mrežo,
- 2 serijska asinhrona komunikacijska kanala,
- 4 DMA kanali,
- programsko nastavljivi števci,
- 12 nivojska prekinitvena struktura,
- aritmetični procesor INTEL 8231,
- ura realnega časa,
- časovnik za nadzor delovanja procesorja (Watch dog),
- 24 digitalnih vhodov,
- 24 digitalnih izhodov,
- EMV vodilo za priključitev modulov z digitalnimi ter analognimi vhodi in izhodi,
- baterijsko napajanje RAM-a.

2.3 Diagnostika EPM 850

Pri načrtovanju diagnostičnega sistema za EPM-850 smo se odločili za trojno preizkušanje računalnika. Najprej se računalnik preikzusi po izdelavi (tako imenovani proizvodni testi).

Drugi pristop je uporabljen za preizkus računalnika ob vklopu oziroma po izpadu napajalne napetosti ali po RESET-u, ki ga lahko izvrši tudi enota za nadzor delovanja sistema v realnem času. Predvideno je tudi preizkušanje in nadzor računalnika med vodenjem industrijskih procesov (ON-LINE TEST).

Ker je EPM-850 mikroračunalnik, ki je ponavadi stalno priključen na proizvodni proces, moramo biti prepričani, da po vsakem krajšem ali daljšem izpadu napetosti sistem pravilno deluje. Zato se ob vsakem vklopu ali po RESET-u sistema najprej izvede diagnostični program, ki preveri delovanje aparaturne opreme mikroračuanlnika. V primeru napake v računalniku diagnostični program ustavi nadaljnje delovanje sistema in javi napako. Če sistem deluje pravilno, se po uspešno izvedenem testnem programu sistem incializira in se začne izvajanje uporabniških programov.


VKLOP
RESET
DIAGNOSTIKA EPM 850 NAPAKA?
da
ne
MONITORSKI PROGRAM
UPORABNIŠKI PROGRAMI
IZPIS KODE NAP. ALARM IZPIS NA TTY
STOP

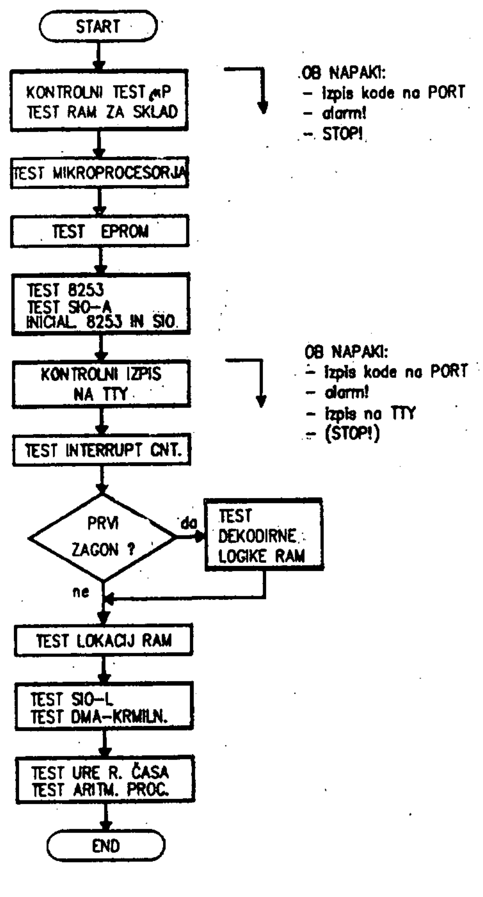
Slika 1: Delovanje sistema po vklopu.

Diagnostični program se začne s kratkim testom mikroprocesorja, med katerim se preizkusi delovanje registrov in se izvrši ena aritmetična operacija. Če je preizkus uspešen, se program nadaljuje s preizkusom dela delovnega pomnilnika (100 lokacij), v katerem bo sklad (delovanje sklada je pogoj za izvrševanje razvejitvenih instrukcij). Test mikroprocesorja se nadaljuje s popolnejšim preizkusom registrov, ki mu sledi temeljit instrukcijski test. Za tem se izvrši preizkus bralnega pomnilnika, v katerem sta shranjena diagnostični in monitorski program, ter sistemska programska oprema. V tem trenutku je diagnostično jedro, ki ga tvorijo mikroprocesor, naslovno in podatkovno vodilo, EPROM z diagnostičnim programom, 100 lokacij RAM in register, v katerega vpisujemo kode napak, preizkušeno in lahko samostojno testira ostale elemente računalnika. V primeru napake program sproži alarm in izpiše kodo napake v 8-bitni register (imenujemo ga PORT), ki ga operater lahko prebere z bitnim analizatorjem ali voltmetrom ter ustavi delovanje sistema.

Za preizkusom osnovnega diagnostičnega jedra, se preizkusijo komponente, ki sodelujejo pri komunikaciji računalnika s terminalom. Potem, ko testni program ugotovi, da komunikacija s terminalom poteka pravilno, javlja napake tudi z izpisom na zaslon. Informativno se na zaslon izpiše besedilo "***DIAGNOSTICS***", temu sledi preizkus prekinitvene strukture računalnika.

Pred preizkusom delovnega pomnilnika mora program ugotoviti ali gre za prvi zagon, ali pa za zagon po izpadu napajalne napetosti. V EPM-850 je predvideno, da se s pomočjo baterijskega napajanja ob izpadu napetosti ohranijo vsi podatki, ki se tedaj nahajajo v delovnem pomnilniku. Ker podprogram za preizkušanje dekodirne logike delovnega pomnilnika zaradi svoje narave med preizkušanjem uniči predhodno vsebino pomnilnika, ga izvajamo le ob prvem zagonu sistema. Program za preizkus posameznih lokacij pomnilnika ohra: predhodno vsebino in ga zato lahko izvršimo ob vsakem vklopu ali RESET-u.

V nadaljevanju se izvršijo podprogrami za preizkušanje serijskega vhodno/izhodnega krmilnika za povezavo v računalniško mrežo ter podprogram za preizkus DMA krmilnika. Na koncu se izvršita še programa za preizkus ure realnega časa in aritmetičnega procesorja.



Slika 2: Diagram poteka diagnostičnega programa.

3. PREIZKUS POSAMEZNIH DELOV SISTEMA

3.1 Preizkus mikroprocesorja

Ker je mikroprocesor gonilo računalnika, je večina napak znotraj njega usodna za nadaljnje delovanje sistema. Obstaja tudi možnost, da napaka ne povzroči takojšnjega izpada, temveč se pojavi le ob izvajanju določenih programov (npr. redko uporabljene instrukcije). Dobro zasnovani testni programi lahko večino teh napak detektirajo in pravočasno ustavijo nadaljne izvajanje.

Pri razvoju postopka za preizkus mikroprocesorja INTEL 8085 za potrebe občasne diagnostike smo izhajali iz grafičnega modela mikroprocesorja Thatea in Abrahama /4/, ki je zgrajen na nivoju registrskih prenosov. Parametri za razvoj testnih procedur so bili organizacija mikroprocesorja in njegov nabor instrukcij, ki je dostopen v uporabniškem priročniku vsakega mikroprocesorja. Testni postopek temelji na modelu napak in na sistemskem grafu mikroprocesorja. V tem grafu predstavljajo posamezni registri vozlišča, povezave med vozlišči pa ponazarjajo pretok podatkov med izvajanjem določene instrukcije. Graf vsebuje tudi dve dodatni vozlišči IN in OUT, ki predstavljata pomnilnik oziroma periferijo.

Za opis testnega postopka smo uporabljali naslednje oznake:

Ij – oznaka instrukcije ($j \in 1...n$), n je število instrukcij procesorja,
Rj – oznaka registra ($j \in 1...m$), m je število registrov v procesorju,
Sj – izvorni register,
Dj – ciljni register.

Instrukcije smo razdelili v štiri razrede:

- razred T1:8-bitne prenosne instrukcije
- razred T2:16-bitne prenosne instr.
- razred B :razvejitvene instrukcije
- razred M :izvršilne instrukcije

Izdelan je bil model napak, ki je bil osnova za izdelavo diagnostičnih procedur. Napake smo razvrstili v naslednje skupine:

Napake funkcij za dekodiranje registrov:

- izvor ni selektiran f(Sj/0),
- selektiran napačen izvor f(Sj/Sk),
- ob pravem je selektiran tudi napačen izvor f(Sj/Sj+Sk),
- cilj ni selektiran f(Dj/0),
- selektiran napačen cilj f(Dj/Dk),
- ob pravem selektiran tudi napačen cilj f(Dj/Dj+Dk).

Napake funkcij za dekodiranje instrukcij:

- instrukcija se ne izvede f(Ij/0),
- izvede se napačna instrukcija f(Ij/Ik),
- ob pravi instrukciji se izvede tudi napačna f(Ij/Ij+Ik).

Možne napake registrov:

- ena ali več celic registra stalno v 0 ali v 1,
- eden ali več parov celic v medsebojnem vplivu.

Možne napake pri prenosu podatkov (vodilo):

- ena ali več linij stalno v 0 ali v 1,
- ena ali več linij je v medsebojnem vplivu zaradi kratkih stikov ali kapacitivnih vezi.

Poleg omenjenih so možne tudi napake pri obdelavi podatkov v različnih funkcionalnih enotah, kot so arimetično-logična enota, prekinitvena enota, enota za inkrementiranje registrov itd.

Pri testiranju registrov je popolnoma upoštevan podan model napak. Najprej preiskusimo možnost napak registerskih celic, nato pa funkcij za dekodiranje registrov. Za detektiranje napak pri prenosu podatkov po internih vodilih nismo izvajali posebnih procedur, ampak smo pri preizkusu ostalih funkcij mikroprocesorja uporabljali testne vektorje, ki bi odkrili tudi možne napake v vodilih - to so vektorski pari FF-00, F0-0F, CC-33 in AA-55.

Program se začne s testom registerskih celic, s katerim preverimo, ali se vsaka celica v registrih lahko postavi v 1 oziroma v 0. To ugotovimo tako, da skozi vse registre prenesemo najprej vektor 55 nato pa AA. Če se test uspešno izvrši, je to tudi dokaz, da ni napake tipa f(Sj/0) ali f(Dj/0). Ker je kazalec sklada 16 bitni register, ga preizkusimo posebej, po testu pa vanj vrnemo predhodno vsebino. Testiranje registrov končamo s testom logike za dekodiranje registrov, torej za napake tipa f(Sj/Sk), f(Sj/Sj+Sk), f(Dj/Dk), f(Dj/Dj+Dk).

Preizkušanje mikroprocesorja nadaljujemo z instrukcijskim testom. Ta test je zgrajen tako, da detektira vse napake tipa f(Ij/0). Napake tipa f(Ij/Ik) in f(Ij/Ij+Ik) program detektira samo znotraj posameznega razreda (T1, T2, B, M), s čemer nekoliko zgubimo na kvaliteti testa, zato pa prihranimo velik del pomnilniškega prostora.

3.2 Preizkus pomnilnika

a) Delovni pomnilnik

V delovnem pomnilniku se pojavljajo tako stalne kot občasne napake. Z našim programom testiramo pomnilnik glede na prisotnost stalnih napak, pri tem pa upoštevamo naslednji model napak:

- dekodirno vezje ne selektira naslovljene lokacije,
- dekodirno vezje selektira napačno lokacijo,
- dekodirno vezje selektira več lokacij hkrati,
- ena ali več celic je stalno v 0 ali 1,
- eden ali več parov celic so v medsebojnem vplivu (sprememba stanja ene celice povzroči spremembo stanja v drugi celici).

Poleg tega mora testni program ugotoviti, če je katera linija naslovnega ali podatkovnega vodila mikroračunalnika v okvari (stalno v 0 ali 1), če je kateri par linij v medsebojnem vplivu ali pa če je prišlo do napake v prevezavi pomnilnika (glede na to, da mora biti podnožje za RAM v EPM-850 za vsako konfiguracijo pomnilnika drugače prevezano je verjetnost take napake velika).

Delovni pomnilnik testiramo z dvema programoma /6/. S prvim preverimo pravilnost delovanja dekodirne logike ter podatkovnih in naslovnih linij. Najprej v delovni pomnilnik vpišemo zaporedje 1,2,3,....,255, ki se periodično ponavlja od začetka do konca pomnilnika. Torej je dolžina zaporedja $255<>2^h$ (ker je 0 izpuščena), znotraj periode pa so vsi podatki različni. V drugem delu testa ponovno generiramo omenjeno zaporedje in ga primerjamo z vsebino pomnilnika. V primeru napake v delovanju dekodirne logike ali napake v povezavah se vsebina pomnilnika, zaradi omenjenih lastnosti zaporedja, nujno razlikuje od generiranega zaporedja in program po primerjavi javi napako.

Test dekodirne logike zaradi svoje narave mora uničiti predhodno vsebino pomnilnika in se zato izvede le ob vklopu računalnika.

Z drugim programom preizkusimo, če lahko v vsako celico pomnilnika vpišemo 0 ali 1 oziroma ugotovimo morebitno odvisnost med celicami znotraj ene lokacije. Program je narejen tako, da ohrani podatkovno vsebino pomnilnika in se zato lahko izvede ob vsakem izpadu napetosti ali pa kot ON-LINE test. Vsako lokacijo preizkusimo tako, da njeno vsebino najprej shranimo, nato vanjo zaporedoma vpišemo in takoj preberemo vektorje AA, 55 in 00. Po preizkusu v celico vrnemo predhodno vsebino.

b) Bralni pomnilnik

Napake, ki se pojavljajo v bralnem pomnilniku, so podobne tistim, ki nastopajo v delovnem pomnilniku, le da je verjetnost naključnih preklopov dosti manjša. Napake nastopajo tako v dekodirni logiki kot na področju pomnilniških lokacij. Model napak za dekodirno logiko je enak tistemu v bralnem pomnilniku, napake na področju pomnilniških celic pa se odražajo kot sprememba vsebine pomnilniških lokacij.

Naloga testnega programa za bralni pomnilnik je ugotoviti morebitno spremembo vsebine v eni ali več pomnilniških lokacij.

Preizkus bralnega pomnilnika je zasnovan na izračunu vsote vsebine vseh lokacij v bralnem pomnilniku. Program sešteje vsebino vseh lokacij v pomnilniku in izračunano vsoto primerja z vnaprej pripravljeno 16-bitno vsoto, ki je vprogramirana v za to določen prostor v pomnilniku. V primeru, da se izračunana in pripravljena vsota ne ujemata, program javi napako in ustavi delovanje mikoračunalnika.

3.3 Testiranje ostalih funkcij EPM-850

Ker se delovanje in zgradba teh komponent zelo razlikujejo, ne obstaja nekakšen splošen pristop k njihovem testiranju, temveč mora načrtovalec testnih programov na osnovi predhodnega proučevanja vsake komponente posamezno izbrati najbolj učinkovit način za njeno preizkušanje.

Testiranje vmesnikov in krmilnikov je s stališča samopreizkušanja dokaj problematično, saj to zahteva ali sodelovanje perifernih naprav, ali pa sodelovanje operaterja, ki simulira signale na vhodih oziroma spremlja določene signale na izhodih. Pri samodiagnostiki mikro računalnika EPM 850 ne pride v poštev ne sodelovanje operaterja, ne sodelovanje perifernih naprav, saj je računalnik priključen neposredno na industrijski proces oziroma na računalniško mrežo in bi ob napaki lahko okolju povzročil določene neprijetnosti.

Zato smo se pri komponentah, ki neposredno vplivajo na okolje, omejili na preizkušanje notranje strukture komponent: podatkovnih vodil, krmilnih linij, vmesnikov, registrov, dekodirne logike, itd. Večino komponent tudi inicializiramo in preverimo status komponente po inicializaciji. Komponente, ki so na določen način izolirane od okolja, oziroma na okolje ne vplivajo, preizkusimo tudi s funkcionalnimi testi, tako da preizkusimo izvajane dejavnosti, ki se izvajajo med delovanjem sistema.

Čeprav zaradi razlik v delovanju in zgradbi komponent ni možno izdelati skupnega modela napak za te komponente, obstoja nekaj tipov napak, ki se pojavljajo pri vseh oziroma pri večini elementov:

- dekodirno vezje ne omogoča dostopa do vseh registrov komponente,
- ena ali več linij internega podatkovnega vodila so stalno v 0 ali v 1,
- dve ali več linij internega podatkovnega vodila so v medsebojnem vplivu,
- onemogočena pravilna inicializacija komponente.

Ta "model" skupaj z napakami, ki so specifične za posamezno komponento, je bil osnova za izdelavo testnih postopkov in programov za preizkušanje prekinitvenega krmilnika, programsko nastavljivega časovnika, serijskih vhodno - izhodnih krmilnikov, krmilnika za neposreden dostop v pomnilnik, ure realnega časa in aritmetričnega procesorja.

4. ZAKLJUČEK

Mikroračunalniški sistemi, zgrajeni brez diagnostične aparaturne opreme ali s premalo le te, se morajo pri odkrivanju napak in okvar v aparaturni opremi opreti predvsem na diagnostično programsko opremo.

Programi, ki preizkusijo sistem nekajkrat dnevno in opozorijo operaterja ob izpadu posamezne komponente, ali dela sistema, veliko prispevajo k pravočasni odpravi napak v računalniškem sistemu. Ti programi preizkusijo tudi nekatere redko uporabljene dele sistemske logike in lahko detektirajo veliko latentnih napak.

Naloga, ki smo si jo zadali, izdelati učinkovit sistem za notranjo diagnostiko enokartičnega perifernega mikroračunalnika EPM-850 je bila uspešno opravljena. Ker aparaturne opreme ni bilo možno spreminjati, smo izdelali popolnoma programsko diagnostiko, ki se izvrši ob vsakem vklopu oziroma resetu računalnika . Program je napisen v zbirnem jeziku za INTEL 8085, ker se tako lažje približamo notranji strukturi mikroračunalnika in prihranimo na pomnilniškem prostoru. Dolžina programa je nekaj več kot 1k-byte, čas izvajanja pa je 3 sekunde. Testni postopki za posamezne dele sistema temeljijo na modelih napak, ki se v njih lahko pojavijo. Med razvojem programov smo veliko napak tudi simulirali in s tem ugotavljali učinkovitost programov.

Nadaljnje delo na tem področju bo vsebovalo razširitev diagnostike na dodatne module mikroračunalnika EPM-850 in, kot novost, raziskave uporabnosti diagnostike pri povečanju zanesljivosti in razpoložljivosti računalniških mrež.

LITERATURA:

1. J.Alunkal: Diagnostic Design Principles for Computer Systems: A Self-Testing Perspective (Electronics Test okt.1982)
2. Robach C.: Microprocessor Systems Testing - a reivew and future prospects (CDI 3/4-1981)
3. Mrdaković D.: Razvoj periferne mikroračunarske stanice PMS-850 (Magistersko delo - Beograd 1985)
4. S.M.Thatte, J.A.Abraham: Test Generation for Microprocessor (IEEE Transactions on Computers, June 1980)
5. Kastelic B.: Občasna diagnostika mikroračunalniških sistemov (Magistersko delo - Ljubljana)
6. Petković D.: Testiranje RAM i (E)PROM memorije u mikroračunarskim sistemima (Elektrotehnika 9, 1981)
7. P.P.Fasang: Microbit brings self-testing on board complex microcomputers (Electronics, March 1983)

NORMALNE FORME U RELACIONIM BAZAMA PODATAKA: LOGIČKE OSNOVE

INFORMATICA 1/87

UDK 681.3.068

Mirko Maleković
VVTŠ KoV, Zagreb

U ovom radu tretiramo normalne forme (2NF, 3NF, BCNF, 4NF) u relacionim bazama podataka. Primjenom rezolucijske procedure rješavamo implikacione probleme o odnosu normalnih formi.

NORMAL FORMS IN RELATIONAL DATABASES: A LOGICAL BASIS
In this work we have presented proofs of rules about relationship between normal forms (2NF, 3NF, BCNF, 4NF) in relational databases. The proofs are based on application of resolution proof procedures.

0. Uvod

U izboru dobrog skupa relacionih šema trebamo respektirati zahtjev za nepostojanjem anomalija unošenja, brisanja i ažuriranja. Normalne forme su takve dekompozicije relacione šeme kod kojih se izbjegavaju navedene anomalije. Proces normalizacije se bazira na zavisnostima funkcionalnog i višeznačnog tipa [1], [2], [4], [5]. Sadržaj članka je kao što slijedi.
U sekciji 1. karakteriziramo 2NF i 3NF, te rješavamo implikacioni problem o odnosu navedenih formi (Teorem 1.).
Sekcija 2. predstavlja karakterizaciju BCNF uz rješenje implikacionog problema o odnosu BCNF i 3NF (Teorem 2.).
U sekciji 3. uvodimo 4NF, a zatim rješavamo implikacioni problem o odnosu 4NF i BCNF (Teorem 3.). Metodologija rješavanja navedenih implikacionih problema je bazirana na rezolucijskim procedurama dokazivanja, [3].

1. 2NF, 3NF: Odnos izmedu 2NF i 3NF

Postupak normalizacije se bazira na zavisnostima funkcionalnog i višeznačnog tipa. Zato, u specificiranju relacione šeme, osim konačnog skupa R, moramo zadati i skup zavisnosti Σ, koje vrijede u R. U ovom radu usvajamo slijedeću notaciju.
R - konačan skup atributa.
Σ - skup funkcionalnih i višeznačnih zavisnosti koje vrijede u R.
Σ^+ - zatvarač od tj. skup svih zavisnosti koje su logička konzekvenca od Σ.
Sada imamo definiciju.

Definicija. Ureden par (R, Σ), u oznaci R_Σ, zovemo relacionom šemom.

Činjenicu da je $X \rightarrow Y \in \Sigma^+$ iskazujemo i tako da kažemo da $X \rightarrow Y$ vrijedi (u R_Σ). Isto se odnosi i na višeznačnu zavisnost $X \twoheadrightarrow Y$.
Sa $\sim(X \rightarrow Y)$, odnosno $\sim(X \twoheadrightarrow Y)$, naznačavamo da $X \rightarrow Y$, odnosno $X \twoheadrightarrow Y$, ne vrijedi (u R_Σ).
Koristeći pojam funkcionalne zavisnosti uvodimo pojam ključa.

Definicija. Neka je zadana relaciona šema R_Σ, gdje je $R = \{A_1, \ldots, A_n\}$. Za $X \subseteq R$ kažemo da je ključ od R_Σ ako i samo ako vrijedi
1) $X \rightarrow A_1 \ldots A_n$ i
2) $\forall X' \subseteq R \; (X' \subset X \Rightarrow \sim(X' \rightarrow A_1 \ldots A_n))$
Zahtjev 1) iz definicije ključa obezbjeduje da vrijednost atributa iz X jedinstveno odreduju vrijednosti svih atributa iz R, a zahtjev 2) daje minimalnost ključa (ključ je, dakle, minimalan identifikator entiteta predstavljeni relacijom za R.
Napomenimo da relaciona šema R_Σ može imati više od jednog ključa. U tom slučaju naznačava se jedan kao primarni ključ. Sa $\mathcal{K}(R_\Sigma)$ označavamo skup ključeva za R_Σ.
U daljem pretpostavljamo da se relaciona šema R_Σ nalazi u 1NF tj. da su domene atributa iz R proste.

Da bismo uveli pojam 2NF, trebamo karakterizirati potpunu funkcionalnu zavisnost.

Definicija. Neka je zadana relaciona šema R_Σ. Kažemo da je funkcionalna zavisnost $X \rightarrow Y \in \Sigma^+$ potpuna ako i samo ako vrijedi

3) $\forall X' \subseteq R\ (X' \subset X \Rightarrow \sim(X' \rightarrow Y))$.

Za funkcionalnu zavisnost $X \rightarrow Y \in \Sigma^+$ kažemo da je parcijalna ako i samo ako nije potpuna tj. ako i samo ako vrijedi

4) $\exists X' \subseteq R\ (X' \subset X \wedge X' \rightarrow Y)$.

Uočimo da se parcijalna zavisnost $X \rightarrow Y$ može reprezentirati kao kompozicija $X \rightarrow X' \rightarrow Y$.

Definicija. Za atribut $A \in R$ kažemo da je primarni atribut ako i samo ako je ispunjeno
$\exists K \in \mathcal{K}(R)\ (A \in K)$.
U protivnome, za A kažemo da je neprimarni atribut.

Kako postojanje parcijalnih zavisnosti neprimarnih atributa od ključa u R uzrokuje anomalije (upisivanja, brisanja i ažuriranja), 2NF ih eksplicite zabranjuje. Zato imamo slijedeću definiciju.

Definicija. Relaciona šema R_Σ je u 2NF ako i samo ako svaki neprimarni atribut potpuno zavisi od svakog ključa tj. ako i samo ako vrijedi

5) $\forall A \in R\, \forall K \in \mathcal{K}(R_\Sigma)$ (A je neprimarni $\Rightarrow$ $K \rightarrow A$ je potpuna zavisnost).

Ako uvedeno predikate PRIM i POT, gdje PRIM (A) odnosno $POT(X \rightarrow Y)$ znači da je A primaran, odnosno da je $X \rightarrow Y$ potpuna zavisnost, 5) možemo zapisati u obliku

6) $\forall A \in R\ \forall K \in \mathcal{K}(R_\Sigma)(\sim PRIM(A) \Rightarrow POT(K \rightarrow A))$,

što je ekvivalentno sa

7) $\sim \exists A \in R\ \exists K \in \mathcal{K}(R_\Sigma)\ (\sim PRIM(A) \wedge \sim POT(K \rightarrow A))$

Iz definicije 2NF proizlazi da, ako su svi atributi iz R primarni, onda je R_Σ u 2NF. Također, ako je svaki ključ jednočlan, onda je R_Σ u 2NF. Relaciona šema može biti u 2NF, a da i dalje postaje navedene anomalije. To govori da definicijski zahtjev za 2NF nije dovoljno strog. Pojačanjem zahtjeva(specificiranjem nepostojanja tranzitivne zavisnosti neprimarnog atributa od bilo kojeg ključa) dolazimo do 3NF. Karakterizacija tranzitivne zavisnosti je kao što slijedi.

Definicija. Neka su $X, Y, Z \subseteq R$. Kažemo da Z tranzitivno zavisi od X ako i samo ako vrijedi

8) $X \rightarrow Y$

9) $Y \rightarrow Z$ (netrivijalno)

10) $\sim (Y \rightarrow X)$

Svojstvo 9) tj. netrivijalnost funkcionalne zavisnosti $Y \rightarrow Z$ znači da nije $Z \subseteq Y$.
Sada karakteriziramo 3NF.

Definicija. Relaciona šema R_Σ je u 3NF ako i samo ako ne postoji tranzitivna zavisnost bilo kojeg neprimarnog atributa od bilo kojeg ključa. Simbolički, R_Σ je u 3NF ako i samo ako vrijedi

11) $\sim \exists A \in R\, \exists K \in \mathcal{K}(R_\Sigma)(\sim PRIM(A) \wedge TRANZ(K \rightarrow A))$.

U formuli 11) $TRANZ(K \rightarrow A)$ znači da je $K \rightarrow A$ tranzitivna zavisnost.
U daljem ćemo varijable A, X, K interpretirati na skupovima R, $\mathcal{P}(R)$, $\mathcal{K}(R_\Sigma)$ respektivno; također, sa $2NF(R_\Sigma)$, $3NF(R_\Sigma)$, $BCNF(R_\Sigma)$ i $4NF(R_\Sigma)$ ćemo naznačiti da se R_Σ nalazi u odgovarajućoj normalnoj formi.
U odnosu 2NF i 3NF govori slijedeći teorem.

Teorem 1. $3NF(R_\Sigma) \Rightarrow 2NF(R_\Sigma)$.

Za dokaz teorema 1. trebamo riješiti implikacioni problem

T1: $\dfrac{3NF(R_\Sigma)}{2NF(R_\Sigma)}$

Standardizacijom $3NF(R_\Sigma) \wedge \sim 2NF(R_\Sigma)$ nalazimo skup S:

$SSF(3NF(R_\Sigma))$: (1) $PRIM(A) \vee \sim TRANZ(K \rightarrow A)$

$SSF(\sim 2NF(R_\Sigma))$: (2) $\sim PRIM(A_o)$
(3) $\sim POT(K_o \rightarrow A_o)$

Skup rečenica $S = \{(1),(2),(3)\}$ proširujemo pravilom

(P): $\forall A\, \forall K\ (\sim POT(K \rightarrow A) \Rightarrow TRANZ(K \rightarrow A))$.

Standardizacijom pravila (P) dobivamo rečenicu

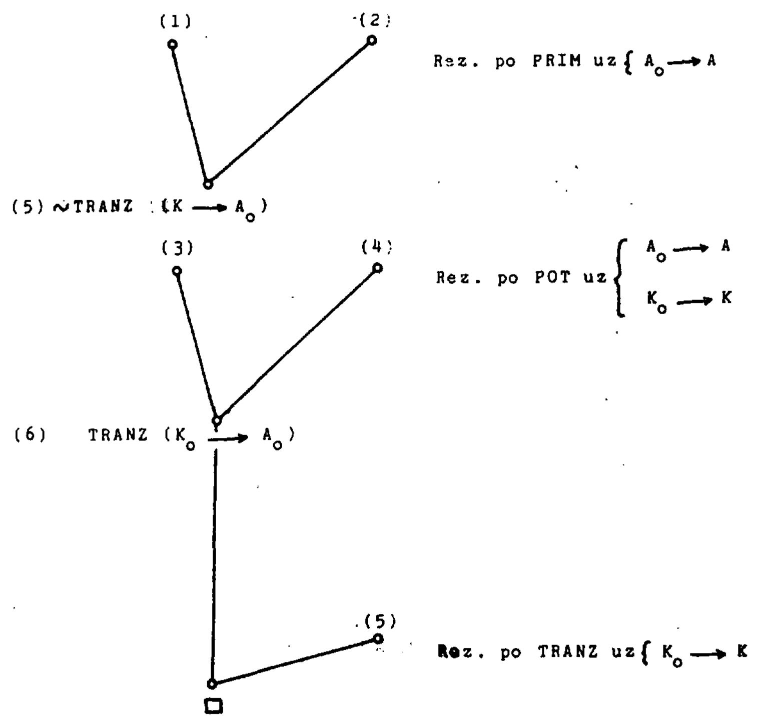
(4) $POT(K \rightarrow A) \vee TRANZ(K \rightarrow A)$.

Kontradiktornost skupa $S' = S \cup \{(4)\}$ dana je na slici 1., a time je implikacioni problem T1 riješen.

2. BCNF:
Odnos između 3NF i BCNF

Niti 3NF-normalizacija relacione šeme R_Σ ne garantira nepostojanje anomalija (upisivanja, brisanja i ažuriranja).
Pojačanjem zahtjeva(definicijskog),specificiranjem sa domena netrivijalne zavisnosti sadrži ključ, dolazimo do BCNF(Boyce-Codd normalna forma). Ponovimo, da za zavisnost $X \rightarrow Y$ kažemo da je trivijalna ako i samo ako je $Y \subseteq X$.
Slijedi precizna definicija BCNF.

Definicija. Za relacionu šemu R_Σ kažemo da je u BCNF ako i samo ako za svaku zavisnost $X \rightarrow Y$ iz Σ^+ vrijedi


(1)
(2)
Rez. po PRIM uz { $A_o \to A$
(5) ~TRANZ (K → A_o)
(3)
(4)
Rez. po POT uz { $A_o \to A$, $K_o \to K$
(6) TRANZ ($K_o \to A_o$)
(5)
Rez. po TRANZ uz { $K_o \to K$

Sl.1.

12) $\sim TRIV(X \to Y) \Rightarrow \exists K \ (K \subseteq X))$.

Da je definicijski zahtjev za BCNF stroži nego za 3NF govori slijedeći teorem.

Teorem 2. $BCNF(R_\Sigma) \Longrightarrow 3NF(R_\Sigma)$.

Iskazani teorem dokazujemo rješavajući implikacioni problem

T2: $BCNF(R_\Sigma)$
$3NF(R_\Sigma)$

Transformirajući $BCNF(R_\Sigma) \wedge \sim 3NF(R_\Sigma)$ u standardnu formu dobivamo skup rečenica S. Prvo standardiziramo $BCNF(R_\Sigma)$.

Iz 12) nalazimo ekvivalentnu formulu

13) $\forall X \ \forall Y \ \exists K \ (TRIV(X \to Y) \vee (K \subseteq X))$.

Eliminacijom egzistencijalnog kvantifikatora, uvodeći umjesto K Skolemovu funkciju f(X,Y), dobivamo

$SSF(BCNF(R_\Sigma))$:

(1) $TRIV(X \to Y) \vee (f(X,Y) \subseteq X)$.

Predimo na standardizaciju od $\sim 3NF(R_\Sigma)$. Negiranjem formule 11), te standardizacijom imat ćemo

$SSF(\sim 3NF(R_\Sigma))$: (2) $\sim PRIM(A_o)$
(3) $TRANZ(K_o \to A_o)$

Rečenica (3) je ekvivalentna sa konjunkcijom rečenica (definicija tranzitivne zavisnosti).

(4) $K_o \to Y_o$

(5) $Y_o \to A_o$

(6) $\sim (Y_o \quad K_o)$

(7) $\sim TRIV(Y_o \to A_o)$

Konačno, S={(1),(2),(4),(6),(7)}.

Skup S proširujemo pravilom koje karakterizira ključ

(K): $\forall K \ \forall X \ (K \to X)$.

Standardizacijom iz (K) dobivamo rečenicu

(8) $K \to X$.

Kontradiktornost skupa $S' = S \cup \{(8)\}$ je dana na slici 2.

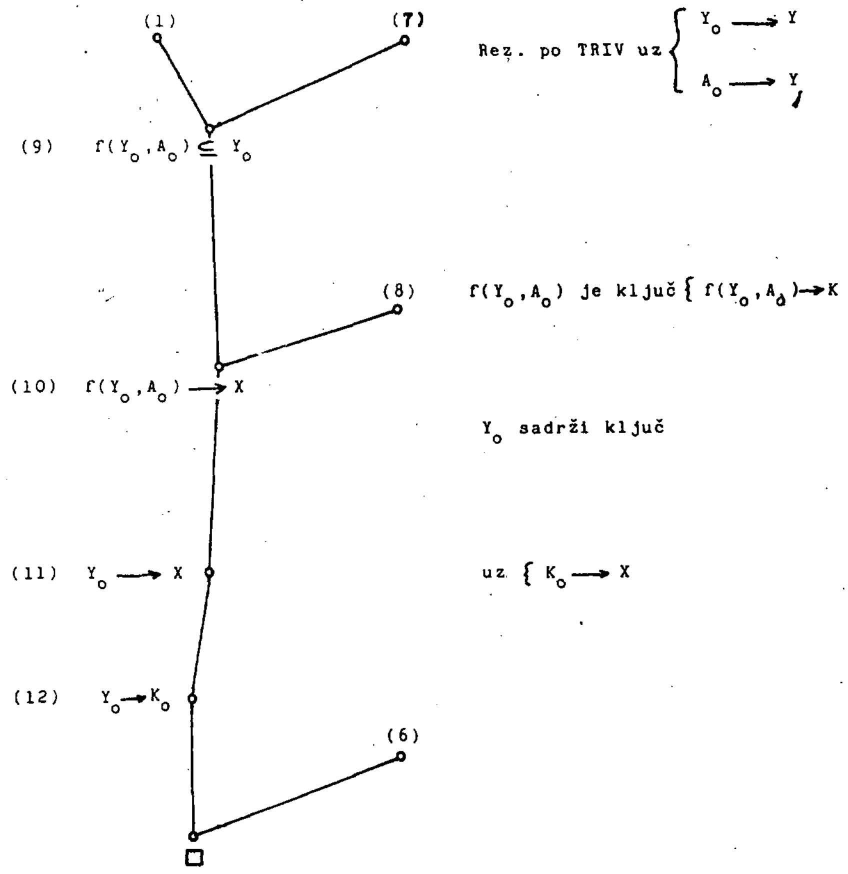
3. 4NF:
Odnos izmedu BCNF i 4NF

Može se pokazati da niti BCNF-normalizacija ne garantira nepostojanje anomalija. Zato uvodimo 4NF, čiji je definicijski zahtjev jači od definicijskog zahtjeva za BCNF. Dok su 2NF, 3NF, BCNF definirani preko funkcionalne zavisnosti, u definiciji 4NF koristimo višeznačnu zavisnost. Pri tome imamo slijedeću definiciju.

Definicija. Neka su $X, Y \subseteq R$. Za višeznačnu zavisnost $X \twoheadrightarrow Y$ kažemo da je trivijalna ako i samo ako je $Y \subseteq X \vee XY = R$.

Sa $TRIVV(X \twoheadrightarrow Y)$ ćemo označavati da je višeznačna zavisnost trivijalna. Karakterizacija 4NF je kao što slijedi.

Definicija. Kažemo da je relaciona šema R_Σ u 4NF ako i samo ako za svaku zavisnost $X \twoheadrightarrow Y$ iz


(1)
(7)
Rez. po TRIV uz { $Y_o \rightarrow Y$, $A_o \rightarrow Y$ }
(9) $f(Y_o,A_o) \subseteq Y_o$
(8)
$f(Y_o,A_o)$ je ključ { $f(Y_o,A_o) \rightarrow K$
(10) $f(Y_o,A_o) \rightarrow X$
Y_o sadrži ključ
(11) $Y_o \rightarrow X$
uz { $K_o \rightarrow X$
(12) $Y_o \rightarrow K_o$
(6)
$\square$

Sl.2.

Σ^{+} vrijedi
13) $\sim TRIVV(X \twoheadrightarrow Y) \Longrightarrow \exists K\ (K \subseteq X)$.

Definicija 13) nam kaže da je R_{Σ} u 4NF ako i samo ako svaka netrivijalna zavisnost, koja vrijedi u R_{Σ}, ima domenu koja sadrži ključ. U slijedećem teoremu govorimo o odnosu BCNF i 4NF.

Teorem 3. $4NF(R_{\Sigma}) \Longrightarrow BCNF(R_{\Sigma})$.

Teorem 3. dokazujemo rješavajući implikacioni problem

T3: $\dfrac{4NF(R_{\Sigma})}{BCNF(R_{\Sigma})}$,

Standardne forme za $4NF(R_{\Sigma})$ i $\sim BCNF(R_{\Sigma})$ su kao što slijedi.

$SSF(4NF(R_{\Sigma}))$:

(1) $TRIVV(X \twoheadrightarrow Y) \vee (f(X,Y) \subseteq X)$.

U (1) $f(X,Y)$ je ključ koji odgovara zavisnosti $X \twoheadrightarrow Y$ (u procesu eliminacije egzistencijalnog kvantifikatora).

Negacijom 13) dobivamo $\sim BCNF(R_{\Sigma})$;

14) $\sim BCNF(R_{\Sigma})$:

$\exists X\ \exists Y\ \forall K\ (\sim TRIV(X \rightarrow Y) \wedge \sim(K \subseteq X))$.

Standardizacijom, iz 13), nalazimo $SSF(\sim BCNF(R_{\Sigma}))$:

(2) $\sim TRIV(X_o \rightarrow Y_o)$

(3) $\sim (K \subseteq X_o)$.

Za kompletiranje rješenja implikacionog problema trebamo pravilo

(P1): $\forall K \forall X \forall Y ((\sim TRIV(X \rightarrow Y) \wedge \sim(K \subseteq X)) \Longrightarrow$
$\Longrightarrow \sim TRIVV(X \twoheadrightarrow Y))$.

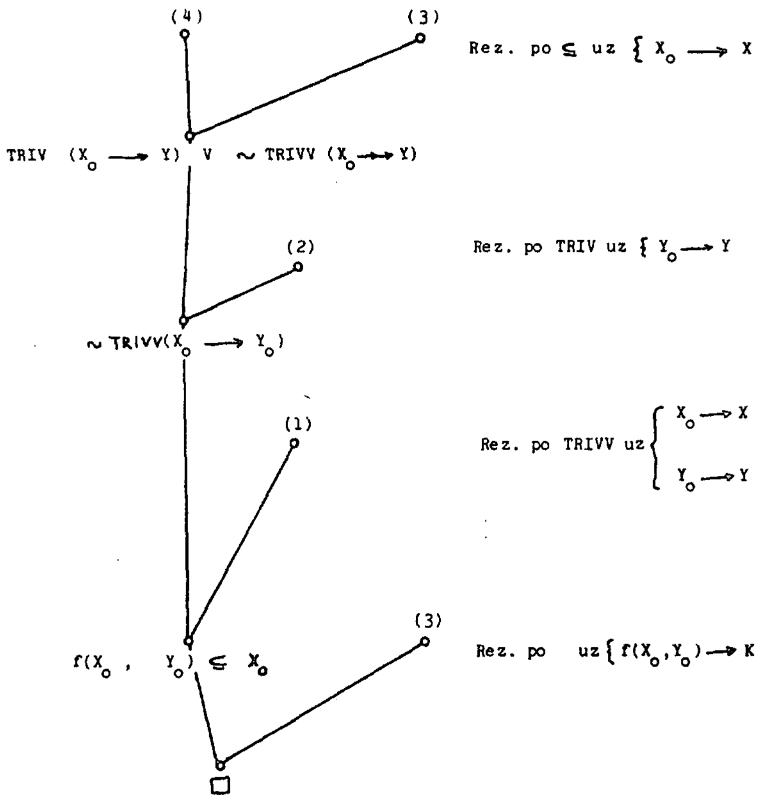
Standardizacijom iz (P1) dobivamo rečenicu

(4) $TRIV(X \rightarrow Y) \vee (K \subseteq X) \vee \sim TRIVV(X \twoheadrightarrow Y)$.

Stablo dokaza kontradiktornosti skupa $S = \{(1), (2), (3), (4)\}$ dajemo na slici 3.

4. Zaključak

U ovom radu razmatrali smo normalne forme u relacionim bazama podataka. Karakterizirali smo 2NF, 3NF, BCNF, 4NF, a zatim, primjenom rezolucijske procedure, riješili implikacione probleme o odnosu navedenih normalnih formi (Teoremi 1., 2., i 3.).



Sl.3.

Literatura

1. Alagić, S.:Relacione baze podataka. "Svijetlost" Sarajevo, 1984.

2. Armstrong, W.W., Delobel, C.:Decompositions and functional dependencies in relations. ACM Trans. Database Syst. 5,4(Dec,1980),

3. Chang,C.L. and Lee, R.C.T.: Symbolic Logic and Mechanical Theorem Proving. Compt. Sci. Appl.Math Academic Press,1973.

4. Paredaens, J. and Jannsens, D.: Decompositions of relations- a comprehensive approach. In Advances in Database Theory, H.Gallaire, J.Minker, and J.M.Nicolas, Eds. Plenum Press, New York, 1981,73-100.

5. Ullman, J.D.: Principles of Data Base Syst., Compt. Sci. Press, Maryland, 1980.

IZBIRA KODNE STRUKTURE V 6. NIVOJU REFERENČNEGA MODELA OSI ZA POTREBE PRENOSA IN IZMENJAVE PODATKOV

INFORMATICA 1/87

B. Jerman-Blažič
Institut Jožef Stefan
Jamova 39, Ljubljana

UDK 681.324.04

Ključne besede
Referenčni model OSI, nabor grafičnih znakov, nivo predstavitve, implementacija standardov, ISO, CCITT, telematske storitve, odprti sistemi, heterogene računalniške mreže, računalniški sistemi.

Povzetek
Referenčni model OSI daje teoretično in praktično podlago za izgradnjo heterogenih mrež, v katere se vključujejo najrazličnejši računalniški sistemi in nudijo različne telematske storitve. Sesti nivo modela OSI opredeljuje predstavitev podatkov. V zadnjih dveh letih sta bila za ta nivo izdelana in sprejeta dva mednarodna standarda za nabor grafičnih znakov kodiranih z 8 biti. Problematika, ki je nastala v zvezi z implementacijo teh standardov v okviru RM OSI še ni razrešena. Prispevek obravnava nastale probleme in ponuja nekatere rešitve v odvisnosti od področja implementacije.

Key words
OSI Reference Model, Graphic character sets, presentation level, standard implementation, ISO, CCITT, Telematic services, open systems, heterogeneous computer networks, computer systems.

"Choice of the Coded Character Set for Text Communication in the Presentation Level of RM OSI"

Abstract
The ISO reference model for open system interconnection is a theoretical framework for development of heterogeneous computer networks with implemented telematic services. The sixth level of the model is the presentation level, responsible for data presentation. The paper deals with the problems which arised after the adoption of two international 8bit coded character sets for text communication within the international bodies for standardization. Some guidelines for implementation of the standards in connection with the application field are presented too.

0.Uvod

Problem kodiranja nabora znakov sega v začetek razvoja računalništva in njemu sorodnih tehnik. Kodiranje in kodne strukture so bile pred pojavom računalniške tehnike predmet telegrafije. Z razvojem računalniške tehnike je nastala potreba za kodiranje grafičnih in krmilnih znakov. Tako so bile postopoma uvajane 6, 7 in 8-bitne kode v odvisnosti od stopnje uvedene računalniške tehnologije. Razvoj kodnih struktur je danes skupni problem računalništva, telekomunikacij in njihovega otroka - telematike. Pri tem imamo v mislih nove telematske storitve (teletex, videotex, telekonferenco ipd.) in široko področje različnih računalniških mrež (LAN, VAN, PSDT, PBX ipd.) ter distribuirane sisteme za obdelavo podatkov.

Razvoj kodnih struktur je treba opazovati iz dveh zornih kotov: kot kodne strukture, ki se uporabljajo znotraj zaprtih računalniških sistemov in kodne strukture, ki se uporabljajo za prenos in izmenjavo podatkov. Sedanji trendi razvoja distribuiranih sistemov, pri katerih ima ključni pomen prenos in izmenjava podatkov, se gibljejo v smeri izgradnje v skladu z referenčnim modelom OSI mednarodne organizacije za standardizacijo.

Model OSI daje teoretično in praktično podlago za izgradnjo heterogenih mrež, v katere se vključujejo najrazličnejši računalniški sistemi in nudijo različne telematske storitve. Sesti nivo modela OSI opredeljuje predstavitev podatkov. Ta nivo predpostavlja, da je prenos podatkov opravljen brez napake. V nadaljevanju prispevka bo obravnavana problematika izbora kodiranja (dodeljevanja odgovarajoče kombinacije bitov grafičnim in krmilnim znakom iz nekega vnaprej dogovorjenega repertoarja) in dekodiranja (predstavitev sprejetih znakov) v 6. nivoju modela OSI.

1.Razvoj 8-bitnih kod za potrebe prenosa in izmenjave podatkov

Izbor in definicija kodnih struktur za potrebe prenosa in izmenjave podatkov je predmet dela SC2 (podkomiteja 2) Mednarodne organizacije za standardizacijo in VIII študijske skupine CCITT-ja. Razvoj prvih standardov za 8-bitne kode sega v konec osemdesetih let. Prva standardizirana 8-bitna kodna tabela za potrebe prenosa podatkov je bila izdelana konec osemdesetih let. Standard je dobil naslov "Nabor grafičnih znakov kodiranih z 8 biti za potrebe izmenjave podatkov ISO 6937". Standard je nastal zaradi potreb CCITT po standardiziranem naboru grafičnih znakov za latinično pisavo evropskih jezikov, ki se je uporabljal pri prenosu in prezentaciji podatkov v okviru teletexa in videotexa. CCITT je objavil predlog standarda v obliki priporočila S.61-1981 oz. T.61-1984 (za potrebe storitve teletexa) in S.101 in T.101 - 1984 (za potrebe videotexa).

Kodna tabela ISO 6937, ki je v osnovi identična s T.61 in T.101, vsebuje dva nabora grafičnih znakov. Na levi strani kodne tabele se nahaja mednarodna referenčna verzija nabora grafičnih znakov, bolj znana po svetu kot ISO 646, oziroma 7-bitni ASCII, na desni strani kodne tabele so diakritični znaki, s katerimi je možno generiranje akcentiranih latiničnih znakov. Diakritični znaki so znaki brez pomika naprej. Vsaka akcentirana črka latinice se generira kot dvozlogovni znak, ki na terminalni napravi zavzame eno mesto. Kodno tabelo ISO 6937 smo ponazorili na sliki 1. Tehnika, definirana v T.61 in T.101, omogoča predstavitev repertoarja grafičnih znakov, ki je po številu veliko večji od nabora znakov iz tabele 1. ISO 6937 omogoča predstavitev 350 grafičnih znakov. S tem standardom je CCITTa rešil problem prenosa grafičnih znakov za potrebe zahodnoevropskih in latinskoameriških držav. Veliki proizvajalci računalniške opreme, zlasti IBM, so po objavi standarda ISO 6937 sklenili, da potrebujejo enozlogovno 8-bitno ASCII kodo. Na spodbudo IBM-a je ISO TC97/SC2 razvil ISO 8859, v katerem je vsak znak kodiran z enim zlogom. Osnovna kodna tabela tega osemdelnega standarda je prikazana na sliki 2. Kodna tabela, prikazana na sliki 2, skupaj z definiranim naborom krmilnih znakov C0 in C1, je bila sprejeta tudi v ANSI kot 8-bitni ASCII (1.1985) in je znana v svetu pod imenom Latinica 1. Enako kodno tabelo z dodatnimi nabori krmilnih znakov je sprejela tudi ECMA kot standard ECMA-94. Tabela na sliki 2 nam kaže prvi del ISO 8859, ki je namenjen govornemu področju 44 dežel (za angleščino, španščino, nemščino, francoščino, portugalščino, flamščino in italijanščino).

				b8	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
				b7	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
				b6	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
				b5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
b4	b3	b2	b1		00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0	0	0	0	00	▨	▨	▨	0	@	P	`	p	▨	▨	▨	°	See 4.3.4	—	Ω	ĸ
0	0	0	1	01	▨	▨	!	1	A	Q	a	q	▨	▨	¡	±	`	¹	Æ	æ
0	0	1	0	02	▨	▨	"	2	B	R	b	r	▨	▨	¢	²	´	®	Đ	đ
0	0	1	1	03	▨	▨	#	3	C	S	c	s	▨	▨	£	³	ˆ	©	ª	ð
0	1	0	0	04	▨	▨	¤	4	D	T	d	t	▨	▨	$	×	˜	™	Ħ	ħ
0	1	0	1	05	▨	▨	%	5	E	U	e	u	▨	▨	¥	µ	¯	♪	See 4.3.4	ı
0	1	1	0	06	▨	▨	&	6	F	V	f	v	▨	▨	See 4.3.3	¶	˘	See 4.3.4	Ĳ	ĳ
0	1	1	1	07	▨	▨	'	7	G	W	g	w	▨	▨	§	·	˙	See 4.3.4	Ŀ	ŀ
1	0	0	0	08	▨	▨	(	8	H	X	h	x	▨	▨	See 4.3.3	÷	¨	See 4.3.4	Ł	ł
1	0	0	1	09	▨	▨	)	9	I	Y	i	y	▨	▨	‘	’	See 4.3.4	See 4.3.4	Ø	ø
1	0	1	0	10	▨	▨	*	:	J	Z	j	z	▨	▨	“	”	˚	See 4.3.4	Œ	œ
1	0	1	1	11	▨	▨	+	;	K	[	k	{	▨	▨	«	»	¸	See 4.3.4	º	ß
1	1	0	0	12	▨	▨	,	<	L	\	l	|	▨	▨	←	¼	_	⅛	Þ	þ
1	1	0	1	13	▨	▨	-	=	M	]	m	}	▨	▨	↑	½	˝	⅜	Ŧ	ŧ
1	1	1	0	14	▨	▨	.	>	N	^	n	~	▨	▨	→	¾	˛	⅝	Ŋ	ŋ
1	1	1	1	15	▨	▨	/	?	O	_	o	▨	▨	▨	↓	¿	ˇ	⅞	ŉ	▨

Sl. 1. ISO 6937/2

Ostali deli ISO 8859 z izjemo 8859/2 so v fazi DIS (draft iternational standard) ali DP (draft proposal). 8859/2 definira latinico št.2, ki je namenjena prenosu teksta, napisanega v enem od naslednjih jezikov: albanskem, češkoslovaškem, nemškem, madžarskem, poljskem, romunskem in slovenskem (oziroma srbohrvaškem). Enako kodno tabelo, kot je definirana v ISO 8859/2, je sprejela tudi Komisija za kode in šifrirne sisteme pri Zveznem zavodu za standardizacijo kot JUS I.B1.013. 8859/3 pokriva govorno področje Francije, Nemčije, Italije, Malte, Južne Afrike in Turčije. 8859/4 pa je namenjena Danski, Finski, Nemčiji, Grenlandiji, Norveški, Švedski in sovjetskim republikam Estoniji, Latviji in Litvaniji. Tem kodnim tabelam je skupno to, da ima v primerih, ko se ista črka pojavlja v dveh različnih kodnih tabelah, zmeraj isto mesto, kar pomeni tudi isto kodno kombinacijo (tako se črka Ž pojavlja v 8859/2 in 8859/4 na istem mestu in ima kodno kombinacijo 10/14).

Leva stran kodne tabele je povsod enaka t.j. ISO 646 ali 7-bitni ASCII. 8859/5 definira kodno tabelo za nabor znakov, ki se uporabljajo v tehničnih in matematičnih znanostih (razni specialni znaki za pisanje matematičnih in tehničnih izrazov oz. formul). 8859/6 definira kodno tabelo za nabor znakov, ki se uporabljajo

				b8	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
				b7	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
				b6	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
				b5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
b4	b3	b2	b1		00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0	0	0	0	00	▨	▨	SP	0	@	P	`	p	▨	▨	NBSP	°	À	Ð	à	ð
0	0	0	1	01	▨	▨	!	1	A	Q	a	q	▨	▨	¡	±	Á	Ñ	á	ñ
0	0	1	0	02	▨	▨	"	2	B	R	b	r	▨	▨	¢	²	Â	Ò	â	ò
0	0	1	1	03	▨	▨	#	3	C	S	c	s	▨	▨	£	³	Ã	Ó	ã	ó
0	1	0	0	04	▨	▨	$	4	D	T	d	t	▨	▨	¤	´	Ä	Ô	ä	ô
0	1	0	1	05	▨	▨	%	5	E	U	e	u	▨	▨	¥	µ	Å	Õ	å	õ
0	1	1	0	06	▨	▨	&	6	F	V	f	v	▨	▨	¦	¶	Æ	Ö	æ	ö
0	1	1	1	07	▨	▨	'	7	G	W	g	w	▨	▨	§	·	Ç	×	ç	÷
1	0	0	0	08	▨	▨	(	8	H	X	h	x	▨	▨	¨	¸	È	Ø	è	ø
1	0	0	1	09	▨	▨	)	9	I	Y	i	y	▨	▨	©	¹	É	Ù	é	ù
1	0	1	0	10	▨	▨	*	:	J	Z	j	z	▨	▨	ª	º	Ê	Ú	ê	ú
1	0	1	1	11	▨	▨	+	;	K	[	k	{	▨	▨	«	»	Ë	Û	ë	û
1	1	0	0	12	▨	▨	,	<	L	\	l	|	▨	▨	¬	¼	Ì	Ü	ì	ü
1	1	0	1	13	▨	▨	-	=	M	]	m	}	▨	▨	SHY	½	Í	Ý	í	ý
1	1	1	0	14	▨	▨	.	>	N	^	n	~	▨	▨	®	¾	Î	Þ	î	þ
1	1	1	1	15	▨	▨	/	?	O	_	o	▨	▨	▨	¯	¿	Ï	ß	ï	ÿ

Sl. 2. ISO 8859/1

v založništvu za formatiranje tekstov, 8859/7 definira kode za grško abecedo in 8859/8 definira cirilico za ruski, srbski, makedonski, bolgarski, ukrajinski in beloruski jezik. Znaki iz kodnih tabel 8859/5/6/7/8 ne nastopajo v naboru znakov v ISO 6937/1/2 in zato so v pripravi dodatki k ISO 6937, ki naj bi vsebovali manjkajoče znake.

2. Izbor kodne strukture

Po objavi ISO 6937/1/2 in ISO 8859/1/2 je v okviru uradnih mednarodnih teles za standardizacijo, kot je CCITT, ISO in zlasti ANSI prišlo do velikega razburjenja. Implementatorji iz krogov proizvajalcev računalniške in telematske opreme so se znašli pred uganko. Katero 8-bitno kodo implementirati v nivoju 6, za posamezno

				b8	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
				b7	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
				b6	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
				b5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
b4	b3	b2	b1		00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0	0	0	0	00	▨	▨	SP	0	@	P	`	p	▨	▨	NBSP	°	Ŕ	Đ	ŕ	đ
0	0	0	1	01	▨	▨	!	1	A	Q	a	q	▨	▨	Ą	ą	Á	Ń	á	ń
0	0	1	0	02	▨	▨	"	2	B	R	b	r	▨	▨	˘	˛	Â	Ň	â	ň
0	0	1	1	03	▨	▨	#	3	C	S	c	s	▨	▨	Ł	ł	Ă	Ó	ă	ó
0	1	0	0	04	▨	▨	$	4	D	T	d	t	▨	▨	¤	´	Ä	Ô	ä	ô
0	1	0	1	05	▨	▨	%	5	E	U	e	u	▨	▨	Ľ	ľ	Ĺ	Ő	ĺ	ő
0	1	1	0	06	▨	▨	&	6	F	V	f	v	▨	▨	Ś	ś	Ć	Ö	ć	ö
0	1	1	1	07	▨	▨	'	7	G	W	g	w	▨	▨	§	ˇ	Ç	×	ç	÷
1	0	0	0	08	▨	▨	(	8	H	X	h	x	▨	▨	¨	¸	Č	Ř	č	ř
1	0	0	1	09	▨	▨	)	9	I	Y	i	y	▨	▨	Š	š	É	Ů	é	ů
1	0	1	0	10	▨	▨	*	:	J	Z	j	z	▨	▨	Ş	ş	Ę	Ú	ę	ú
1	0	1	1	11	▨	▨	+	;	K	[	k	{	▨	▨	Ť	ť	Ë	Ű	ë	ű
1	1	0	0	12	▨	▨	,	<	L	\	l	|	▨	▨	Ź	ź	Ě	Ü	ě	ü
1	1	0	1	13	▨	▨	-	=	M	]	m	}	▨	▨	SHY	˝	Í	Ý	í	ý
1	1	1	0	14	▨	▨	.	>	N	^	n	~	▨	▨	Ž	ž	Î	Ţ	î	ţ
1	1	1	1	15	▨	▨	/	?	O	_	o	▨	▨	▨	Ż	ż	Ď	ß	ď	˙

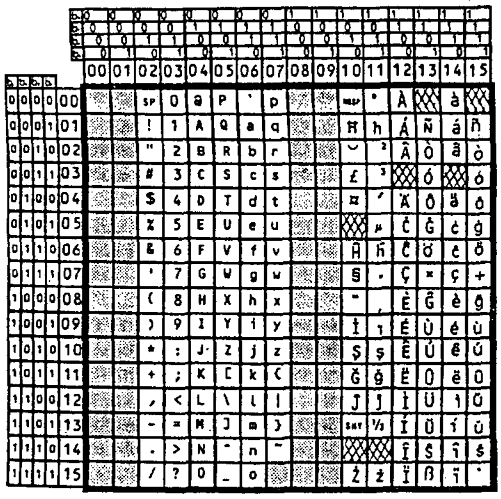
Sl. 2. ISO 8859/2

aplikacijo.Znano je, da učinkovita komunikacija in izmenjava podatkov v odprtih sistemih povezovanja ni mogoča brez enotnih in standardiziranih kod. Odstavki, ki sledijo obravnavajo problem izbora kodne strukture in stališča bodočih uporabnikov oziroma proizvajalcev. Odločitve, ki bodo sprejete v svetu po tem vprašanju, bodo vplivale tudi na razvoj telematskih storitev in izdelkov v Jugoslaviji (kot so na primer terminali za teletex, videotex, večjezične tipkovnice terminalov za vnos podatkov ter različne aplikacije v okviru JUPAKa).

2.1 Nekatera znana dejstva

ISO 6937 je aplikacijsko orientiran standard, ki poleg grafičnih znakov za pisanje teksta tudi definira tudi krmilne funkcije. ISO 6937 je izdelan za potrebe nivoja predstavitve pri prenosu teksta za aplikacije kot so Teletex in Videotex. Za te storitve so v uporabi posebej razviti terminali, ki omogočajo sprejem in predstavitev sprejetih podatkov. Sprejeti podatki se prikazujejo na zaslonu ali pa na tiskalniku. Posebna izbira in shranjevanje teh podatkov zaradi nadaljne obdelave ni predvidena. V postopku izdelave je četrti del standarda, ki definira krmilne funkcije za preformatiranje teksta (tistega, ki je že formatiran in tistega, ki ni).

Večina proizvajalcev in uporabnikov je pričakovala, da bo ISO 6837 standard z najbolj splošno uporabo. Ta standard je zadovoljeval potrebe



				b_8	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
				b_7	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
				b_6	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
				b_5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
b_4	b_3	b_2	b_1		00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0	0	0	0	00	▒	▒	SP	0	@	P	`	p	▒	▒	NBSP	°	À	▩	à	▩
0	0	0	1	01	▒	▒	!	1	A	Q	a	q	▒	▒	Ħ	ħ	Á	Ñ	á	ñ
0	0	1	0	02	▒	▒	"	2	B	R	b	r	▒	▒	˘	²	Â	Ò	â	ò
0	0	1	1	03	▒	▒	#	3	C	S	c	s	▒	▒	£	³	▩	Ó	▩	ó
0	1	0	0	04	▒	▒	$	4	D	T	d	t	▒	▒	¤	´	Ä	Ô	ä	ô
0	1	0	1	05	▒	▒	%	5	E	U	e	u	▒	▒	▩	µ	Ċ	Ġ	ċ	ġ
0	1	1	0	06	▒	▒	&	6	F	V	f	v	▒	▒	Ĥ	ĥ	Ĉ	Ö	ĉ	ö
0	1	1	1	07	▒	▒	'	7	G	W	g	w	▒	▒	§	·	Ç	×	ç	÷
1	0	0	0	08	▒	▒	(	8	H	X	h	x	▒	▒	¨	¸	È	Ĝ	è	ĝ
1	0	0	1	09	▒	▒	)	9	I	Y	i	y	▒	▒	İ	ı	É	Ù	é	ù
1	0	1	0	10	▒	▒	*	:	J	Z	j	z	▒	▒	Ş	ş	Ê	Ú	ê	ú
1	0	1	1	11	▒	▒	+	;	K	[	k	{	▒	▒	Ğ	ğ	Ë	Û	ë	û
1	1	0	0	12	▒	▒	,	<	L	\	l	|	▒	▒	Ĵ	ĵ	Ì	Ü	ì	ü
1	1	0	1	13	▒	▒	-	=	M	]	m	}	▒	▒	SHY	½	Í	Ŭ	í	ŭ
1	1	1	0	14	▒	▒	.	>	N	^	n	~	▒	▒	▩	▩	Î	Ŝ	î	ŝ
1	1	1	1	15	▒	▒	/	?	O	_	o	▒	▒	▒	Ż	ż	Ï	ß	ï	˙

Sl. 2. ISO 8859/3

zahodnega dela sveta, ki je istočasno tudi najbolj razviti del sveta. V tem delu sveta se nahaja 90 najmočnejših proizvajalcev računalniške in komunikacijske opreme ter največje število uporabnikov sodobnih telematskih storitev. Kmalu po sprejemu ISO 6937 se je izkazalo, da rešitev, ki jo določa standard, ni sprejemljiva za sisteme za obdelavo podatkov, ker se po tem standardu nekateri od grafičnih znakov prenašajo kot dvozlogovne informacije, nekateri pa kot enozlogovne. Takšen način prenosa je nesprejemljiv pri prenosu podatkov za nadaljno obdelavo, kot je na primer prenos datotek (file transfer). Razprava v zvezi z ISO 6937 se je začela po opozorilu ISO TC97/SC2/WG4, da je tehnika uporabe grafičnih znakov z različno dolžino kode (eno in dvozlogovno) za večino prevajalnikov programskih jezikov nesprejemljiva. Večina obstoječe programske opreme je narejena po principu en znak en zlog.

Po sprejetju teh pripomb se je CCITT javno ogradil od splošne uporabnosti ISO 6937 in izjavil, da je bil ISO 6937 razvit in namenjen le za prenos in ne za obdelavo tekstovnih podatkov, torej le v sklopu storitev Teletexa in Videotexa. V teh telematskih storitev je tekst zapisan v strogo formatirani obliki (strani), se v taki obliki prenaša in ni namenjen nadaljni obdelavi. Torej se tekst prenaša v končni obliki po omrežju od računalnika (host) do terminala ali od terminala do terminala. Uporabnik tekst le vidi ali pa ga izpiše. Sodobni sistemi za avtomatizacijo pisarniškega poslovanja pa zahtevajo uporabo telematskih storitev, ki omogočajo tudi obdelavo in predelavo sprejetega oziroma odposlanega teksta, kot je preformatiranje, dodajanje in arhiviranje teksta. Mogoče je ravno ta omejenost Videotexa ter izjemno zastarele metode preiskovanja informacij (drevesne strukture, ki so prepočasne), botrovala slabi uveljavitvi te usluge na tleh ZDA. Primerjava obeh standardov nas pripelje do sledečih ugotovitev:

2.2 Prednosti in pomanjkljivosti ISO 6937 in ISO 8859

Gledano iz zornega kota razvitih držav zadostuje za latinično pisavo le nabor grafičnih znakov iz ISO 6937/1/2. To pomeni, da v 8-bitnem okolju, če imamo opravka s čistim tekstom (isto velja tudi za slovensko latinico), ni potrebno preklapljati iz ene kodne tabele v drugo s pomočjo tehnike za razširitev 8-bitnega okolja (za referenco glej ISO 2022 ali ISO 4873 ali I.B1.010 - tehnike razširitve v 7 in 8-bitnem okolju). 8. bit v kodni tabeli nam pokaže, kateri del kodne tabele je v uporabi, levi (8.bit je ničla) ali desni (8.bit je enica). ISO 8859 za prikaz vseh latiničnih pisav sveta potrebuje 4 kodne tabele. V primerih, ko potrebujemo črke vseh 4 jezikovnih področij, je potrebno preklapljanje (switching). Ta trditev drži le do neke meje, ker nekatere od kodnih tabel ISO 8859/1-4/ vsebujejo znake za več kot eno področje oziroma so redundančne. Ta prednost ISO 6937 se takoj spremeni v pomanjkljivost, če bi zahtevali znake iz abeced, ki niso latinične. Tak primer je Jugoslavija, kjer bi potrebovali istočasno latinično in cirilično pisavo. Podobno velja za Grčijo, kjer uporabljajo dve pisavi, latinično in grško. Namen ISO standardov je, da zadosti potrebam večjemu številu dežel in ne le deželam, kjer se uporablja izključno latinična pisava. Torej dolgoročno gledano ta prednost ISO 6937 ne drži.

Druga prednost ISOa 6937 je, da je to standard, ki se že nekaj časa uporablja (sprejet je bil 1.1980), za razliko od ISO 8859/3-8, ki je še v fazi DIS. Ta prednost se bo zgubila v kratkem, ko bo ISO 8859/3-8 postal veljaven standard.

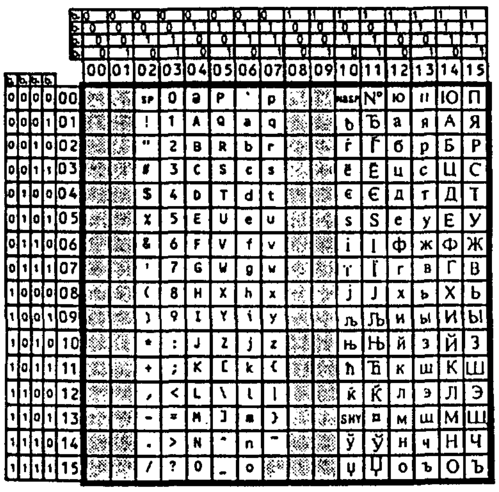
Izbor kodnih tabel v ISO 8859 je narejen po regijah, v katerih se nahajajo sorodne države. Leva kodna tabela je ASCII, kar ne velja za ISO 6937. Večina dežel sveta je svoje nacionalne kodne tabele razvila na podlagi ASCII kodne tabele (ISO 646) z nacionalnimi razširitvami, specificiranimi v ISO 646, mednarodna verzija. Tako je ISO 8859 kompatibilen z veliko večino nacionalnih standardov.

ISO 8859 predstavlja lažji prehod iz obstoječih 7bitnih kod v 8bitno kodo. ISO 8859 predstavlja tudi lažji prehod iz 8-bitnega okolja v 16 bitno okolje (dvozlogovno kodiranje), vsak znak v 16-bitni kodni tabeli bo potreboval za predstavitev le eno 8-bitno kodno kombinacijo. ISO SC2/WG6 trenutno dela na razvoju standardizirane 16-bitne kodne tabele. Zaradi široke uporabe ASCII kodne tabele v veliki večini dežel lahko rečemo, da je ISO 8859 kompatibilen z obstoječo programsko opremo in obstoječimi podatkovnimi bazami.

ISO 8859 je bil že v fazi nastajanja sprejet v večini dežel kot nacionalni standard za nabor znakov, kodiranih z 8bitno kodo, tak primer je tudi Jugoslavija. ISO 6937 ni bil sprejet kot nacionalni standard nikjer, z izjemo ZR Nemčije, ki je kljub temu sprejela še ISO 8859. ISO 8859 omogoča lažje povezovanje (interworking) izdelkov z ASCII in EBCDIC kodo. Izmenjava znakov iz ISO 8859 v okolju z ISO 6937 je možna z uporabo IGS-a (Identity Graphic Sub-repertoar).

Uporaba obeh kodnih sistemov ISO 6937 in ISO 8859 v okviru enega računalniškega sistema je komplicirana in neučinkovita. Uporaba ISO 6937 bi morala biti omejena na telematske storitve, konverzija kode iz ISO 6937 v ISO 8859 bi bila potrebna le pri vhodu podatkov v računalnik. Taka strategija bi bila zlasti pomembna za sledeče kategorije uporabnikov in proizvajalcev:
a)za implementatorje in razvijalce izdelkov kot so: terminali, tiskalniki in programski paketi,
b) za nacionalne odbore in organizacije za standardizacijo za področje kodiranja, avtomatizacije pisarniške opreme in komunikacij ter
c)za vse ostale odbore ISO in CCITT, ki se ukvarjajo s standardizacijo medijev in komunikacij.

V primeru, da bo to sprejeto in da se zahteva izločitev ISO 6937 in pospešena uporaba ISO



					0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
					0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
					0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
					0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
					00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0	0	0	0	00	▨	▨	SP	0	@	P	`	p	▨	▨	NBSP	№	ю	п	Ю	П
0	0	0	1	01	▨	▨	!	1	A	Q	a	q	▨	▨	ђ	Ђ	а	я	А	Я
0	0	1	0	02	▨	▨	"	2	B	R	b	r	▨	▨	ѓ	Ѓ	б	р	Б	Р
0	0	1	1	03	▨	▨	#	3	C	S	c	s	▨	▨	ё	Ё	ц	с	Ц	С
0	1	0	0	04	▨	▨	$	4	D	T	d	t	▨	▨	є	Є	д	т	Д	Т
0	1	0	1	05	▨	▨	%	5	E	U	e	u	▨	▨	ѕ	Ѕ	е	у	Е	У
0	1	1	0	06	▨	▨	&	6	F	V	f	v	▨	▨	і	І	ф	ж	Ф	Ж
0	1	1	1	07	▨	▨	'	7	G	W	g	w	▨	▨	ї	Ї	г	в	Г	В
1	0	0	0	08	▨	▨	(	8	H	X	h	x	▨	▨	ј	Ј	х	ь	Х	Ь
1	0	0	1	09	▨	▨	)	9	I	Y	i	y	▨	▨	љ	Љ	и	ы	И	Ы
1	0	1	0	10	▨	▨	*	:	J	Z	j	z	▨	▨	њ	Њ	й	з	Й	З
1	0	1	1	11	▨	▨	+	;	K	[	k	{	▨	▨	ћ	Ћ	к	ш	К	Ш
1	1	0	0	12	▨	▨	,	<	L	\	l	|	▨	▨	ќ	Ќ	л	э	Л	Э
1	1	0	1	13	▨	▨	-	=	M	]	m	}	▨	▨	SHY	¤	м	щ	М	Щ
1	1	1	0	14	▨	▨	.	>	N	^	n	‾	▨	▨	ў	Ў	н	ч	Н	Ч
1	1	1	1	15	▨	▨	/	?	O	_	o	▨	▨	▨	џ	Џ	о	ъ	О	Ъ

Sl. 2. ISO 8859/8

8859, bi lahko nastali sledeči problemi:
-obstoječi terminali z implementirano ISO 6937 kodno tabelo bi postali neuporabni (na primer Xerox terminali).
-11 evropskih proizvajalcev je obljubilo, da bodo podpirali ISO 6937 in bi nova odločitev povzročila zmedo
-CEN/CENELEC, ki je sprejel obveznost, da podpre ISO 6937 bo postavljen v nerodno situacijo.

3. Stališča glede implementacije ISO 6937 in ISO 8859

3.1 Implementacija na terminalih in tiskalnikih

Če je v terminalu implementiran ISO 6937, potem ni problem implementirati tudi ISO 8859, ker je množica znakov enaka. Spremeniti je treba le del strojne opreme (firmware), ki namesto generiranja znaka kot kombinacije diakritičnega znaka in navadnega znaka generira le en znak.

V primeru, ko imamo naprave z visoko resolucijo, kjer za vsak znak potrebujemo veliko število bitov, se akcentirani znaki generirajo kot kombinacija zgornega in spodnjega akcenta ter same črke. Pri napravah nizke resolucije se akcentirani znak generira naenkrat brez kombinacij.

Pri tiskalnikih z marjeticami, kjer je število znakov omejeno, ima prednost ISO 8859. Tiskalniki z dvema marjeticama imajo možnost uporabe vseh črk ISO 8859, ena marjetica (190 znakov) bi pokrivala ASCII in druga obe desni strani kodnih tabel dveh delov ISO 8859. Pri tiskalnikih, ki uporabljajo verige, če je veriga relativno velika (na primer od 350 znakov) ni problem implementirati obeh standardov, seveda s čim manjšim številom dvojnih črk. Pri teh tiskalnikih ni korespondence med kodo in prikazovanjem določenega znaka, medtem ko je v sami napravi poskrbljeno za ustrezno korespondenco. Pri teh napravah bi se pojavil le problem počasnosti zaradi velikega števila znakov v verigi.

Gledano s stališča določenega geo območja je bolj varčno in gospodarno implementirati ISO 8859. Če želimo implementirati vse kodne tabele ISO 8859, potem ta prednost glede varčnosti odpade. Enako velja, če hkrati implementiramo latinično in cirilično pisavo, ker v tem primeru spet potrebujemo dve kodni tabeli in uporabo tehnike razširitve v 8-bitnem okolju. Če uporabljamo tehnologije z izmenljivimi tiskalnimi elementi, potem ima prednost ISO 8859. Vsak tiskalni element ustreza določeni kodni tabeli. To pomeni, da obstoječe tiskalnike z marjetico še naprej uporabljamo in po potrebi izmenjujemo marjetico v skladu s kodno tabelo. Kot zaključek temu lahko povemo še, da je cena tiskalnikov in terminalov sorazmerno nepomembna glede na ceno programske opreme.

3.2 Implementacija ISO 6937 in ISO 8859 v programski opremi

Prednost kode, v kateri je vsak znak določen z enim zlogom, je neprimerljiva s kodo z nekonstantno dolžino. Velika večina obstoječe programske opreme zelo lahko obdeluje 8-bitno kodo iz ISO 8859. Problemi se pojavljajo v primerih, ko želimo, da naša programska oprema sprejema dve kodni tabeli, na primer ISO 8859/2 in ISO 8859/3. V teh primerih uporabljamo ali:
-poseben krmilni znak ("single shift" funkcijo), ki nam kaže, da so prihajajoči znaki predstavljeni z dvema zlogoma, prvi zlog je koda za SS1 in drugi za sam znak,
-poseben znak ("locking shift" funkcijo), s katerim spremenimo celo desno stran kodne tabele.
Nekateri od zagovornikov ISO 6937 trdijo, da je prednost te kodne tabele v možnosti njene uporabe kot 7 in 8-bitno, v odvisnosti od sprejemne naprave. Pri terminalih se da problem sprejemanja znakov z različno dolgimi zlogi dokaj lahko rešiti, pri programski opremi pa se zatakne, ker vemo, da je težko pisati programe, ki sprejemajo hkrati 7 in 8-bitne zloge. Pri 7-bitnih zlogih večina zahodno evropskih pisav uporablja SS1 in LS1 funkcije za potrebe generiranja akcentiranih znakov. Pri tem ISO 6937 kaže na določene prednosti, ker ni pojava redundančnega kodiranja (v kodni tabeli ni enakih znakov). To velja le, če se uporablja samo ISO 6937/1/2. Pri uporabi drugih kodnih tabel iz ISO 6937, ki so v pripravi (na primer grške črke iz grške abecede in iz kodne tabele za tehnično uporabo) se ta prednost zgubi.

4.Aplikacije in uporaba ISO 6937 in ISO 8859

Problemi, ki lahko nastanejo pri uporabi ISO 6937 pri obdelavi podatkov, so mogoče najbolj vidni pri sortiranju. Kako lahko primerjamo dva znaka, od katerih je eden brez akcenta in je zapisan z 8-biti in drug akcentiran znak, ki je zapisan s 16 biti Če v programski opremi za sortiranje to ni posebej specificirano, lahko nastanejo težave. Pogosta tehnika pri sortiranju je transformacija vsakega znaka v numerično vrednost in uporaba te vrednosti za primerjavo.

Večina CPU-jev ima (tako kot IBM 370), vgrajen ukaz prav za to operacijo. Kaj se bo zgodilo, če namesto na 8-bitni zlog naleti ukaz na 16 bitov Tudi ostali programi za konverzijo med ASCII in EBCIDIC kodo so narejeni na podoben način - translacija le 8-bitnih zlogov.

Drug problem, ki lahko nastane pri uporabi ISO 6937, je urejanje teksta. Večina zaslonsko orientiranih urejevalnikov je narejenih na podlagi predpostavke, da mesto, ki ga zavzame črka na zaslonu, odgovarja enemu znaku, zakodiranemu z 8-biti. Tehnika, s katero pozicioniramo kursor, sloni na predpostavki, da je vsak znak zakodiran z enako dolgim zlogom. Pozicioniranje kursorja ter ažuriranje zaslona v primeru implementacije ISO 6937 bo zahtevalo spremembe v opremi zaradi prilagajanja na spremenljivo dolžino kode. Podobni problemi lahko nastajajo pri specifikacijah dolžin polj. Koliko znakov bomo spravili v polju z določeno dolžino, bo odvisno od tega, kakšni znaki bodo nastopali. Na primer: ukaz "enter last name, up to 12 character" nam ne zagotavlja, da bomo dobili pravilen zapis v podatkovni bazi, ker je to, koliko znakov bomo pobrali, odvisno od tega, kateri so ti znaki.

5. Sklepne misli

Iz vsega povedanega lahko izločimo sledeče:

ISO 6937 je aplikacijsko orientiran standard namenjen za implementacijo v storitvah Videotex in Teletex ter za naprave, ki to aplikacijo omogočajo. Po definiciji ISO-ja in CCITT-ja je tekst informacija, namenjena ljudem, ki to informacijo sprejemejo, jo obdelajo in jo naprej posredujejo, zato sta 8-bitni ASCII in ISO 8859 standarda namenjena predvsem za prenos in obdelavo teksta, ISO 6937 pa le komunikacijam s tekstovnimi podatki. To pomeni, da je treba ISO 6937 uporabljati le pri čistem prenosu in v primerih, ko je tekst shranjen na magnetnih medijih, le za prikazovanje. Prvotna ideja SC2 WG4 pri razvoju ISO 6937 je bila izdelati splošno uporaben standard za prenos in izmenjavo podatkov. Opozorila strokovne javnosti, da je ta standard neprimeren za splošno uporabo zaradi problemov pri obdelavi, so povzročila, da je SC2 WG4 dobil nalogo, da še enkrat obdela ISO 6937 in omeji natančneje njegovo uporabo.

Eden od glavnih motivov za razvoj ISO 8859 je bilo zavračanje nekaterih proizvajalcev, da implementirajo ISO 6937. Kljub temu, da bo širok sprejem ISOa 8859 obsodil terminale z implementiranim ISO 6937, večina proizvajalcev trdi, da obstoječe terminalne naprave v največji možni meri podpirajo 8-bitne kode s podobnimi lastnostmi kot ISO 8859. Pričakovati je, da bo število še večje po sprejemu ISO 8859/3-8 v okviru ISO SC2 in ANSI BSR X3.134.2. Kaže, da je večina ameriških proizvajalcev pripravljena podpreti ASCII terminale, v katerih bo vključena tudi najnovejša ASCII 8-bitna koda. V primeru, da bo ISO 6937 še naprej podpiran od nekaterih proizvajalcev kot prevladajoč standard za slepi prenos, ni treba pričakovati, da bodo ameriški proizvajalci in njihovi uporabniki prilagajali svoje aplikacije temu standardu.

Dosedanja razprava kaže, da bo najverjetneje v kratkem (na plenarnem zasedanju ISO SC2 marec 1987) sprejeto priporočilo ANSI X3 in ISO SC2, s katerim bo ISO 8859 postal prevladujoč standard za prenos in obdelavo podatkov, vključno z uporabo pri izmenjavi tekstovnih podatkov v okolju avtomatiziranih pisarn. Ugotovitev, da mora uporaba računalnikov sloneti na enakih standardih ne glede na to, kje se računalnik nahaja, ali v pisarni ali nekje drugod, je bila eden od pomembnejših dejavnikov, ki so pripeljali do združitve med ISO TC95 (office systems) in ISO TC97 (information processing). V tem kontekstu je uporaba različnega nabora znakov v avtomatizirani pisarni in v poslovno proizvodnem sistemu podjetja skrajno nezaželena.

Obstaja verjetnost, da ne bo mogoče zaradi obstoječih telematskih mrež popolnoma izpodriniti ISO 6937. ISO 6937 je koda, ki je narejena za specifično uporabo (Videotex in Teletex) in taka mora tudi ostati. Terminali in tiskalniki z določenimi dodatki lahko interpretirajo oba standarda, zato je treba pričakovati, da bo njihova uporaba in prodaja ostala neprizadeta. Te naprave lahko uporabljamo pri telematskih storitvah, lahko pa kot periferijo računalnikov s splošno uporabo. Konverzija med ISO 6937 in ISO 8859 bo relativno enostavna, ker vsebuje ISO 6937 podmnožico znakov ISO 8859. Translacijo podatkov, ki so namenjeni nadaljni obdelavi iz formatiranih podatkov lahko zmeraj opravimo s programom za preformatiranje strani.

Iz vsega navedenega sledi, da je ISO 8859 boljša tehnična rešitev in da je bolj splošno uporabna. Tako je pričakovati, da bo ISO najverjetneje priporočil kot splošno uporaben standard ISO 8859. Priporočati le eno možno rešitev pomeni, da bodo nekateri pri tem prizadeti in da bodo drugi pridobili na račun prvih. Naša naloga je, da podpremo boljšo tehnično rešitev v interesu boljše standardizacije in reševanja naših potreb (uporaba tudi cirilice).

6.Reference

1.Delovni dokumenti ISO TC97/SC2, WG4 in WG2
2.Delovni dokumenti ANSI X3L2

NOVE KNJIGE

Terry Winograd and Fernando Flores: Understanding Computers and Cognition: A New Foundation for Design. Ablex Publishing Corporation. Norwood, New Jersey (1986).

Terry Winograd je danes prva avtoriteta umetne inteligence (Center for the Study of Language and Information, Stanford University), center v stanfordskem univerzitetnem taboru pa je svetovno središče umetne inteligence. Fernando Flores (podjetje Action Technologies) je bivši minister za gospodarstvo in finance v vladi Salvadorja Allendeja (1970-73). Knjiga "Razumevanje računalnikov in spoznavanja: novi temelji oblikovanja" je posvečena čilenskemu ljudstvu. Knjiga je posebnost, ki ji ne najdemo primerjave v svetovni znanstveni, tehnološki, informatični, umetnointeligenčni in računalniški literaturi. Knjiga je svojevrstno odkritje za tehnika, ki mu je trdna znanost in tehnološka stvarnost življensko vodilo, ki pri svojem delu in tehnološkem oblikovanju ne vidi in ne najde smisla v filozofski raznovrstnosti. Vendar je knjiga tudi novost za filozofa, ki iz nje lahko razpoznava ontologičnost (bolj domače metafizičnost in s tem relativnost vsakršnje filozofske resničnosti) polpretekle in današnje filozofije. Knjiga obravnava dve, navidezno razhajajoči se usmeritvi: računalniško tehnologijo in naravo človekovega obstoja. Tu pokaže, kako so razhajanja v bistvu povezana in soodvisna.

To, kar je za tehnika presenetljivo, je prav gotovo filozofsko ozadje knjige. Domala heretično za tehnika zveni dovčerajšnje pojmovanje hermenevtike in njeno vpeljevanje v tehnološko problematiko oblikovanja. In kaj naj imata tehnika in tehnologija skupnega z razumevanjem biti, ki je na Slovenskem le nekakšna zakasnela Filozofska bit? Vendar se bralec kmalu sprijazni z umestnostjo razumevanja ontologije, z ilustracijo vrženosti (v svet), s tubitjo, z evolucijo, jezikom, avtopoiesis, racionalistično usmeritvijo itd. Knjiga je razdeljena v tri poglavja: teoretično ozadje; računanje, misel in jezik; oblikovanje.

Kakšna je prav za prav pot knjige, ki je pred nami? Namen knjige je, da razvije novo usmeritev z nastajanjem dvogovora med njo in bralcem. Nekatera filozofska dela iz bližnje preteklosti so dovolj izzivalna v teoretičnem predpostavljanju t. i. racionalistične tradicije, ko se postavljajo vprašanja o stvareh, ki se nam dozdevajo podarjena z našo tradicijo. Ta tradicija je pripeljala na površino pojavnost, ki je značilna tudi za današnjo računalniško tehnologijo. Ta pojavnost išče nove alternativne usmeritve pri oblikovanju računalniških orodij.

Prvi del knjige (poglavja 1-6) opisuje racionalistično tradicijo in prikazuje tri ločene predmetnosti tega dela, od katerih je vsaka v nasprotju s tradicijo, čeprav te predmetnosti bistveno vplivajo na naše razumevanje. Pri tem postane jasno, kako najpomembnejše iluzije racionalistične tradicije temeljijo na prepričanju, da je znanje sestavljeno iz eksplicitnih teorij.

Drugo poglavje opisuje podrobneje racionalistično tradicijo in pokaže, kako ta tradicija utemeljuje razumevanje kulturne zdrave pameti jezika, mišljenja in racionalnosti. Cilj tega poglavja je v razkrivanju nagnjenosti (predsodkov) in podmen, ki se skrivajo, zamolčujejo in prikrivajo za svojimi pojavnostmi v ozadju našega jezika.

Tretje poglavje se ukvarja s tradicijo, ki vsebuje hermenevtiko (raziskovanje predstavljanja) in fenomenologijo (filozofsko preučevanje temeljev izkustva in delovanja). Ta tradicija se je razvila iz humanističnih ved in je povezana z razmerjem individualnega in kontekstnega, še posebej socialno kontekstnega, v katerem posameznik živi. Tu se poudarjajo tista področja človekovega izkustva, v katerih imata individualna predstavitev in intuitivno razumevanje (v nasprotju z logičnim izpeljevanjem in zavestno refleksijo) osrednji pomen. Tu se postavlja izziv prepričanju, da je formalno analitično razumevanje teh pojavov sploh mogoče.

Osnova tega razmišljanja so dela Hansa Georga Gadamerja in Martina Heideggra. Tudi vrsta drugih filozofov je raziskovala sorodne ideje, vključno s fenomenologisti kot so Husserl, Ricoeur in Merleau-Ponty, z eksistencialisti kot je Sartre, s pragmatisti kot sta Mead in Dewey, z današnjimi političnimi filozofi kot sta Habermas in Apel in kot so celo nekateri filozofi z bolj analitičnim ozadjem, npr. Wittgenstein. Heidegger in Gadamer sta bila izbrana zaradi vpliva njunih del na današnje naše učenje in zaradi pomena njunih del v današnji tradiciji. Heidegger tu prav gotovo izstopa kot sodoben filozof, saj je opravil najtemeljitejšo, najprodornejšo in najradikalnejšo analizo vsakdanjega izkustva. Njegove ideje ostajajo marsikje na poti drugih filozofov in so vkoreninjene v današnje usmeritve. Gadamer je bil najbolj sistematičen pri uporabi te usmerjenosti na problem jezika, ki ostaja prej ko slej osrednji problem današnjosti.

Cetrto poglavje prikazuje delo čilenskega nevrobiologa Humberta R. Maturane, ki je postal široko znan s svojim delom o nevrofiziologiji vida. Njegova osnovna izobrazba je biološka in ne filozofska in danes preučuje Maturana naravo bioloških organizmov z vidika mehanističnih strukturnodeterminiranih sistemov. Njegova dela so bistveno prispevala k razumevanju spoznavanja in k perspektivi racionalistične tradicije. Od njega prihaja tudi ideja, da interakcije med živimi bitji niso komunikacijske. Drevesi, ki rasteta eno ob drugem, ne razpravljata o delitvi razpoložljivega prostora, vendar eno drugega vznemirjata tako, da rasteta nesimetrično. Maturana poudarja, da to, kar ljudje izmenjujejo medseboj, ni komunikacija, temveč perturbacija (zmeda, zmešnjava, vznemirjenje, motnja, nemir). Nasmeh ni isto kot izjava "Srečen sem".

Pisci kot Heidegger so izpostavili dominantni vidik uma, ko so izjavili, da spoznavanje ne temelji na sistematični manipulaciji predstavitev. Ta perspektiva je bila uporabljena kot osnova za večkratno kritiko umetne inteligence (H. L. Dreyfus: What Computers Can't Do; J. Haugeland: The Nature and Plausibility of Cognitivism), ki je bila na svojem začetku mistično področje za druge znanosti. Pri vpraševanju o splošnem razumevanju razmerij med percepcijo, predstavitvijo in mislijo, to sklicevanje na prvi pogled zanikuje fizične temelje človekovega delovanja. Maturana ponuja dvoje uporabnih uvidnosti (razumevanj), s katerima je omogočena izognitev tako omejenemu pred-razumevanju: vloga opazovalca pri oblikovanju pojavnostnih domen in zamisel strukturnega povezovanja, ki dopušča razumevanje obnašanja, ko je to sicer mehanično generirano, vendar ni programirano. Kot biolog poudarja Maturana konceptualni okvir, v katerem nastajajo pojavi predstavljanja kot nujna posledica zgradbe bioloških bitij. Hkrati nas poziva, da sprejmemo novo razumevanje o načinu, s katerim govorimo o fizični naravi in da to razumevanje uporabimo na nas same.

Vprašanja o spoznavanju so prepletena z vprašanji o naravi jezika. Peto poglavje pokaže najprej, kako je uvidnost v hermenevtiko pomembna za tradicijo, ki se je razvila z lingvistiko in analitično filozofijo jezika. Tu je poudarjena vloga poslušalca pri aktivnem generiranju pomena in pokazano je, zakaj je idealizacija "dobesednega pomena" nemogoča na ravni navadnega jezika. Nadalje se v tem poglavju uvaja teorija govornega akta, kot sta jo razvila J. L. Austin (How to Do Things with Words, Cambridge, MA: Harvard University Press, 1962) in J. R. Searle (Speech Acts, Cambridge: Cambridge University Press, 1969 in še vrsta drugih del tega avtorja) in je bila kasneje sprejeta pri socialnih filozofih, kot je Juergen Habermas. Čeprav se je ta teorija razvila iz analitične filozofske smeri, je njen vidik jezika kot govornega akta izzval racionalistično tradicijo s predpostavko, da je jezik in z njim tudi misel konec koncev osnovana na socialni interakciji. Teorija govornega akta je začetek določenega razumevanja jezika kot dejanja socialne kreacije. V zadnjem delu poglavja je prikazana nova sinteza teorije govornega akta in hermenevtičnega razumevanja jezika, razvitega v tretjem poglavju. Ta sinteza predstavlja osrednjo zamisel predstavitve računalniške tehnologije v drugem delu knjige in nas pripelje do sklepa, da oblikujemo svet skozi jezik. To pa lahko še kako bistveno vpliva na oblikovanje (tehnološko načrtovanje).

Šesto poglavje je zamišljeno kot prehod na nadaljevanje knjige s povzetkom bistvenih točk iz prejšnjih poglavij. Tu se dokončno srečujemo z zavračanjem spoznavanja kot oblike manipulacije znanja objektivnega sveta, s primarnostjo akcije in njene osrednje vloge v jeziku in z nezmožnostjo popolnega zajetja ozadja, ki vpliva na kritiko današnje računalniške tehnologije.

Drugi del knjige (poglavja 7-10) postavlja konkretna vprašanja o delovanju računalnikov. Tu je cilj razumevanje in prevrednotenje sedanjega razvoja in predvidevanje prihodnjega.

Sedmo poglavje opisuje, kaj se dogaja s človekom, ki programira. Poudarek je na razmerju med namerami programerjev in obnašanjem strojev, ki izvajajo oblikovane programe. Programiranje je postopek oblikovanja simboličnih predstavitev, ki bodo interpretirane na neki ravni v okviru konstruktne hierarhije z različnimi stopnjami abstrakcije. Interakcije med predstavitvenimi ravnmi so lahko zapletene, ker je vsaka raven odvisna od nižjih ravni. Ta opis oblikuje temelje za razpravo o računalniški inteligenci v naslednjih poglavjih.

Osmo poglavje raziskuje računalne metode, ki so bile predložene kot osnova umetne inteligence. Tu se izhaja iz nasprotja določenega prepričanja: danes ni mogoče konstruirati strojev, ki bi izkazovali ali uspešno modelirali inteligentno obnašanje. Tu se vprašuje, zakaj se pripisujejo umske lastnosti le računalnikom in ne tudi drugim strojem, ki procesirajo informacijo. Temeljiteje se opisujeta delo v umetni inteligenci in analiza omejitev. Večina težav današnjih raziskav izvira iz tistih temeljnih usmeritev, ki izenačujejo inteligenco z racionalističnim problemom reševanja z uporabo hevrističnih procedur.

Deveto poglavje načenja tematiko računalniških programov za obdelavo naravnih jezikov. Tu je pokazano, zakaj se vrsta programov, ki so bili razviti v sedemdesetih letih, ni približala sposobnosti interpretacije pomena. Kljub iznajdljivim analiznim in razpoznavnim metodam, je ostalo obzorje razumevanja bistveno omejeno. Čeprav obstajajo praktične aplikacije računalniškega procesiranja formalizmov naravnih jezikov, bodo računalniki prej ko slej ostali nesposobni za uporabo jezika na način, ki je značilen za človeško bitje; ta omejitev bo veljala tako za interpretiranje kot za generiranje informacije, ki je jezikovno bistvena. J. Fodor (Methodological Solipsism Considered as a Research Strategy in Cognitive Psychology) je pokazal, kako so jasno opredeljeni modeli umetne inteligence na splošno ekvivalentni starejši filozofski analitični tradiciji, ki črpa svoje ideje iz obdobja Aristotla in pred njim.

Deseto poglavje kritično osvetljuje nekatere glavne raziskovalne usmeritve na področjih, kot so tehnika znanja, izvedeniški sistemi in t. i. računalniki pete generacije. Tu je opisan premik iz splošnega cilja oblikovanja programov, ki lahko razumejo jezik in misel v smer oblikovanja programske opreme za posebne nalogovne domene; opisana in ocenjena je projekcija razvoja v naslednjih letih.

Tretji del (poglavji 11 in 12) prikazuje alternativno oblikovalno usmeritev, ki temelji na teoretičnem ozadju, razvitem v knjigi. Tu ni bistveno vprašanje primerjave med računalnikom in človekom, temveč vprašanje odprtja računalniškega potenciala, smiselnega za človekovo življenje in delo. Ko se enkrat odmaknemo od slepote, generirane s starimi vprašanji, se nam odpre širša perspektiva o tem, kaj računalniki zmorejo.

Enajsto poglavje se ukvarja z nalogo oblikovanja računalniških orodij v organizacijskem kontekstu. Tu je v središču pozornosti dejavnost "upravljavcev", ki nastaja tudi v socialni interakciji in pri naporih sodelovanja. S heideggrsko razpravo o "vrženosti" in "porušitvi" prihajamo do sklepa, da modeli racionalističnega reševanja problemov ne odražajo tega, kako so akcije dejansko determinirane in da programi, ki temeljijo na takih modelih, ne morejo biti uspešni. Neglede na to, pa ostaja vloga računalniške tehnologije kot podpora upravljavcem pri soočanju z zapletenimi konverzacijskimi strukturami v okviru organizacije ohranjena. Večina dela, ki ga opravljajo upravljalci, je povezana z začenjanjem, spremljanjem in koordiniranjem mreže govornih aktov, ki sestavljajo socialno akcijo.

Dvanajsto poglavje se vrača k osnovnim vprašanjem oblikovanja in pregleduje možnosti računalniške tehnologije, ki se odpirajo z razumevanjem snovi, razvite v prejšnjih poglavjih. Po kratkem pregledu pomembnih teoretičnih izhodišč se preučujejo oblikovalni pojavi s prikazom novega pristopa v dejanskem primeru.

Ta knjiga je v nekem smislu delo o računalnikih, čeprav posega v možnosti, kaj računalniki zmorejo. In tu odpira problematiko o razvoju računalnikov v naslednjih desetletjih. Prepričanje avtorjev knjige je, da računalniki ne morejo razumevati jezika in da po sedanjih poteh umetne inteligence ne bo mogoče doseči inteligentnega obnašanja strojev. Raziskovalci UI često poudarjajo, da so njihovi programi pomanjkljivi in verjamejo, da bo te pomanjkljivosti sčasoma mogoče odpraviti. Avtorja poudarjata, da ta pot UI ni pot tistega izboljševanja, ki bi vodila do inteligence. Večina knjige je prepredena z metodičnimi napadi na mišljenje ljudi, ki delajo z računalniki.

Anton P. Zeleznikar

PISMA BRALCEV

Laboratorij za programsko opremo
Univerza Edvarda Kardelja v Ljubljani
Fakulteta za elektrotehniko
B.Vilfan, V.Mahnič, T.Mohorič

Ljubljana, 9.10.1986

Uredništvu časopisa INFORMATICA

V vašem časopisu (INFORMATICA ŠT.4, 1986, str. 97) je bil objavljen dopis tov. Ranka Smokvine, ki se nanaša na članek B.Vilfan, V.Mahnič, T. Mohorič: Ocena programskih orodij IDA, INFORMATICA št. 3, 1986. Kot avtorji omenjenega članka in sklicujoč se na Zakon o javnem obveščanju (Uradni list SRS št. 2/86) prosimo, da objavite naslednji odgovor.

Strinjamo se s tov. Smokvino, da "će sve sadašnje i buduće korisnike IDA-alata zanimati praktičan rad sa IDA-alatima (i usporedbe sa ULTROM) kao i rezultati mjerenja performansi". Kljub temu smo se odločili, da v omenjenem članku ne obdelamo praktičnih meritev (in to odločitev tudi jasno izrekli v uvodu članka) iz naslednjih razlogov:

1. Preden se lotimo kakršnihkoli praktičnih primerjav med različnimi (a sorodnimi) programskimi orodji je potrebno skrbno proučiti tisto, kar proizvajalci zagotavljajo glede njih oziroma njihove deklarirane karakteristike. Očitno je, da so ti podatki dostopni le preko priročnikov.

2. Kvaliteta nekega sistema za upravljanje podatkovnih baz ima izredno mnogo aspektov in po naše imajo nekateri aspekti, ki niso povezani z rezultati meritev, pogosto večjo težo (n. pr. podatkovni modeli in fleksibilnost modeliranja stvarnosti, podatkovna neodvisnost, sočasnost dostopa do podatkov, zaščitni mehanizmi, pripomočki za razvoj aplikacij ipd.). Torej smo želeli na prvem mestu izpostaviti takšne aspekte IDA-orodij.

3. Praktične meritve delovanja sistemov za upravljanje podatkovnih baz predstavljajo določen problem s stališča izbire karakteristik testnega nabora podatkov. V tem primeru nimamo opravka s tako preprosto situacijo kot pri merjenju delovanja računalniškega sistema, kjer zadostuje izbrati en testni primer, pri katerem prevladuje vhod/izhod, enega, pri katerem prevladujejo računske operacije (morda izbrane vrste, n.pr. realna aritmetika) in enega, ki leži nekje vmes. Na misel pride vsaj nekaj karakteristik, ki močno vplivajo na izmerjene rezultate:

   -uporabljen podatkovni model v zunanji shemi (n.pr. Logical User View pri Ultri)
   -število sočasnih uporabnikov podatkovne baze
   -obseg aktiviranja zaščitnih in nadzornih mehanizmov
   -način iskanja podatkov (iskanje, ki je bilo vnaprej predvideno z logično organizacijo setov ali iskanje na vnaprej nepredviden način)
   -fizična razmestitev podatkov in obseg redundanc
   -gosteči računalniški sistem (aparaturna oprema in operacijski sistem)
   -uporabljen gostеči programski jezik
   itd.

   Zato smo menili, da merjenje delovanja podatkovne baze pravzaprav predstavlja dokaj obsežen samostojen problem, ki mora tudi biti deležen samostojne obravnave.

4. V svetu obstaja vrsta organizacij, ki analizirajo podbne programske produkte in v svojih publikacijah objavljajo informacije o njih, namenjeje uporabnikom (n.pr. Datapro, Auerbach, Data Decisions itd.). V teh analizah praviloma ni ocen performans, pa vendar jih uporabniki smatrajo za koristne.

Upamo, da podana pojasnila zadostno utemeljujejo našo izbiro snovi pri članku.

B. Vilfan, V. Mahnič, T. Mohorič, l. r.

VELEBIT
OOUR INFORMATICA o.sol.o.
ZAGREB

Zagreb, 29.9.1986

Uredništvu časopisa INFORMATICA

Molim redakciju da objavi u prvom slijedećem broju Informatica-e priloženo pismo kao odgovor na članak "OCENA PROGRAMSKIH ORODIJ IDA", autora B.Vilfana, V.Mahniča i T.Mohorica u Informatica 3/86.

VELEBIT OOUR INFORMATIKA je zastupnik firme CINCOM SYSTEMS, spomenute u članku, od 1975. godine, što je znano mnogima, a naročito ISKRA-DELTA-i.

Ja Luka Omejec, sam od prvog dana zastupanja jedan od aktivnih sudionika na planu marketinga i tehničke podrške Cincom Systems proizvoda u Jugoslaviji, kao i odgovorna osoba za dogadjaje u Jugoslaviji prema Cincom-u.

Iz navedenih razloga, vjerujem da ćete naći opravdanje za objavljivanje priloga.

Drugarski pozdrav - sretno.
Hvala.

mgr. Luka Omejec, dipl. ing., savjetnik

Prilog

U Informatica 3/86 objavljen je članak "Ocena programskih orodij IDA", autora B.Vilfan, V.Mahnič, T.Mohorič.

Budući da je u članku spomenuta ULTRA firme CINCOM SYSTEMS INC., koje je Velebit OOUR Informatika - Zagreb zastupnik od 1975 god., smatramo za potrebno da kažemo sljedeće:

Test je naručen od ISKRA DELTA koja je bila u povoljnijoj poziciji tumačenja svojeg proizvoda. Iskra Delta nema službeno priručnike Ultre.

Na zahtijev Iskra Delta, izvršen je benchmark test IDA-ULTRA na aplikaciji osobnih dohodaka, u jednom ERC-u u Jugoslaviji.

Ultra je potrošila jednako vreme kao i IDA (oboje su radile u TOTAL like modu).

Autori, po mom saznanju, nisu tražili niti od Velebita niti od Cincom Systems priručnike kao ni tumačenje odredjenih stvari.

Zbog svega toga, vjerujem, da su odredjene tvrdnje u članku posljedica niza nesporazuma koje bih pokusao navesti.

Nesporazum broj jedan:

IDA- Baza je DBMS-software tzv. III. GENERACIJE - CODASYL tip

ULTRA- je relacijski DBMS IV. GENERACIJE.

Iz tog razloga trebalo bi prvo i isključivo znanstveno i objektivno utvrditi, ako je to uopće moguće, prednosti Cullinane-Codasyl modela u odnosu na Codd-Relacijski model.

Nesporazum broj dva:

IDA-BAZA je u klasi "one scheme architecture"

ULTRA je u klasi "three scheme architecture".

Iz tog razloga trebalo bi prvo i isključivo znanstveno i objektivno oboriti, ako je to uopće moguće, preporuke ANSI SPARC iz 1977.

Nesporazum broj tri:

IDA-COGEN, IDA-EKRAN, IDA-LEKSIKON su pokušaji oživljavanja COBOL-A, programskog jezika III. Generacije.

Nesporazum broj četiri:

"...glede na to, da relacijski sistemi upravljanja podatkovnih baz, ki so danes na voljo po svojih performansah še vedno zaostajajo za mrežnim,..."

Već je rečeno, da je radjen test u Jugoslaviji po narudžbi ISKRA-DELTA, te da je ULTRA bila jednako brza kao i IDA-BAZA radeći "TOTAL-LIKE".

ULTRA ne "izhaja" iz TOTAL-a (kao IDA), nego je to potpuno nova relacijska -three scheme-arhitektura.

Postoji još čitav drugi niz nesporazuma, koji su kao i prethodni koncepcijsko-teoretijski zasnovani, te u pragmatizmu zadataka autora logično prisutni.

Ovih nekoliko napomena, smatram, trebalo bi biti poticaj teorijskom fundiranju kriterija sumjerljivosti sumjerljivog, a ne "baba i žaba".

Spremni, smo u tom smislu, sudjelovati u svim slijedećim aktivnostima koje bi inicirala ISKRA-DELTA, UNIVERZA LJUBLJANA, redakcija INFORMATICA-e, ili bilo koja druga institucija.

Naša adresa: RO Velebit, OOUR Informatika
Trg J. F. Kennedy 6A
41000 Zagreb

```
============================================
=                                          =
=     Avtorsko abecedno kazalo časopisa    =
=     Informatica, letnik 10 (1986)        =
=                                          =
============================================
```

Clanki

Bogunović N.: Analitički postupci u odredivanju neproduktivne obradezajednu vrstu procesnih računarskih sastava. Informatica 10 (1986), št. 1, str. 62-68.

Brajak P.: Parallel Processing: Architecture of the 1990s. Informatica 10 (1986), št.3, str. 3-10.

Djordjević S.J.: Organizacije ključeva u organizacijama podataka. Informatica 10 (1986), št.3, str. 41-45.

Dujmanović J.J., M.Levanić: PLOT5- Jedan jednostavni programski sistem za grafičke terminale i koordinatne crtače. Informatica 10 (1986), št. 4, str. 68-73.

Furht B., V.Milutinović: A Survey of Microprocessor Architectures for Memory Management. Informatica 10 (1986), št. 2, str. 18-36.

Gams M., T. Zrimec: MICROPROLOG. Informatica 10 (1986), št. 1, str. 80-85.

Gerkeš M.: Logični modeli računalniških struktur II. Informatica 10 (1986), št. 1, str. 7-22.

Jamšek J.: Operacijski sistem VM/SP. Informatica 10 (1986), št. 1, str. 32-42.

Jenko M., Vodopivec A.: Možnosti, ki jih ponuja mikroelektronika sistemskim načrtovalcem. Informatica 10 (1986), št. 4, str. 33-43.

Jerman-Blažič B., I.Fabič: Trinivojska arhitektura informacijskih sistemov po modelu ISO TC97/DP 9007. Informatica 10 (1986), št. 3, str. 26-33.

Kemeci Z., T. Turčić: Jedna mogućnost proširenja skupa upravljačkih naredbi RATFOR preprocesora za FORTRAN. Informatica 10 (1986), št. 1, str. 3-6.

Kolbezen P., B. Mihovilović: Mikroprogramiranje s pretokom podatkov. Informatica 10 (1986), št. 3, str. 61-63.

Kononenko I., I. Bratko, E. Roškar: Sistem za induktivno učenje Asistent. Informatica 10 (1986), št. 1, str. 43-52.

Lakner B., F. Dacar: Reailizacija z mikroračunalnikom. Informatica 10 (1986), št. 1, str. 23-27.

Maleković M.: Primjena mehaničkog dokazivanja teorema u rješavanju implikacionog problema za t-zavisnosti u relacionim bazama podataka. Informatica 10 (1986), št. 2, str. 60-67.

Maleković M.: Implikacioni problem za višeznačne zavisnosti i mehaničko dokazivanje teorema. Informatica 10 (1986), št. 3, str. 46-51.

Maleković M.: Podskup-zavisnosti u relacionim bazama i mehaničko dokazivanje teorema. Informatica 10 (1986), št. 3, str. 57-60.

Mavrič S., B. Mihovilović, P. Kolbezen: Povezovalne mreže večprocesorskih sistemov. Informatica 10 (1986), št. 4, str. 44-51.

Mihovilović B., S. Mavrič, P. Kolbezen: Transputer - osnovni gradnik večprocesorskih sistemov. Informatica 10 (1986), št. 4, str. 81-84.

Novak A.: Ladijski informacijski sistem in računalniško tovorjenje ladij. Informatica 10 (1986), št. 4, str. 51-59.

Ozimek I., V. Avbelj: Videoterminal PMT-100. Informatica 10 (1986), št. 3, str. 22-25.

Pavešić N., S.Ribarić: Paralelni sustav za raspoznavanje znakova. Informatica 10 (1986), št. 4, str. 60-67.

Petković D.: Optimal Code Generation for Some Special Classes of Machines. Informatica 10 (1986). št. 2, str. 45-47.

Prešern S., R. Murn, D. Peček: Paralelno procesiranje slik. Informatica 10 (1986), št. 3, str. 18-21.

Radovan M.: Model deeduktivne baze podataka implementiran u PROLOGu. Informatica 10 (1986), št. 1, str. 69-75.

Radovan M.: Logika i procesiranje znanja. Informatica 10 (1986), št. 4, str. 3-17.

Robič B., J. Silc: On Choosing a Plan for the Execution of Data Flow Program Graph. Informatica 10 (1986), št. 3, str. 11-17.

Robič B., J. Silc: Razvrstitev novogeneracijskih računalniških arhitektur. Informatica 10 (1986), št. 4, str. 18-32.

Ružić A., A. Klofutar: Pregled paralelnega programiranja. Informatica 10 (1986), št. 2, str. 48-59.

Silc J., B. Robič: Procesor s podatkovno pretokovno arhitekturo. Informatica 10 (1986), št. 4, str. 74-80.

Tvrdy I.: Okolje novih telematskih storitev in ISDN. Informatica 10 (1986), št. 4, str. 85-89.

Vilfan B., V. Mahnič, T. Mohorič: Ocena programskih orodij IDA. Informatica 10 (1986), št. 3, str. 52-56.

Završnik B.: Sistemsko vodilo Multibus II. Informatica 10 (1986), št. 3, str. 34-40.

Žabjek-Dolinšek J.: Operativno načrtovanje gibanja vlakov. Informatica 10 (1986), št. 2, str. 37-44.

Železnikar A. P.: Overlapping: A Paradigm of Parallel and Sequential Processing. Informatica 10 (1986), št. 2, str. 3-17.

Nove računalniške generacije

Jurak L.: O japonskem projektu računalnikov nove generacije. Informatica 10 (1986), št. 3, str. 64-65.

Jurak L.: Serijski stroj za sklepanje. Informatica 10 (1986), št. 3, str. 65-69.

Jurak L.: ESP- Razširjen serijski PROLOG (Extended serial PROLOG). Informatica 10 (1986), št. 3, str. 69-73.

Jurak L.: Sekvenčni PROLOG računalnik. Informatica 10 (1986), št. 3, str. 73-75.

Prešern S.: Advanced Microprocessors and High-Level Language Computer Architecture. Informatica 10 (1986), št. 4, str. 90-96.

Silc J., B. Robič: Podatkovno pretokovni procesor uPD7281. Informatica 10 (1986), št. 3, str. 76-77.

Železnikar A. P.: Od Sappora do Tokia nazaj v Ljubljano. Informatica 10 (1986), št. 2, str. 68-74.

Železnikar A.P.: Kako si Američani predstavljajo peto računalniško generacijo. Informatica 10 (1986), št. 2, str. 75.

Železnikar A.P.: Paralelno procesiranje: ali se res približuje. Informatica 10 (1986), št. 2, str. 75-76.

Železnikar A. P.: Paralelno procesiranje: resničnost ali fantazija. Informatica 10 (1986), št. 2, str. 77-81.

Novice in zanimivosti

Železnikar A. P.: ZDA in Japonska ukinjata carine. Informatica 10 (1986), št. 2, str. 81.

Železnikar A. P.: Siemens se hitro razvija. Informatica 10 (1986), št. 2, str. 81.

Železnikar A. P.: V izdelavi je superračunalnik z 61,4 gigaflops. Informatica 10 (1986), št. 2, str. 81.

Železnikar A. P.: Quo vadis Austro-Chip? Informatica 10 (1986), št. 2, str. 81.

Pisma bralcev

Pismo tov. R.Smokvina in odgovor urednika. Informatica 10 (1986), št. 4, str. 97.

EWSL 87

2nd European Working Session on Learning

Bled, Yugoslavia – May 13-15, 1987

CALL FOR PAPERS

EWSL is an annual meeting on Machine Learning enabling European researchers to present recent research results and the current European projects in this area. In particular, this event is also the main meeting of the participating laboratories of the Machine Learning and Knowledge Acquisition project which is part of the European initiative in Artificial Intelligence and Pattern Recognition COST-13. However, the participation is not limited to Europeans only, and papers from non European countries are expected as well. The first EWSL Meeting, held at Orsay, France, in February 1986, attracted about 60 participants representing large majority of the European research in this field.

TOPICS

All topics relevant to Machine Learning, Human Learning and Computer-Aided Instruction. The emphasis is on Machine Learning.

SUBMISSION OF PAPERS

Authors should submit five copies of full papers of no more than 5000 words no later than 20 January 1987 to the Programme Chairman:

Ivan Bratko (for EWSL)
Institute Jozef Stefan
Jamova 39
61000 Ljubljana
Yugoslavia
Tel. (+38) (61) 214 399

The following information has to be included on the title page: author's name, address, telephone number, telex number; abstract of 100-200 words; keywords.

The timetable is as follows:

- submission deadline: 20 January 1987
- notification of acceptance or rejection: 10 March 1987
- camera ready copy: 1 April 1987

The language of the conference is English; all papers submitted must be written in English. It is intended to publish a collection of conference papers in book form.

PROGRAMME COMMITTEE MEMBERS

FORMAT OF THE CONFERENCE

The conference will consist of presentations of submitted technical papers, short project progress reports, and in-depth panel discussions on special topics. Contributions for discussion on these special topics should be submitted directly to the panel organisers who will decide on the format of the discussions. The planned themes of the panel sessions and their organisers are as follows:

- Generalisation Techniques for Machine Learning

  Yves Kodratoff
  Laboratoire de Recherche en Informatique
  Batiment 490
  Universite de Paris-Sud
  91405 Orsay Cedex
  France

- The Role of Representation in Learning

  Robert Holte
  Tower C
  Brunel University
  Uxbridge UB8 3PH
  England

- Learning in Presence of Noise

Pavel Brazdil	Joel Quinqueton
University of Porto	INRIA
Faculdade de Economia	Domain de Voluceau
Rua Doctor R. Frias	Rocquencourt
4200 Porto	B.P. 105
Portugal	78153 Le Chesnay Cedex
	France

Communications to this panel can be sent to any address above.

LOCAL ORGANISATION

Nada Lavrac
Institute Jozef Stefan
Jamova 39
61000 Ljubljana
Yugoslavia
tel. (+38) (61) 214 399

Tiskarna Kresija