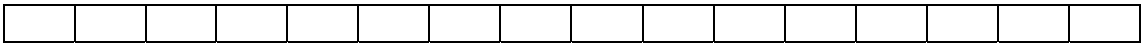
Mikroprogramavimas

Nėra griežtų ribų tarp kompiuterio aparatūrinės ir programinės įrangos. Be to tos ribos visą laiką kinta. Tradicinės įsivaizduojamos komandos ankstesniuose kompiuteriuose buvo realizuojamos atskiromis schemomis. Dabartiniuose kompiuteriuose visos tradicinės komandos yra interpretuojamos mikroprogramų pagalba. Mikroprogramos savo ruožtu sudarytos iš mikrokomandų, kurios realizuojamos aparatūriškai schemų lygyje. Mūsų tikslas yra susipažinti su mikroprogramininiu lygiu ir mikroprogramavimu.
Mikroprogramininio lygio tikslas yra užtikrinti aukštesnio lygio virtualios mašinos komandų interpretavimą.

Procesoriaus fizinio lygio komponentės

1. Registras – tai įrenginys, skirtas informacijos saugojimui. Registrai išdėstyti pačiame procesoriuje, jie yra analogiški atminties žodžiams, tačiau greitesni.
Tegul yra N registrų : 0, 1 , 2, 3, …, N-1 . Registrai charakterizuojami bitų skaičiumi. Tegul yra 16 bitų registras:

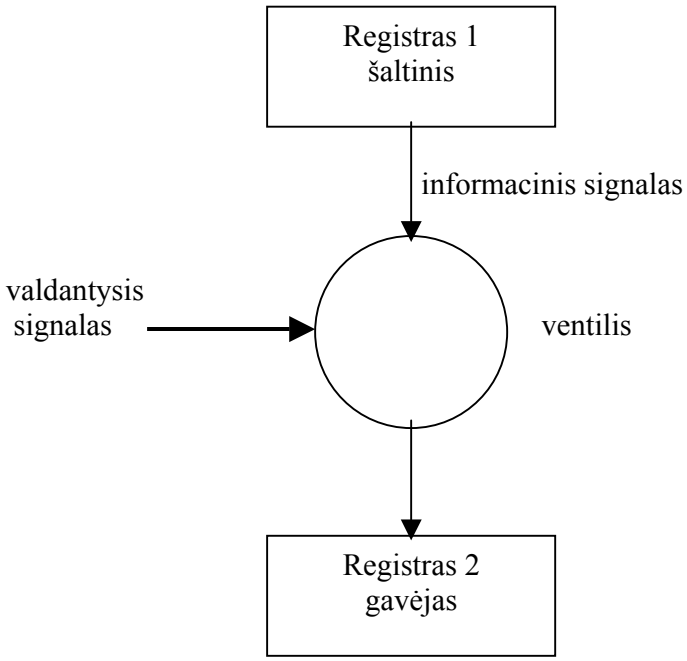


15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Skaitant iš registro, informacija jame nekinta.

2. Magistralė – tai elektros grandinė, jungianti du registrus.Magistralė susideda iš lygiagrečių laidininkų kiekvienam jungiamųjų registrų bitui. 16 bitų magistralės plotis yra 16. Fiziškai yra daugiau laidininkų, tačiau mus domina loginis lygis. Sakykime, kad magistrale informacija perduodama viena kryptimi.

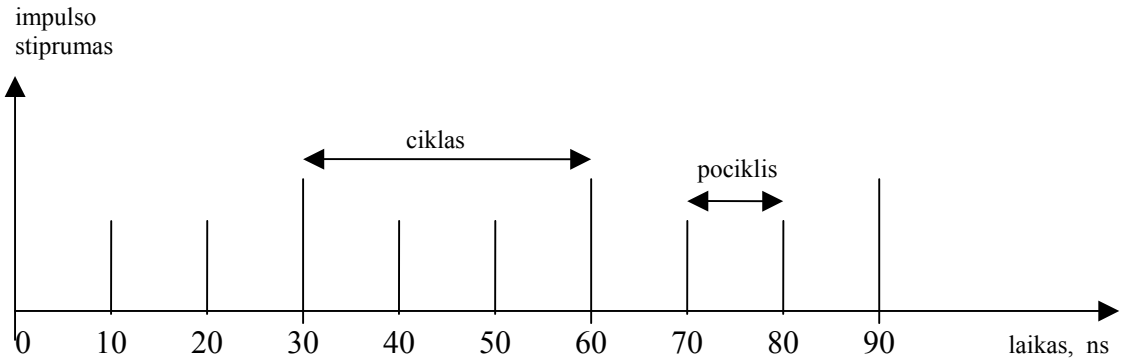
3. Ventiliai tipo ∧ (sakoma tipo ‘ir’). Jie skirti informacijos perdavimui iš registrų į magistrales ir iš magistralių į registrus. Ventilio panaudojimo schema:



Jeigu valdantysis signalas yra 1, tai ventilis tipo ‘ir’ informacinį signalą praleidžia, o jeigu valdantysis signalas yra 0, tai informacinis signalas nepraleidžiamas.

Valdantysis signalas	∧	Informacinis bitas	Rezultatas
1		1	1
1		0	0
0		1	0
0		0	0

4. Taktinių impulsų generatorius. Taktiniai impulsai naudojami procesų sinchronizavimui. Taktinių impulsų periodas vadinamas ciklu. Ciklo viduje silpnesnių impulsų intervalai vadinami pocikliais. Tokiu būdu laikas sudalinamas diskretiniais intervalais.

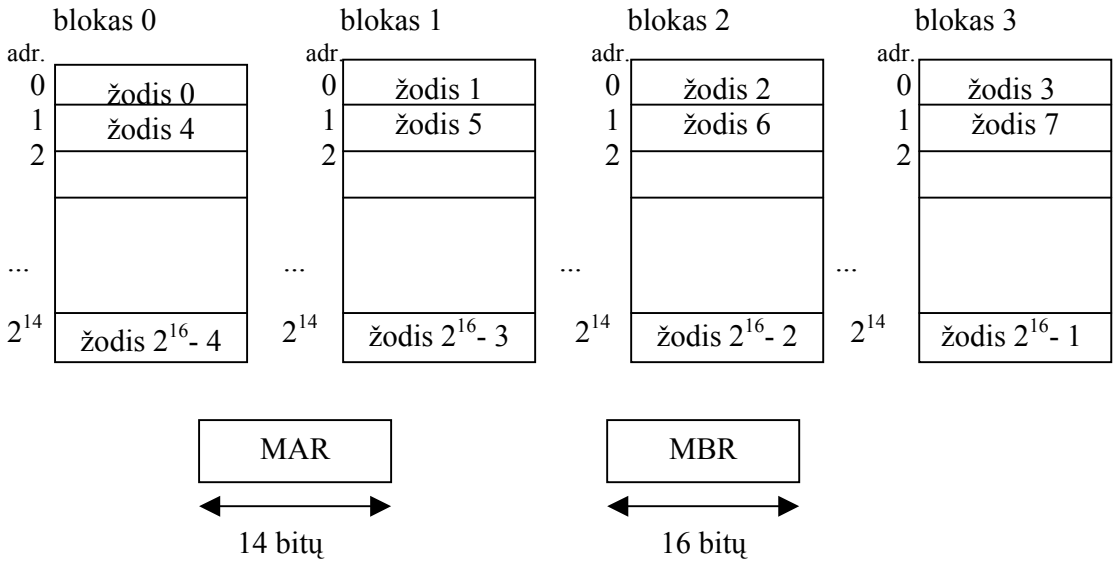


Pastaba: ns = 10^{-3} s
Pavyzdžiui ventilis intervale 0 – 20 galėtų būti uždarytas, o intervale 20 – 30 atidarytas.

5. Kreipimasis į atmintį. Kreipimuisi į atmintį naudojami du procesoriaus registrai: MAR – atminties adresinis registras ir MBR – buferinis registras. Jeigu ląstelės adresas yra MAR ir duotas skaitymo signalas, tai ląstelės turinys patalpinamas į MBR, o jeigu duotas rašymo signalas, tai iš MBR adresu MAR rašoma į ląstelę. Atminties ciklas tai yra laikas priimti skaitymo signalą, užrašyti ląstelės turinį į MBR ir pasiruošti sekančio signalo priėmimui. Atminties ciklas skiriasi nuo procesoriaus ciklo.

MBR bitų skaičius nustato atminties magistralės plotį. Kuo didesnis magistralės plotis, tuo mažiau reikia atminties ciklų informacijos fiksuotam kiekiui paimti. Tačiau atminties magistralės plotis ‘brangiai kainuoja‘.

Pastaba.
Apsikeitimo pagreitinimui galima naudoti lygiagretų apsikeitimą su keliais identiškais atminties blokais. Nagrinėkime įprasto dydžio atminties modulį – 64 K. Tegul yra 4 blokai po 2^{14} žodžių, kurie sumoje duoda 2^{16} žodžių:



Adreso tradicinėje komandoje du jauniausi bitai nustato bloką {0, 1, 2, 3}, o likę 14 bitų nurodo adresą bloke. Todėl *tokiu atveju* registro MAR dydis būtų 14 bitų.

6. Aritmetinis loginis įrenginys dažniausiai turi sudėties komandą, kartais atimties, paprastai logines komandas (and, or, ...). Komandų sistema mikrolygyje sudaroma taip, kad iš vienos pusės būtų galima realizuoti visus matematinius veiksmus, o būtinas minimumas yra sudėtis, nes atimtis tai yra sudėtis su papildomu kodu, daugyba – daug sudėčių cikle, o dalyba – daug atimčių.

Loginės komandos:

p	q	p ∧ q *and*	p ∨ q *or*	p ^ q *exclusiveor*	p ≡ q *equivalent*	p / q *nand*
0	0	0	0	0	1	1
0	1	0	1	1	0	1
1	0	0	1	1	0	1
1	1	1	1	0	1	0

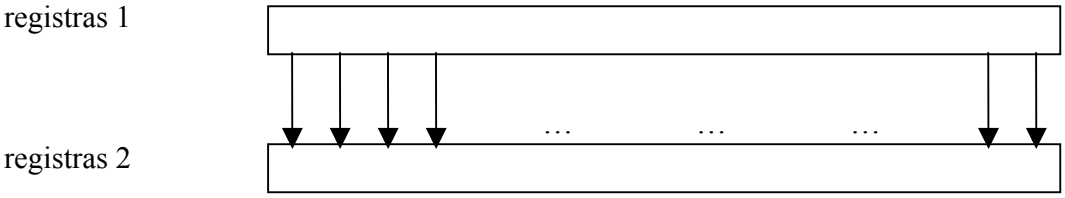
p ∧ q = (p / q) / (p / q)
p ∨ q = (p / p) / (q / q)
p ^ q = ((p / p) / q) / (p / (q / q))
p ≡ q = (p / q) / ((p / p) / (q / q))

Mikrokomandos

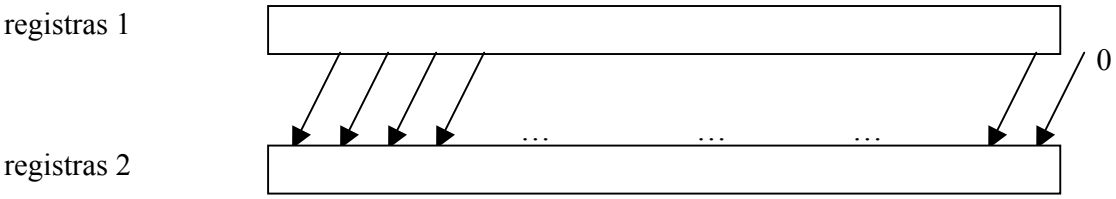
Pagrindinė komanda yra persiuntimas iš registro į registrą. Persiuntimas tarp registrų gali būti:

1. Tiesioginis
2. Asimetrinis
3. Dalinis
4. Grupinis

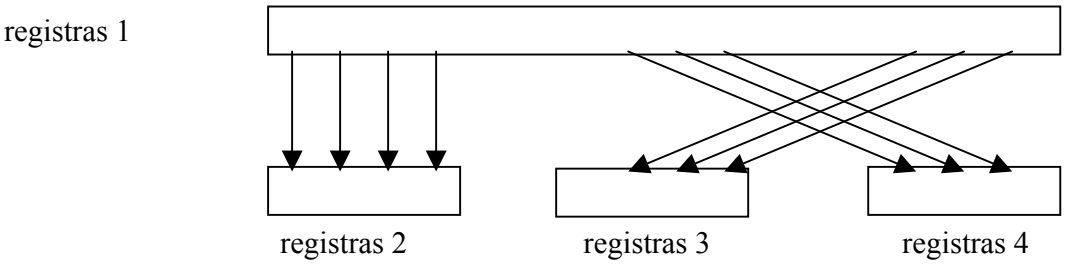
Dažniausiai naudojamas tiesioginis persiuntimas:



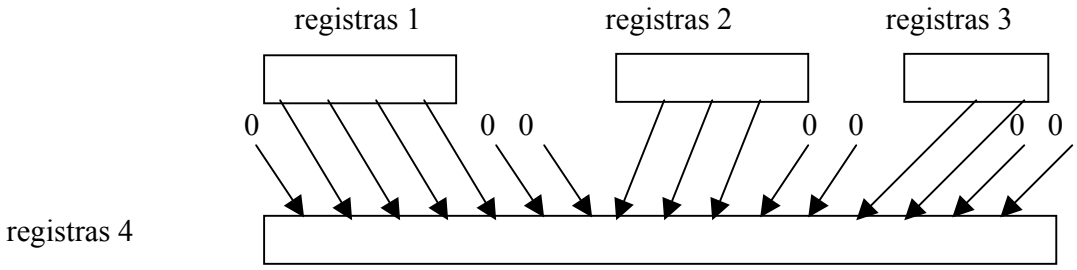
Asimetriniame persiuntime i-tasis bitas pereina į (i + k) –tąjį bitą, kai k > 0 arba k < 0:



Dalinis persiuntimas yra tada, kai vieno registro dalys yra persiunčiamos įvairiems registrams, neišlaikant bitų numeracijos, o tik nuoseklumą:



Esant grupiniam perdavimui įvairių registrų turinių dalys perduodamos vienam registrui:

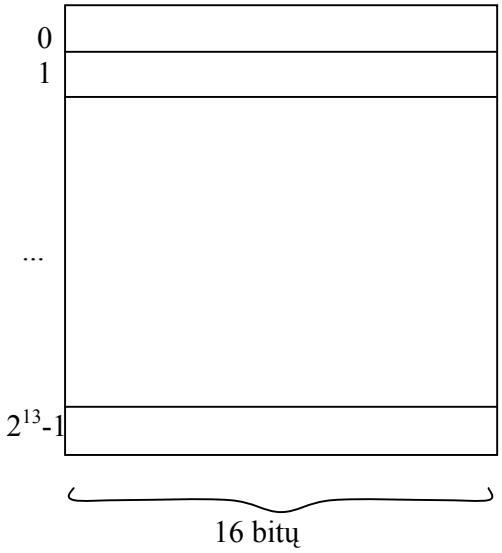


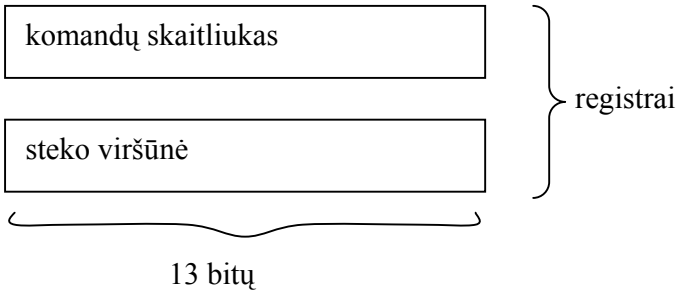
Pavyzdžiui, postūmio įrenginys veiksmui atlikti naudoja laikiną registrą ir asimetrinį persiuntimą *registras 1* → *registras 2* bei tiesioginį persiuntimą *registras 2* → *registras 1*. Postūmis per N bitų gaunamas pakartojus postūmį N kartų.
Postūmis į kitą pusę gali būti gautas tuo pačiu įrenginiu, bet tam reikia turėti kitą magistralę, kuri realizuoja kitos krypties asimetrinį persiuntimą.

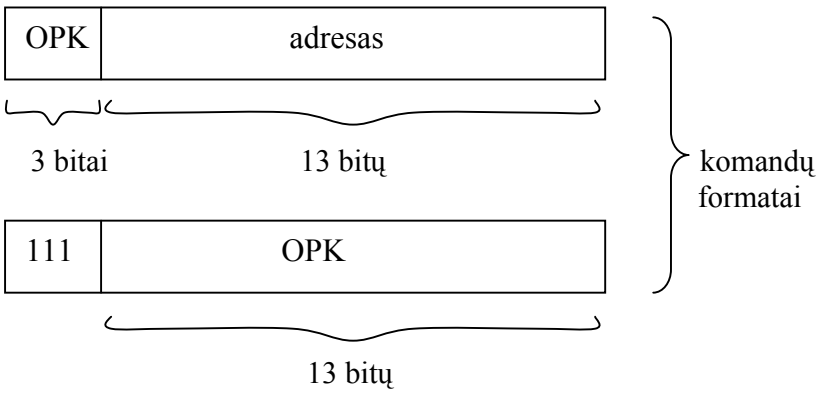
Kita komanda yra užrašymas į atmintį ir skaitymas iš atminties. Užrašymui ir skaitymui gali būti viena bendra arba dvi skirtingos komandos.

Trečia komanda yra sąlyginė. Tai yra arba speciali komanda, kuri tikrina bitą, arba komandoje laukas (‘maskė‘) naudojamas kartu su kokiu tai kitu registru (rezultato požymis).

Panagrinėsime konkretų tradicinį komandų lygį, kuris realizuojamas interpretavimo būdu ir vadinamas interpretuojamu lygiu. Tegul turime virtualią mašiną su 16 bitų žodžiais, kurie adresuojami 0, 1, 2, …, 2^{13} žodžių, o baitų 2^{14} . Adresinio registro ilgis yra 13 bitų, buferinio registro ilgis yra 16 bitų.







Tegul naudojamas stekinis atminties organizavimo principas, kai komandos operuoja su informacija, esančia steko viršūnėje, ir todėl tokios komandos neturi adreso lauko. Tam, kad užpildyti steką ir paimti iš steko saugojimui, naudojamos vieno adreso komandos. Be to adresinės komandos naudojamos ir besąlygiam bei sąlyginiam valdymo perdavimui. Sąlyginės valdymo perdavimo komandos paima iš steko viršūnės žodį, jį patikrina ir priklausomai nuo patikrinimo rezultatų perduoda valdymą. Procedūros iškvietimo komanda grįžimo adresą talpina į steką, o valdymą perduoda pagal adresą. Aritmetinės komandos pradžioje paima antrą operandą, po to pirmą operandą ir rezultatą patalpina į steką. Aritmetiniai veiksmai gali būti atliekami su skaičiais papildomu kodu.

Komandos:

PUSH	000	M

Komanda PUSH atminities žodis su adresu M patalpinamas į steko viršūnę.

POP	001	M

Komanda POP iš steko viršūnės paimama reikšmė (žodis) ir užrašoma adresu M.

JUMP	010	M

Komanda JUMP valdymas perduodamas adresu M.

JNEG	011	M

Komanda JNEG atliekamas sąlyginis valdymo perdavimas – pereiti į komandą adresu M, jei steko viršūnėje esanti reikšmė yra < 0 .

JZER	100	M

Komanda JZER atliekamas sąlyginis valdymo perdavimas – pereiti į komandą adresu M, jei steko viršūnėje esanti reikšmė yra = 0.

JPOZ	101	M

Komanda JPOZ atliekamas sąlyginis valdymo perdavimas – pereiti į komandą adresu M, jei steko viršūnėje esanti reikšmė yra >= 0.

CALL	110	M

Komanda CALL valdymas perduodamas adresu M, o grįžimo adresas išsaugomas steke.

ADD	111	0 0 0 0 0

Sudėtis: iš steko paimti operandus T ir S, juos sudėti ir rezultatą S + T užrašyti į steką.

SUB

111	0 0	0 0 0 1

Atimtis: iš steko paimti operandus T ir S, atimti ir rezultatą S - T užrašyti į steką.

MUL

111	0 0	0 0 1 0

Daugyba: iš steko paimti operandus T ir S, juos sudauginti ir rezultatą S · T užrašyti į steką.

DIV

111	0 0	0 0 1 1

Dalyba: iš steko paimti operandus T ir S, padalinti ir rezultatą S / T užrašyti į steką.

RETURN

111	0 0	0 1 0 0

Komanda RETURN iš steko viršūnės paimamas grįžimo adresas ir juo perduodamas valdymas.

Yra galimas beadresinių komandų praplėtimas, pavyzdžiui, loginių operacijų komandomis.

Interpretuojantis lygis

Interpretuojančio lygio architektūra turi 15 registrų, 39 magistrales su ventiliais, 16-os bitų sumatorių ir postūmio per viena bitą įrenginį.

Registrai:
PC – interpretuojamos mašinos komandų skaitliukas, registras yra 13 bitų;
SP – interpretuojamos mašinos steko rodyklė, registas yra 13 bitų;
IR – 16 bitų registras interpretuojamos mašinos vykdomos komandos saugojimui;
MAR – 13 bitų interpretuojamos mašinos adresinis registras;
MBR – 16 bitų interpretuojamos mašinos buferinis registras;

A
B
C
D
} mikroprogramų darbiniai registrai, 16 bitų;

X – mikroprogramų ciklų skaitliukas, 16 bitų;

Yra dar penki registrai su pastoviomis reiksmėmis:
+1
0
-1
15
} konstantiniai registrai;

SIGN registro reikšmė yra - 32768

Aritmetinis loginis įrenginys (ALI).

ALI susideda iš sumatoriaus, kurio įėjime yra du šešioliktainiai skaičiai, o išėjime yra vienas skaičius – suma, išreikšta papildomu kodu, kuris iškart paduodamas postūmio įrenginiui. Postūmis gali būti į kairę arba į dešinę, pastūmus atsiranda nuliniai bitai. Ženklo bitas stumiant į dešine nepereina į jauniausią bita (pvz., pastūmus per vieną į dešinę reikšmę 1000 0000, turėtume 0000 0000).

Magistralės ir ventiliai.

I. *Sumatoriaus įėjimas*.

Kairysis įėjimas

1. X → S naudojama atėmimui 1 iš X, ciklų skaičiavimui.
2. SP → S naudojama vienu padidinti arba sumažinti SP – steko viršūnę.
3. PC → S naudojama PC perdavimui į atmintį per MBR (PC dydis yra 16 bitų, MBR dydis yra 13 bitų, taigipločių skirtumas yra 3).
4. A → S naudojama aritmetinėse operacijose bei perduodant D ir MBR.
5. B → S naudojama aritmetinėse operacijose bei perduodant A, D ir MBR.
6. MBR → S naudojama aritmetinėse operacijose.
7. +1 → S naudojama pridėti 1 prie registre esančios reikšmės.
8. 0 → S naudojama persiuntimui, pavyzdžiui C → A .
9. -1 →S naudojama atimti 1 iš registro.
10. SIGN → S naudojama 32 bitų postūmiuose, kurie reikalingi sujungus du 16 bitų registrus, taip pat pernešimo bitų įskaitymui.

Dešinysis įėjimas.

11. SIGN → S analogiškai 10 magistralei.
12. -1 → S analogiškai 9 magistralei.
13. 0 → S analogiškai 8 magistralei, naudojama perdavimui iš A arba B į D arba MBR.
14. +1 → S analogiškai 7 magistralei.
15. MBR → S analogiškai 6 magistralei.
16. C → S naudojama aritmetinėse operacijose bei perdavimui iš C į A, D ir MBR
17. 0 → S naudojama aritmetinėse operacijose bei perdavime iš D į A, MBR.

II.*Perdavimas iš registro į registrą.*

18. IR → PC naudojama vykdant valdymo perdavimo komandas, magistralės plotis yra 13 bitų.
19. PC → MAR naudojama komandos paėmimui iš atminties.
20. SP → MAR naudojama darbui su steko viršūne.
21. IR → MAR naudojama komandoms PUSH ir POP. Magistralės plotis 13 bitų.
22. MBR → IR naudojama vykdomos komandos paėmimui.
23. MBR → A
24. MBR → B } registrų užkrovimas
25. MBR → C

26. MBR → D
27. 15 → X naudojama ciklinėje operacijoje pradinei skaitliuko reikšmei suformuoti.

III.*Papildomo kodo sukūrimas ir postūmis.*

28. Kai ši magistralė atidaryta, kairysis signalas patenka į sumatorių papildomu kodu.
29. Kai ši magistralė atidaryta, dešinysis signalas patenka į sumatorių papildomu kodu.
30. Kai ši magistralė atidaryta, sumatoriaus išėjimo signalas pastumiamas per vieną bitą į kairę prieš perduodant priėmėjui registrui.
31. Kai ši magistralė atidaryta, sumatoriaus išėjimo signalas pastumiamas per vieną bitą į dešinę prieš perduodant priėmėjui registrui.

IV.*Sumatoriaus įėjimas.*

32. S → X naudojama ciklinėse operacijose keičiant X reikšmę nuo 15 iki -1.
33. S → A naudojama rezultatui užrašyti į registrą A.
34. S → SP naudojama atliekant veiksmus su steku, vienu didinant arba mažinant SP – steko viršūnės registrą.
35. S → PC naudojama perėjime prie sekančios komandos.
36. S → MBR naudojama rezultatui užrašyti į atmintį.
37. S → D naudojama rezultatui užrašyti į registrą D.

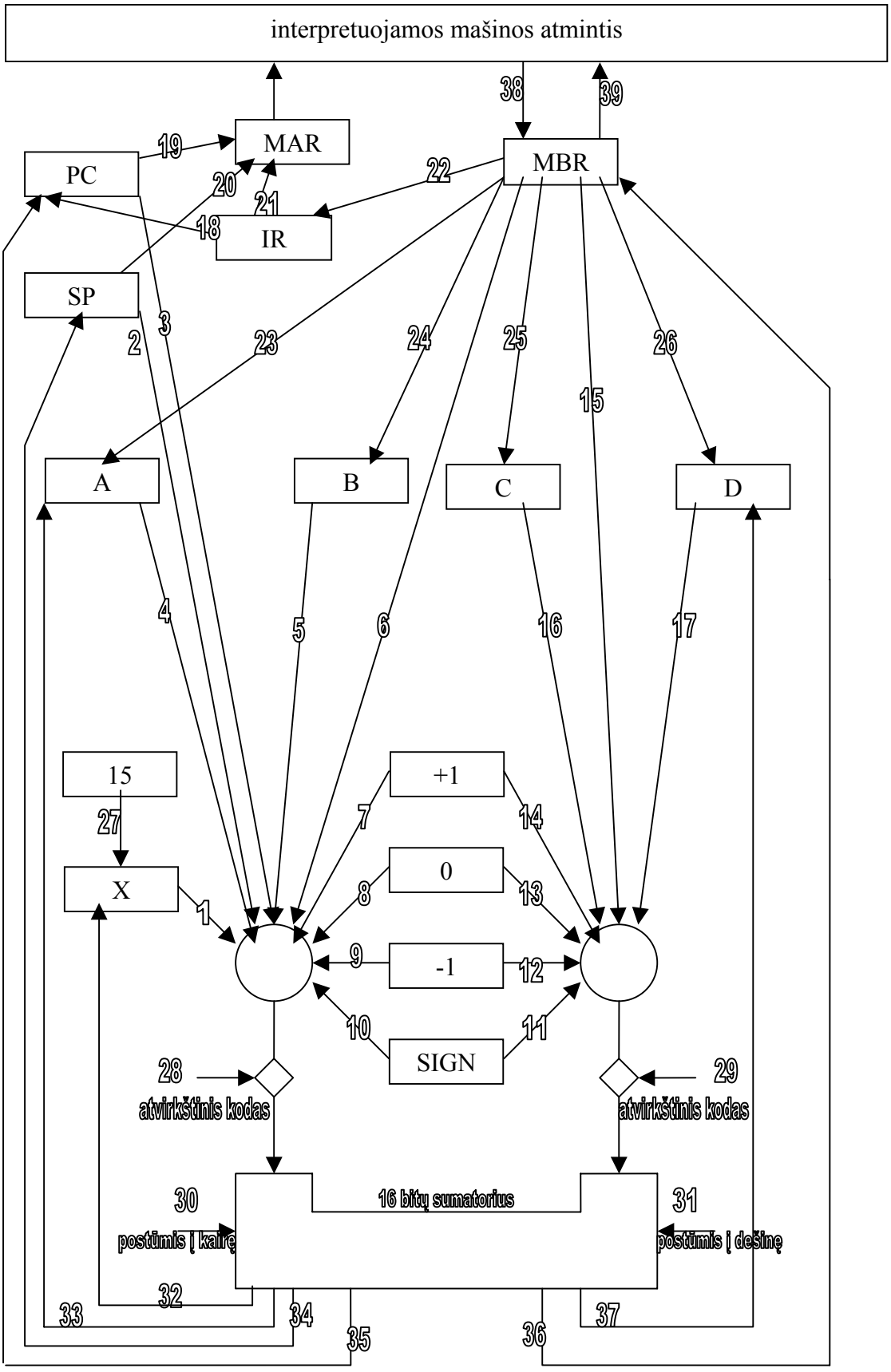
V.*Apsikeitimas su interpretuojamos mašinos atmintimi.*

38. → MBR iš atminties imamas žodis, kurio adresas yra MAR, ir jis užrašomas į atmintį.
39. MBR → žodis, esantis MBR užrašomas į atmintį adresu MAR.

Magistralė, jungianti registrą MAR su interpretuojamos mašinos atmintimi, neturi ventilio. Todėl signalas, einantis šia magistrale, nėra programiškai valdomas. Ši magistalė visada yra atidaryta.

Perdavimas iš registro gali būti lygiagretus, bet perdavimas į registrą turi būti nuoseklus.

Mikroprograminio lygio architektūra



Ventilių darbas

Interpretuojamos mašinos komanda ADD gali būtu įvykdyta per keturis žingsnius:
1. Paimti antrą operandą iš steko ir užrašyti į registrą.
2. Paimti pirmą operandą iš steko ir užrašyti į registrą.
3. Suformuoti sumą.
4. Sumą užrašyti į steką.

Šie keturi žingsniai gali būti įvykdyti per tris ciklus, inicijuojamus taktiniais impulsais. Atidarant ir uždarant ventilius valdoma perdavimų sinchronizacija: ryšyje *registras 1 → registras 2 → registras 3,* perduoti *registras 2 → registras 3* galima tik tada, kai *registras 1 → registras 2* signalas jau atėjo. Priešingu atveju gali kilti neapibrėžtumas to, kas bus paduota į *registrą 3* .

Pocikliai

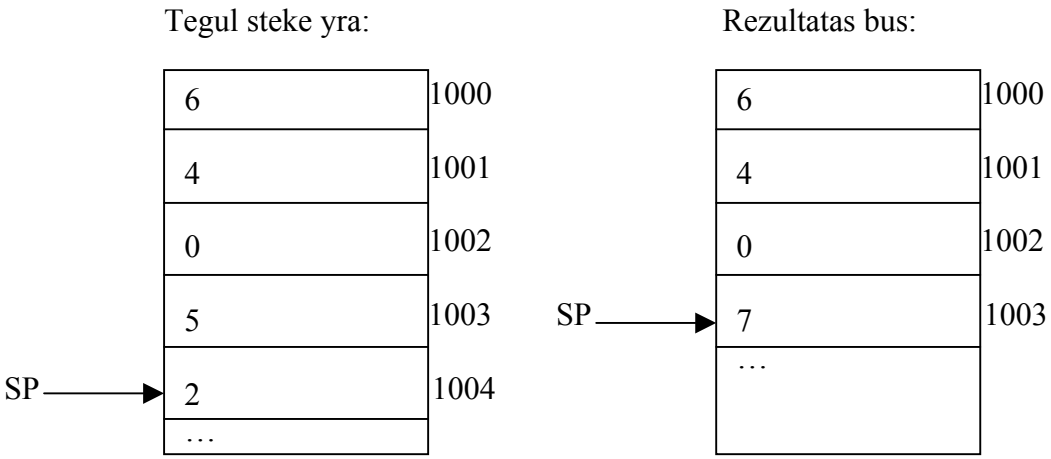
Tam, kad išvengti neapibrėžtumų yra įvedami pocikliai.
Pirmame pociklyje gali būti atidarytos magistralės 1 – 29 ;
antrame pociklyje – magistralės 30 – 37 ;
trečiame pociklyje – 38 arba 39 magistralės.

Pirmame pociklyje atliekamas apsikeitimas tarp registrų; antrame pociklyje atliekami veiksmai sumatoriuje, postūmio įrenginyje, rezultatai paduodami į registrus; trečiame pociklyje apsikeičiama su atmintimi.

Realiai atminties ciklas yra didesnis nei procesoriaus ciklas, bet paprastumo dėlei į tai nekreipiame dėmesio.

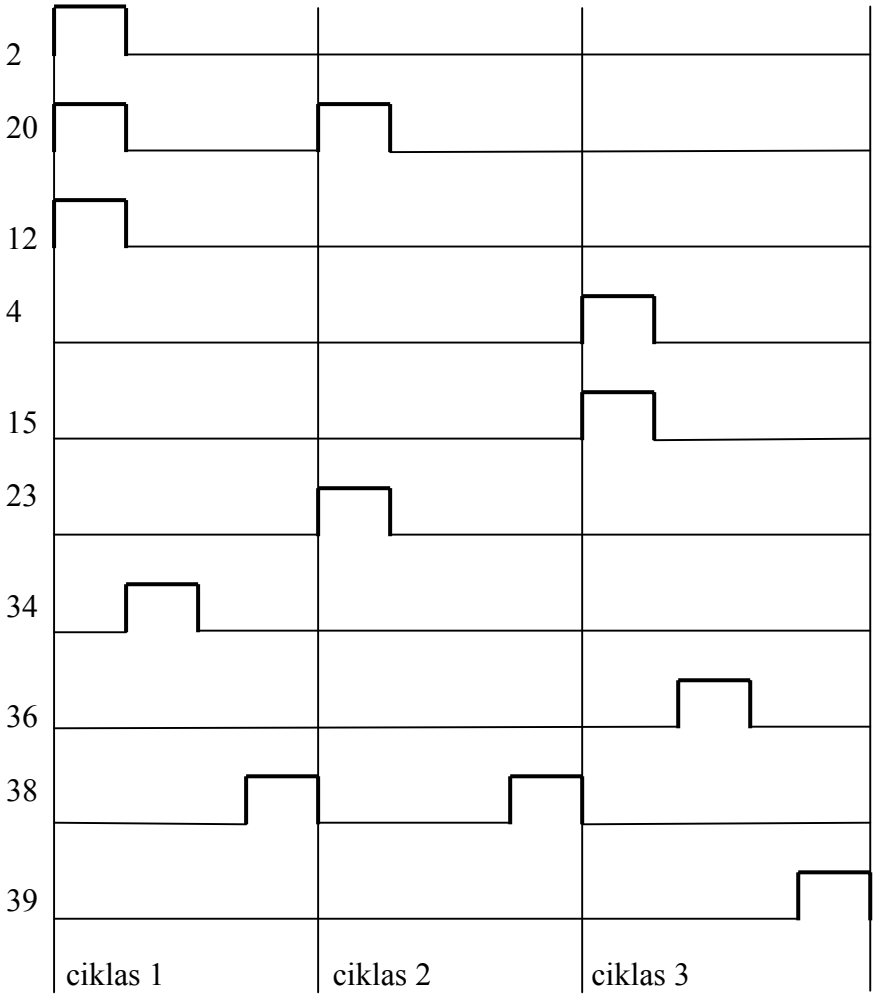
Komandos ADD realizacija

Tegul komanda jau yra registre IR ir yra nustatytas operacijos kodas.



	Pocikliai	SP	MAR	MBR	A	Schemos
ciklas1	1	1004	?	?	?	20, 2, 12
	2	1004	1004	?	?	34
	3	1003	1004	?	?	38
ciklas 2	1	1003	1004	2	?	20, 23
	2	1003	1003	2	2	
	3	1003	1003	2	2	38
ciklas 3	1	1003	1003	5	2	4, 15
	2	1003	1003	5	2	37
	3	1003	1003	7	2	39

Pavaizduosime schematiškai magistralių atidarymą, realizuojantį sudėties komandą:

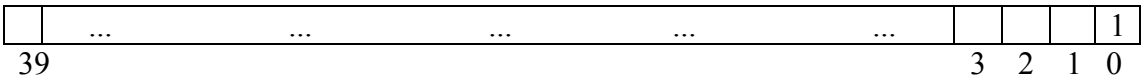


Ventilių mikroprogramins valdymas

Tam, kad gauti interpretuojamos mašinos programos vykdymo efektą, reikia atitinkamais laiko momentais atidarinėti ir uždarinėti atitinkamus ventilius. Interpretuojančio lygio komandų sistema susideda iš dviejų komandų: GATE ir TEST. Operacijos kodas yra reikšmė 1 (komanda GATE) arba 0 (komanda TEST) dešiniajame bite.

Mikrokomanda TEST naudojama mikrokomandų GATE vykdymo nuoseklumui pakeisti, priklausomai nuo tam tikrų sąlygų.

GATE

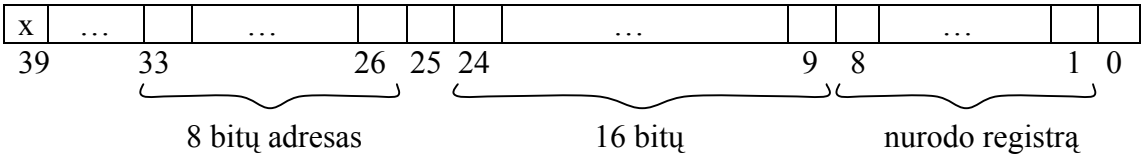


Nulinis bitas yra operacijos kodas. Likusieji 39 komandos bitai atitinka 39 ventilius, bito reikšmė 1 reiškia, kad atitinkamas ventilis turi būti atidarytas, o bito reikšmė 0 reiškia, kad atitinkamas ventilis turi būti uždarytas. Iš viso komanda užima 40 bitų.

Mikroprograma, išreiškianti sudėtį:

39	38	37		35	34	33		24	23	22	21	20	19		16	15	14	13	12	11		4	3	2	1	0
0	1	0	..	0	1	0	..	0	0	0	0	1	0	..	0	0	0	0	1	0	..	0	0	1	0	1
0	1	0	..	0	0	0	..	0	1	0	0	1	0	..	0	0	0	0	0	0	..	0	0	0	0	1
1	0	1	..	0	0	0	..	0	0	0	0	0	0	..	0	1	0	0	0	0	..	1	0	0	0	1

TEST



1 – 8 bitais pasakome, kuris registras dalyvaus sąlygos formulavime: bito, atitinkančio registrą, reikšmė yra 1.
1-as bitas atitinka registrą A;
2-as bitas atitinka registrą B;
3-as bitas atitinka registrą C;
4-as bitas atitinka registrą D;
5-as bitas atitinka registrą MBR;
6-as bitas atitinka registrą X;
7-as bitas atitinka registrą IR;
8-as bitas atitinka registrą Ø.

Registrai yra 16 bitų. 9 – 24 bite reikšmė 1 nurodo, kuris nustatyto registro bitas dalyvaus sąlygos formulavime. 9-tas bitas atitinka nulinį registro bitą, 24-tas bitas atitinka penkioliktąjį registro bitą.

25-tas bitas yra konstanta, jis yra palyginimo bitas.
Kiekviena mikrokomanda TEST apibrėžia vieną tikrinamą bitą iš 128 galimų. Tokiais bitais yra registrų A, B, C, D, MBR, X, IR, Ø visi 16 bitų ($8 \cdot 16 = 128$). Jeigu nustatytas bitas ir 25-tasis komandos bitai sutampa, tai mikrokomandų vykdymo nuoseklumas pakinta ir sekančios vykdomos mikrokomandos adresas nustatomas mikrokomandos TEST 26 – 33 bitais. Priešingu atveju vykdymo nuoseklumas nepasikeičia.

Bitų grupėse 1 – 8 ir 9 – 24 tik vieno bito reikšmė kiekvienoje grupėje gali būti 1.

Kadangi mikrokomandos ilgis yra 40 bitų, dar turime nepanaudotus 26 – 39 bitus, t.y., 14 bitų. Galėtume jais adresuoti 2^{14} mikroatminties žodžių (mikroatminties žodžio ilgis yra 40 bitų). Tačiau interpretatoriui parašyti pakanka $2^8 = 256$ mikroatminties žodžių, todėl adresui nurodyti naudojame 8 bitus, t.y., 26 – 33 bitus, o likusieji 34 – 39 yra nenaudojami.

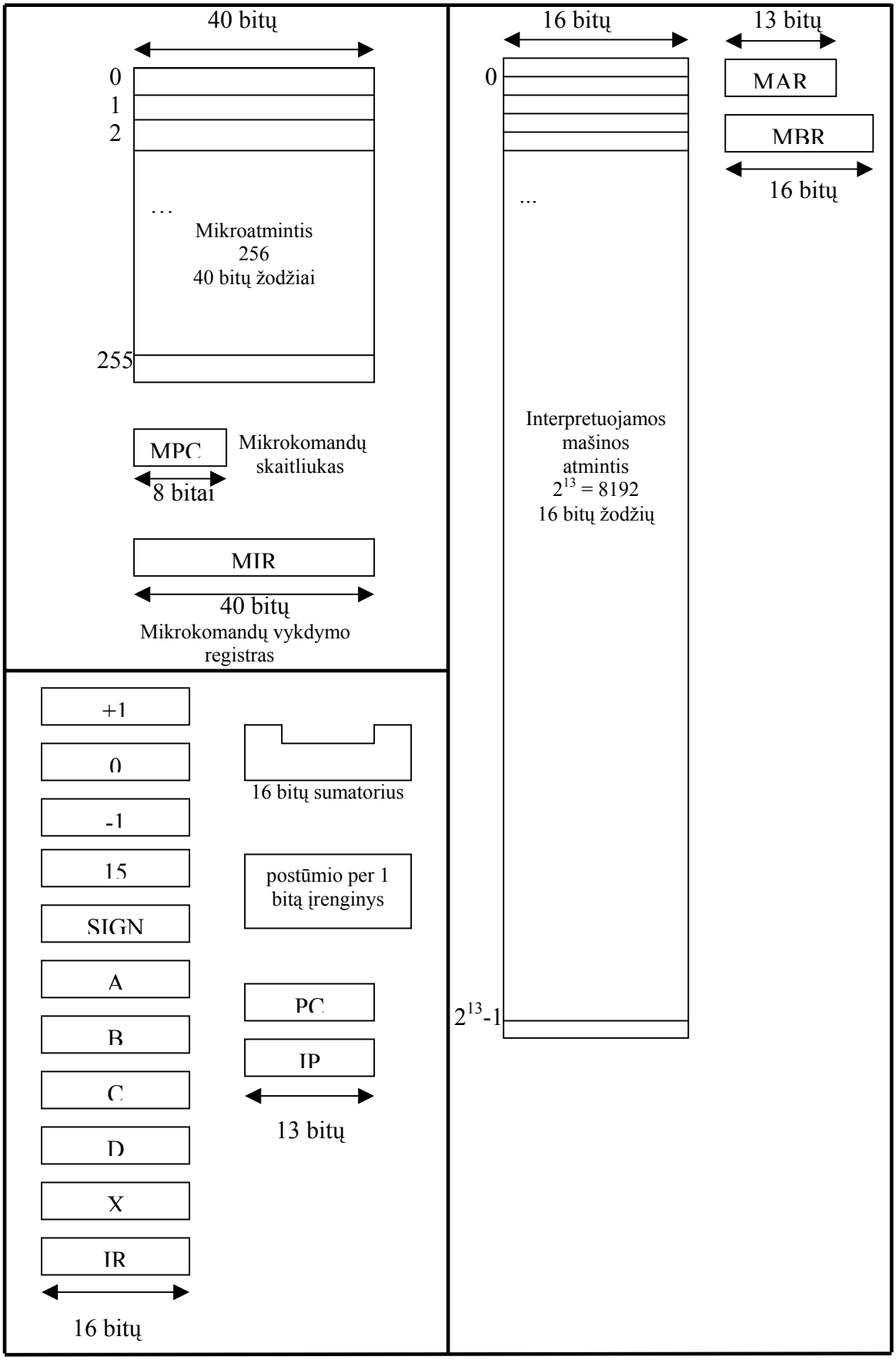
Mikroprograma saugoma mikroatmintyje, kuri logiškai skiriasi nuo interpretuojamos mašinos atminties. Fiziškai abi atmintys gali būti realizuotos neatskirtos, tik paskirstant į dvi dalis. Mikroatminties žodžio ilgis yra 40 bitų. Kaip jau buvo minėta, apsiribosime 2^8 žodžių adresine erdve.

Mikroprogramininio lygio architektūroje turi būti mikrokomandų skaitliukas, atminties adresinis registras, atminties buferinis registras, mikrokomandų vykdymo registras. Dėl mikroprogramininio lygio paprastumo atminties adresinį registrą galima sutapatinti su mikrokomandų skaitliuku, o atminties buferinį registrą – su mikrokomandų vykdymo registru. MPC bus mikroprogramos komandų skaitliukas, o MIR – mikrokomandų interpretavimo (vykdymo) registras.

Aprašysime centrinio procesoriaus darbą. Taktinis impulsas užduoda naujo ciklo pradžią. Atminties žodis su adresu iš MPC užrašomas į registrą MIR. Jeigu tai yra mikrokomanda GATE, tai atitinkami ventiliai, nurodyti vienetiniais bitais, atidaromi atitinkamo pociklio bėgyje. Jeigu tai yra mikrokomanda TEST, tai nustatomas reikiamas bitas ir jis patikrinamas. Jeigu bitai sutampa, tai mikrokomandos bitai 26 – 33 pasiunčiami į MPC. Mikrokomanda yra ivykdoma ciklo bėgyje, o atitinkami ventiliai atidaromi atitinkamame pociklyje.

Sudarant mikroprogramas galima operuoti 40 bitų komandomis, bet galimas ir kitas kelias, kai pradžioje apibrėžiamos ir realizuojamos bazinės operacijos mikrokomandų lygyje. Vėliau reikalingos mikroprogramos gaunamos atitinkamos programos, sudarytos interpretuojamos mašinos kalba, darbo rezultate.

Interpretuojančios mašinos architektūra



Mikroprogramavimo kalba

Programuoti mikrokomandų kodais yra labai sudėtingas darbas ir todėl reikalinga mikroprogramavimo kalba. Naudosime kalbą MPL (Mikro Programming Language) . Ypatumas yra tame, kad mikrokomandos užduoda lygiagrečiai vykdomus veiksmus, persiuntimus įvairiomis magistralėmis. Kiekviena MPL eilutė apibrėžia vieną mikrokomandą, net jeigu vienoje eilutėje yra keli MPL operatoriai. Mikroprogramos vienos eilutės operatoriai vykdomi lygiagrečiai.

Mikrokomandos GATE programavimas.
Perdavimai tarp registrų nurodomi priskyrimo operatoriais.
Pavyzdžiui:
A = MBR;
reiškia, kad 23 magistralė vykdant šį operatorių yra atidaryta.

Operatorių nuoseklumas yra nesvarbus eilutės ribose.
A = MBR; MAR = IR; X = 15;
MAR = 12; X = 15; A = MBR;
Abi eilutės reiškia tą patį – 21, 23 ir 27 magistralių atidarymą.

Tačiau operatoriai :
A = MBR ;
MAR = IR ;
X = 15;
reiškia tris atskiras mikrokomandas.

Sumatoriaus panaudojimas nurodomas operacija '+' .
MBR = A + C; reiškia 4, 16 ir 36 magistralių atidarymą.
SP = SP + (-1); reiškia 2, 12 ir 34 magistralių atidarymą. Tai yra steko rodyklės sumažinimas vienetu.
A = MBR; PC = PC + 1; reiškia 3, 14, 23 ir 35 magistralių atidarymą.
D = 0 + C; reiškia 8, 16 ir 37 magistralių atidarymą. Registrai C ir D tiesiogiai nesujungti, todėl norėdami perduoti iš C į D naudojamės sumatoriumi.

Atvirkštinis kodas gaunamas atidarant 28 arba 29 magistrales. Tokį veiksmą atitinka funkcija COM. 28 ir 29 magistralės valdo reikšmės konvertavimą, tačiau norint gauti reikšmę su priešingu ženklu, reikia dar pridėti vienetą. Pavyzdžiui, norėdami iš 5 gauti -5, elgiamės taip:

0000 0101	= 5
1111 1010	invertacija
1	pridedame vienetą
1111 1011	= -5

Pavyzdžiui:
MBR = COM(MBR) + 1; reiškia 6, 14, 28, 36 magistralių atidarymą. Tai yra buferiniame registre esančios reikšmės ženklo pakeitimas.

A = 0 + COM(SIGN); reiškia 8, 11, 29, 33 magistralių atidarymą. Tai yra maksimalios teigiamos reikšmės priskyrimas registrui A.

Sumatoriaus išėjime reikšmę galima pastumti per vieną bitą pasinaudojant magistralėmis 30 ir 31. Postūmiai į kairę arba į dešinę nurodomi funkcijomis LEFT_SHIFT ir RIGHT_SHIFT. Pavyzdžiui:
MBR = LEFT_SHIFT(1 + 1);reiškia 7, 14, 30, 36 magistralių atidarymą. Tokiu veiksmu į buferinį registrą užrašome reikšmę 4.
A = RIGHT_SHIFT(A + 0); reiškia 4, 13, 31, 33 magistralių atidarymą. Tokiu veiksmu į registrą A užrašoma jame buvusi reikšmė, padalinta iš dviejų.

Apsikeitimas su atmintimi (skaitymas ir rašymas) yra atliekamas panaudojant funkciją MEMORY(MAR), kuri iššaukia 38 arba 39 magistralės atidarymą.

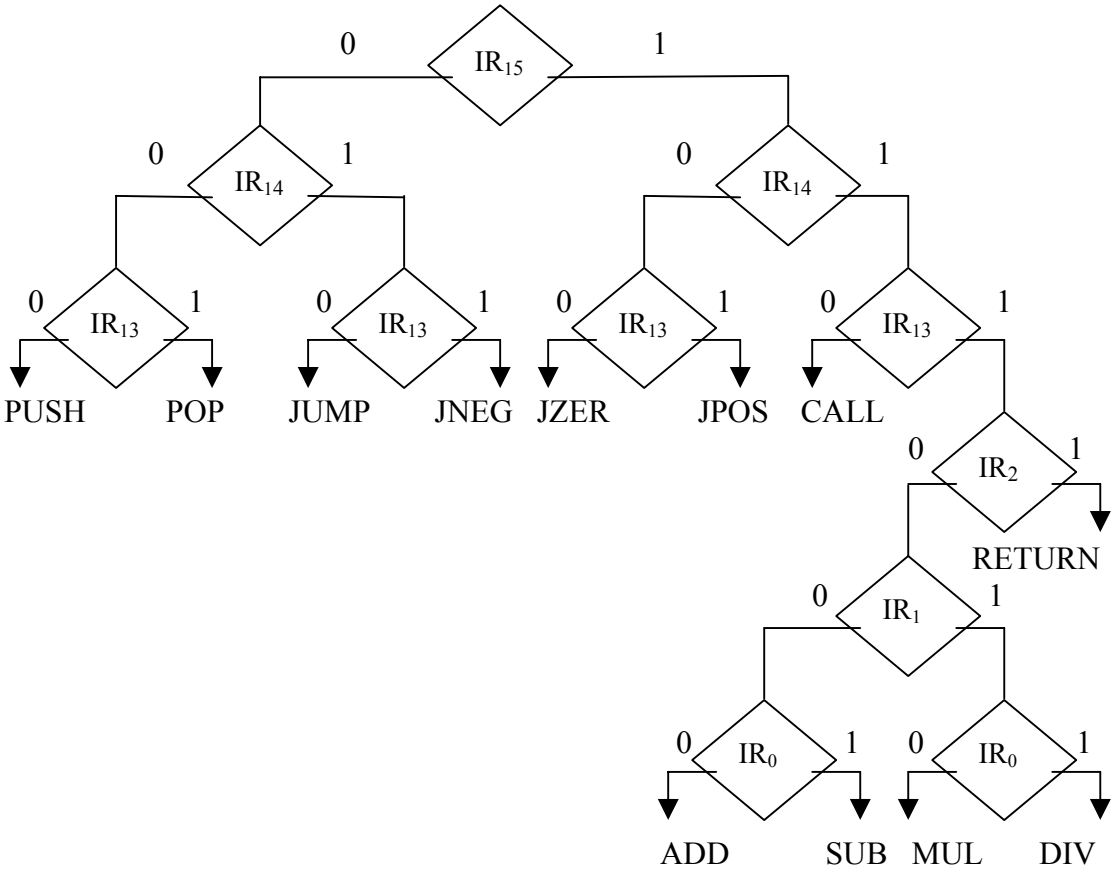
Mikrokomandos TEST užrašymas.
IF BIT(*k, reg*) = *b* THEN GOTO *label*;
k yra bito numeris registre, *k* = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};
reg yra registras, *reg* = {A, B, C, D, MBR, IR, X, Ø};
b atitinka 25-tą komandos TEST bitą, *b* = {0, 1};
label nurodo vardą (žymė yra simbolinis vardas), kuriuo perduodamas valdymas, atitinka 26 – 33 komandos TEST bitus.

Jeigu *b* = 0, *reg* = Ø ir *k* yra bet kuris bitas, tada valdymo perdavimas yra besąlyginis, tai yra užrašoma GOTO *label*.

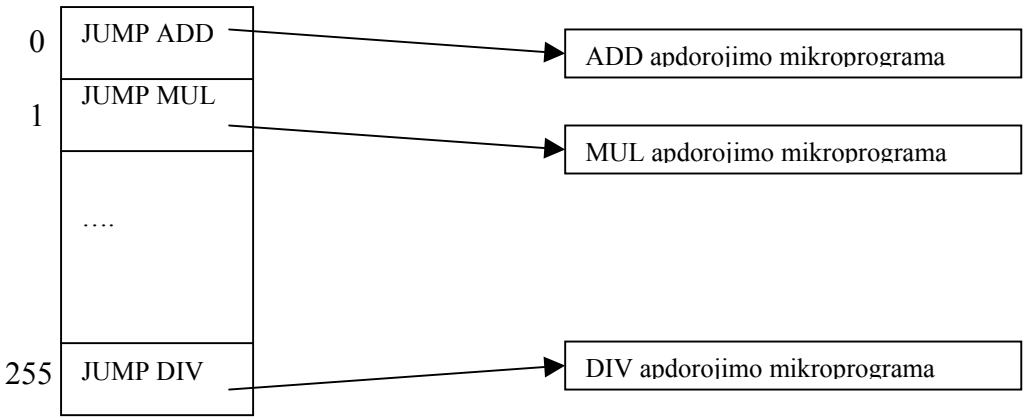
Pavyzdžiai:
IF BIT(13, IR) = 1 THEN GOTO POP;
IF BIT(15, X) = 0 THEN GOTO NULLOOP;

Iš esmės MPL yra asemblerio lygio kalba, nes ji leidžia valdyti generuojamą kodą.

Operacijos kodo dešifratoriaus algoritmas



Šis operacijos dekodavimo būdas yra nepakankamai efektyvus. Jeigu operacijos kodas būtų 8 bitų, tai reikėtų iki 8 mikrokomandų TEST. Tam, kad to išvengti, galima būtų panaudoti operacijos kodo reikšmę kaip modifikaciją perduodant valdymą mikrokomandai, apdorojančiai atitinkamą operaciją. Tiksliau tai būtų mikrokomandos TEST, kurios perduotų valdymą į atitinkamos interpretuojamos komandos apdorojimo mikroprogramine šaką. Taigi tegu operacijos kodas yra 8 bitai ir tegu komandos ADD operacijos kodas yra 0000 0000, komandos MUL operacijos kodas yra 0000 0001, ir t.t., tada aukščiau aprašyta sprendimą galima pavaizduoti taip:



Tokios galimybės realizacijai reikėtų dalinai modofikuoti mūsų interpretuojančios mašinos architektūrą.

Interpretuojamos mašinos intepretatorius

```
0     MAINLOOP: MAR = PC; MBR = MEMORY(MAR);
1               IR = MBR; PC = PC + 1;
2               IF BIT(15, IR) = 1 THEN GOTO OP4567;
3               IF BIT(14, IR) = 1 THEN GOTO OP23;
4               IF BIT(13, IR) = 1 THEN GOTO POP;
5     PUSH:      MAR = IR; SP = SP + 1; MBR = MEMORY(MAR);
6               MAR = SP; MEMORY(MAR) = MBR;
7               GOTO MAINLOOP;
8     POP:       MAR = SP; SP = SP + (-1); MBR = MEMORY(MAR);
9               MAR = IR; MEMORY(MAR) = MBR;
10              GOTO MAINLOOP;
11    OP23:      IF BIT(13,IR) = 1 THEN GOTO JNEG;
12    JUMP:      PC = IR;
13              GOTO MAINLOOP;
14    JNEG:      MAR = SP; SP = SP + (-1); MBR = MEMORY(MAR);
15              IF BIT(15, MBR) = 1 THEN GOTO JUMP;
16              GOTO MAINLOOP;
17    OP4567:    IF BIT(14, IR) = 1 THEN GOTO OP67;
18              IF BIT(13, IR) = 1 THEN GOTO JPOS;
19    JZER:      MAR = SP; SP = SP + (-1); MBR = MEMORY(MAR);
20              IF BIT(15, MBR) = 1 THEN GOTO MAINLOOP;
21              MBR = MBR + (-1);
22              IF BIT(15, MBR) = 1 THEN GOTO JUMP;
23              GOTO MAINLOOP;
24    JPOS:      MAR = SP; SP = SP + (-1); MBR = MEMORY(MAR);
25              IF BIT(15, MBR) = 0 THEN GOTO JUMP;
26              GOTO MAINLOOP;
27    OP67:      IF BIT(13,IR) = 1 THEN GOTO OP7;
28    CALL:      SP = SP + 1;
29              MAR = SP; MBR = PC + 0; MEMORY(MAR) = MBR;
30              GOTO JUMP;
31    OP7:       IF BIT(2, IR) = 1 THEN GOTO RETURN;
32              IF BIT(1, IR) = 1 THEN GOTO MULDIV;
33    ADDSUB:    MAR = SP; SP = SP + (-1); MBR = MEMORY(MAR);
34              IF BIT(0, IR) = 0 THEN GOTO SUM;
35              MBR = COM(MBR) + 1;
36    SUM:       MAR = SP; A = MBR; MBR = MEMORY(MAR):
37              MBR = A + MBR; MEMORY(MAR) = MBR;
38              GOTO MAINLOOP;
39    MULDIV:    IF BIT(0, IR) = 1 THEN GOTO DIV;
40    MUL:       MAR = SP; SP = SP + (-1); MBR = MEMORY(MAR);
41              C = MBR; MAR = SP; MBR = MEMORY(MAR);
42              X = 15; A = 0 + 0; D = 0 + 0;
43              IF BIT(15, MBR) = 0 THEN GOTO MULLOOP;
44              MBR = COM(MBR) + 1;
45              B = MBR; MBR = 1 + COM(C);
46              C = MBR; MBR = B + 0;
47    MULLOOP:   IF BIT(0, MBR) = 0 THEN GOTO NOADD;
```

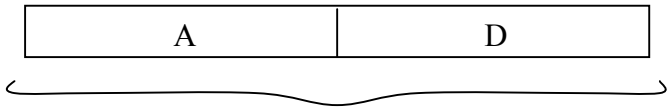
```
48                  A = A + C;
49      NOADD:      MBR = RIGHT_SHIFT(MBR + 0);
50                  D = RIGHT_SHIFT(0 + D);
51                  IF BIT(0,A) = 0 THEN GOTO NOCARRY;
52                  D = SIGN + D;
53      NOCARRY     A = RIGHT_SHIFT(A + 0);
54                  IF BIT(15, C) = 0 THEN GOTO MULEND;
55                  IF BIT(14, A) = 0 THEN GOTO MULEND;
56                  A = A + SIGN;
57      MULEND:     X = X + (-1);
58                  IF BIT(15, X) = 0 THEN GOTO MULLOOP;
59                  MBR = 0 + D; MEMORY(MAR) = MBR;
60                  GOTO MAINLOOP;
61      DIV:        MAR = SP; SP = SP + (-1); MBR = MEMORY(MAR);
62                  C = MBR; B = MBR; MAR = SP; A = 0 + 0;
                                                MBR = MEMORY(MAR);
63                  D = MBR; MBR = COM(MBR) + 1; X = 15;
64                  IF BIT(15, MBR) = 1 THEN GOTO DIVPOS;
65                  D = MBR; MBR = COM(B) + 1; B = MBR;
66      DVDPOS:     MBR = 1 + COM(C);
67                  IF BIT(15, C) = 0 THEN GOTO DIVLOOP;
68                  C = MBR; MBR = COM(MBR) + 1;
69      DIVLOOP:    A = LEFT_SHIFT(A + 0);
70                  IF BIT(15, D) = 0 THEN GOTO NOCARRY2;
71                  A = A +1;
72      NOCARRY2: D = LEFT_SHIFT(0 + D);
73                  A = A + MBR;
74                  IF BIT(15, A) = 1 THEN GOTO DIVNEG;
75                  D = D + 1;
76      DIVNEG:     IF BIT(15, A) = 0 THEN GOTO DIVPOS;
77                  A = A + C;
78      DIVPOS:     X = X + (-1);
79                  IF BIT(15, X) = 0 THEN GOTO DIVLOOP;
80                  IF BIT(15, B) = 0 THEN GOTO DIVEND;
81                  D = 1 + COM(D);
82      DIVEND:     MBR = 0 + D; MEMORY(MAR) = MBR;
83                  GOTO MAINLOOP;
84      RETURN:     MAR = SP; SP = SP + (-1); MBR = MEMORY(MAR);
85                  IR = MBR;
86                  GOTO JUMP;
```

Daugybos algoritmas.

Atliekama daugyba stulpeliu iš karto kaupiant dalinių sandaugų sumą. Pavyzdžiui:

```
      1 1 0 1
      1 0 0 1
      1 1 0 1
    0 0 0 0        } sumuojamos dalinės
  0 0 0 0              sandaugos
1 1 0 1
1 1 1 0 1 0 1
```

Registrai A ir D naudojami kaip 32 bitų sumatorius: 16 vyresnių bitų yra registre A ir 16 jaunesnių bitų yra registre D.



Dauginamasis yra registre C, o daugiklis yra registre MBR. Interpretatoriaus eilutės 40 – 46 formuoja pradines registrų reikšmes. Jomis yra suformuojama tokia pradinė situacija, kad antrasis operandas – daugiklis būtų >= 0 . Jeigu antrasis operandas buvo neigiamas, tai tada pakeičiami abiejų operandų ženklai, ir rezultate sandauga nepasikeičia. Ženklų keitimą reikia atlikti naudojant registrą MBR, nes registras C turi įėjimą tik iš MBR. Laikinam MBR reikšmės saugojimui panaudojamas registras D.

Tam, kad nereikėtų dalinių sandaugų vis pridėti prie skirtingų bitų, pridedant prie registro A, atliekamas pastūmimas į dešinę. Registre A prieš pastūmimą į dešinę yra patikrinama, ar nuliame registro bite yra 1, jeigu taip, tai jį reikia pernešti į registrą D. Tokį pernešimą atliekame pridėdami registre SIGN esančią reikšmę. Tada pastumiame į dešinę.

Registre A ženklo bito plėtimas irgi atliekamas pastūmimu į dešinę.Registre A tikriname 14-to bito reikšmę, nes nulis 15-tame bite gali būti ir dėl to, kad dar nieko neužrašėme į registrą A. Reikia patikrinti, ar buvo postūmis, t. y, ar 14-tame bite yra 1 – postūmio rezultate atėjęs ženklo bitas.

16 bitų perpildymas aparatūriškai nėra fiksuojamas.

Dalybos algoritmas.

Surandama skaičiaus dalis, kuri dalosi iš daliklio. Jeigu nesidalina, tai liekana papildoma ir tikrinama, ar jau dalosi. Jeigu ne, tai liekana plečiama ir dalmenyje prirašomas 0. Jeigu dalosi, tai dalmenyje prirašomas 1 ir veiksmas vėl kartojamas.

Pradžioje gaunamas absoliučios abiejų operandų reikšmės. Jeigu reikia, tai neigiama reikšmė rezultatui priskiriama jau padalinus.

32 bitų sumatorius suformuojamas iš registro A (kairioji pusė) ir registro D (dešinioji pusė). Tam, kad efektyviai atlikti atimtį, daliklio absoliučios reikšmės papildomas kodas saugomas registre MBR. Jeigu atėmus iš dalomojo gavosi neigiamas skaičius, tam, kad atstayti jo reikšmę (anuliuoti atimtį), daliklio absoliuti reikšmė saugoma registre C.

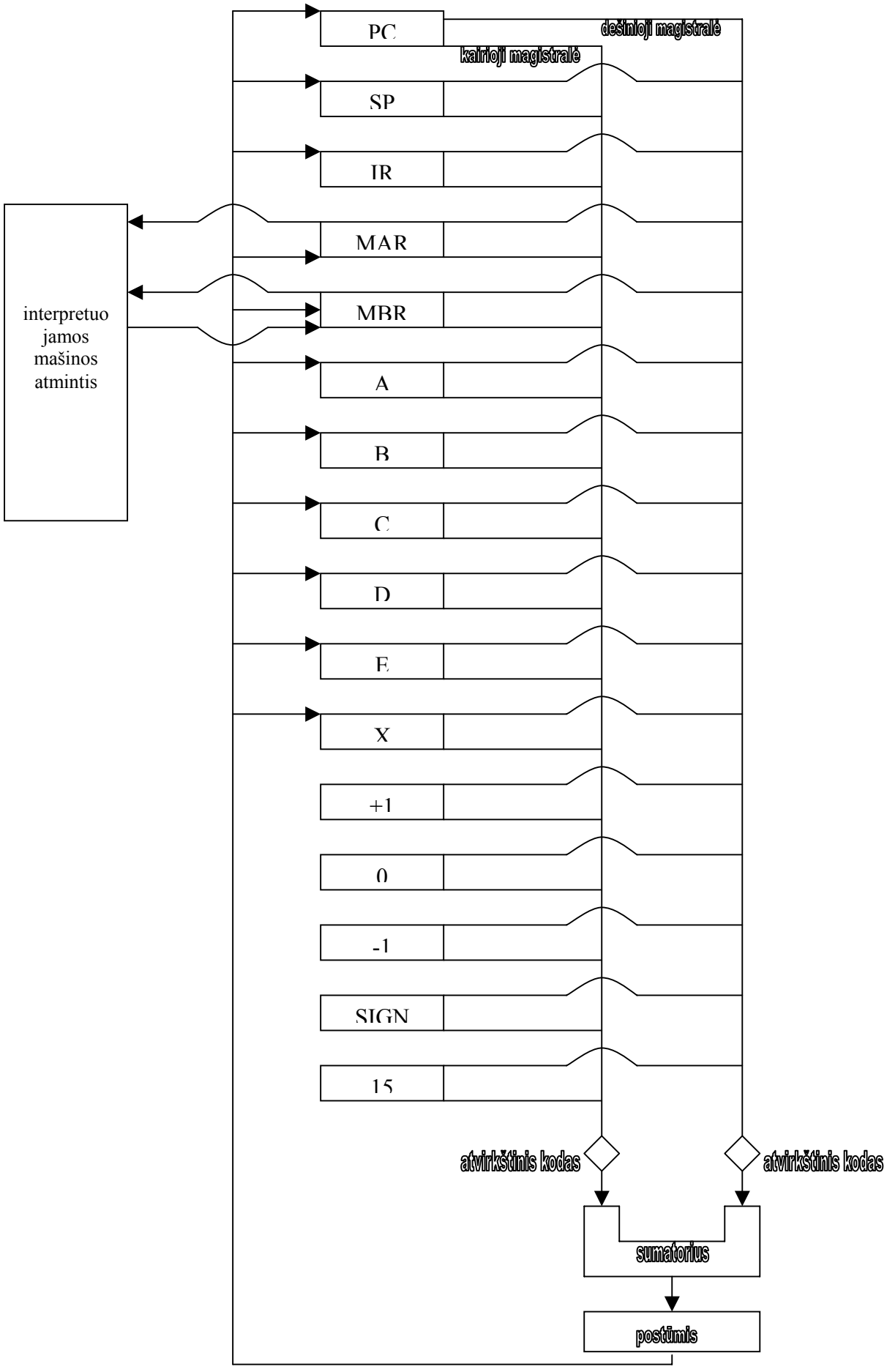
Jeigu pirmas operandas >= 0, tai registre B saugoma antro operando reikšmė. Jeigu pirmas operandas < 0, tai regsitre B saugomas antro operando papildomas kodas. Rezultato ženklas bus toks pat, kaip ir reikšmės, saugomos registre B.

Mikroprogramininio lygio projektavimas

Mikrokomandos, kurių kiekvienas bitas yra susietas su atskiru ventiliu, pasižymi struktūros ir realizacijos paprastumu. Tačiau kompiuterio ventilių skaičius gali siekti kelis šimtus. Mikrokomanda, užimanti kelis šimtus bitų, yra pernelyg ilga, užima perdaug mikroatminties.

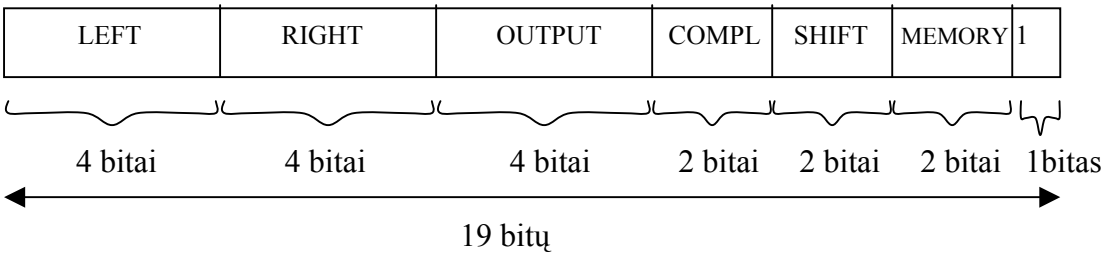
Pagal mikroproprogramininio lygio architektūrą, ne bet kokie ventiliai kuriuo tai laiko momentu gali būti atidaryti. Pavyzdžiui, į kairįjį sumatoriaus įėjimą gali būti paduotas tik vienas signalas iš dešimties galimų. Jiems koduoti pakanka 4 bitų. Į dešinįjį įėjimą gali būti paduotas tik vienas iš 7 galimų signalų. Jam koduoti pakanka 3 bitų. Praktikoje sumatoriaus išėjimas nukreipiamas tik vienu keliu. Todėl reikalingas tik vienas signalas iš 6 galimų. Jam koduoti pakanka 3 bitų. Todėl 23 ventilių darbas gali būti užkoduotas 10 bitų. Toks mikrokomandų formatas vadinamas koduojamu. Koduojamo formato trūkumas yra tas, kad kiekvienam koduojamam laukui reikalingas dešifratorius, kuris lauko reikšmę pervestų į ventilio numerį. Koduojant prarandamas tam tikras lankstumas užduodant lygiagrečius veiksmus. Į koduojamą lauką paprastai įtraukiami vienu metu nesuderinami ventiliai.

Galimas sprendimas yra modifikuoti informacinius kanalus, įvedant tris bendras magistrales ir naują registrą E.

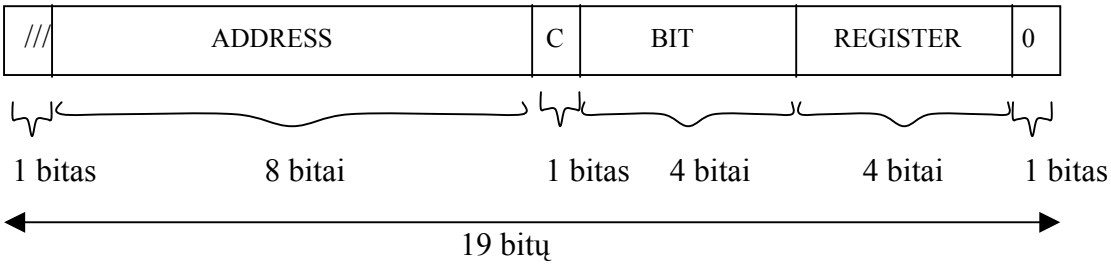

PC
SP
IR
MAR
MBR
A
B
C
D
E
X
+1
0
-1
SIGN
15
interpretuo jamos mašinos atmintis
dešinioji magistralė
kairioji magistralė
atvirkštinis kodas
atvirkštinis kodas
sumatorius
postūmis

Tokios architektūros mikroprograminio lygio komandos gali būti tokios:

GATE



TEST



REGISTER

0	=	$(0000)_2$	PC
1	=	$(0001)_2$	SP
2	=	$(0010)_2$	IR
3	=	$(0011)_2$	MAR
4	=	$(0100)_2$	MBR
5	=	$(0101)_2$	A
6	=	$(0110)_2$	B
7	=	$(0111)_2$	C
8	=	$(1000)_2$	D
9	=	$(1001)_2$	E
10	=	$(1010)_2$	X
11	=	$(1011)_2$	+1
12	=	$(1100)_2$	0
13	=	$(1101)_2$	-1
14	=	$(1110)_2$	SIGN
15	=	$(1111)_2$	15

COMPL

0	=	$(0000)_2$	abiems įėjimams atvirkštinio kodo neskaičiuoti
1	=	$(0001)_2$	skaičiuoti atvirkštinį kodą kairiajam operandui
2	=	$(0010)_2$	skaičiuoti atvirkštinį kodą dešiniajam operandui
3	=	$(0011)_2$	abiems oprerandams skaičiuoti atvirkštinį kodą.

SHIFT

0	=	$(0000)_2$	išėjimo nestumti
1	=	$(0001)_2$	stumti į kairę per 1 bitą
2	=	$(0010)_2$	stumti į dešinė per 1 bitą
3	=	$(0011)_2$	nenaudojamas

MEMORY

0	=	$(0000)_2$	nėra apsikeitimo su atmintimi
1	=	$(0001)_2$	skaityti iš atmintes
2	=	$(0010)_2$	rašyti į atmintį
3	=	$(0011)_2$	nenaudojamas

BIT – bito registre nurodymas

C

0	=	$(0000)_2$	perduoti valdymą, jei tikrinamas bitas yra 0
1	=	$(0001)_2$	perduoti valdymą, jei tikrinamas bitas yra 1

ADDRESS – mikrokomandos absoliutus adresas, kuriai perduodamas valdymas.

Architektūra, kurią nagrinėjome anksčiau, yra vadinama horizontalia. Joje kiekvienas bitas nurodo ventilį, taip gaunama ilga mikrokomanda. Architektūra, kurią nagrinėjame dabar, vadinama vertikalia. Joje naudojami koduojami laukai ir taip gaunama trumpa mikrokomanda.

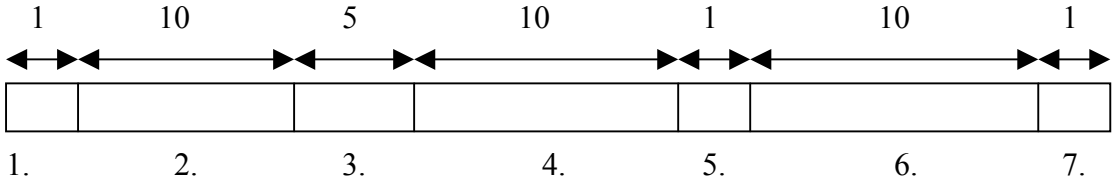
Horizontali struktūra užtikrina didesnį paralelizmą, yra efektyvesnė, reikia mažiau mikrokomandų, tačiau jos yra ilgesnės. Tačiau jeigu tokioje architektūroje yra N registrų, tai tarp jų reikia 2^N magistralių. Todėl jeigu registrų yra daug (N yra didelis), kompiuteris yra nehorizontalios architektūros. Kita vertus, lygiagretus perdavimas vienu metu paprastai vykdomas tik trijose magstralėse, ir didelis registrų skaičius čia neturi jokios įtakos.

Jeigu mašina yra vertikalios architektūros, tai išskirtinės magistralės (intensyviai naudojamos) gali būti realizuotos horizontaliu principu.

Atminties ciklų trukmė

Procesoriaus ciklas beveik pilnai apsprendžiamas mikroatminties ciklo dydžiu. Pagrindinės atminties ir mikroatminties ciklų dydžių santykis yra viena iš pagrindinių kompiuterio darbinių charakteristikų.Interpretuojamos mašinos atmintis yra lėtesnė už interpretuojančios mašinos atmintį. Vykdant interpretuojamą komandą, į pagrindinę atmintį paprastai reikia kreiptis vieną ar du kartus, o į mikroatmintį – dešimtis ar net šimtus kartų. Dažnai praktikoje tas santykis yra 1, nes abiems loginėms atmintims naudojama ta pati fizinė atmintis.

Mikroatmintį yra stengiamsi padaryti greitesne, netgi naudojant pastovią atmintį, prieinamą tik skaitymui. Tegul atminties ciklų santykis yra 10 : 1. Tarkime, kad atliekama komanda PUSH:



1. Interpretuojamos komandos skaitymo inicijavimas.
2. Interpretuojamos komandos nuskaitymo laukas.
3. Komandos dekodavimas ir operando skaitymo inicijavimas.
4. Operando skaitymo laukimas.
5. Rašymo inicijavimas.
6. Rašymo pabaigos laukimas.
7. Perėjimas į pagrindinį ciklą.

Čia 79% laiko procesorius laukia ir yra nenaudojamas. Tam, kad sumažinti laiką, kada procesorius nenaudojamas, reikia arba greitinti pagrindinę atmintį, arba išlygiagretinti (suderinti laike) interpretuojamos mašinos komandų valdymą.

Pavydžiui, vykdant dalybos komandą, reikia maždaug 150 mikrokomandų ir per tą laiką galima iš anksto iš atminties nusiskaityti 10 sekančių interpretuojamų komandų. Nuskaitytas komandas reikia kur nors padėti saugojimui. Galima padėti į papildomus registrus, bet tai apsunkina architektūrą, nes reikia papildomų magistralių ir mikrokomandos tampa ilgesnės. Tačiau nuskaitytas komandas galima padėti ir į mikroatmintį. Vien tik pasiruošti komandas iš anksto nepakanka, nes tada pagrindinė laukimo priežastis bus duomenys.

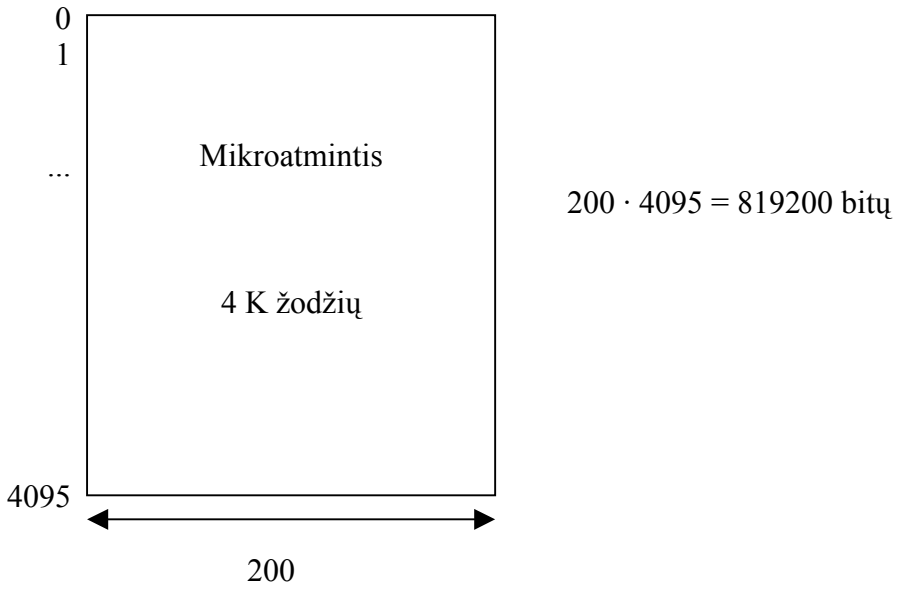
Turint stekinę mašiną nesudėtinga iš steko viršūnės nuskaityti duomenis iš anksto, kad nebūtų laukimo dėl duomenų skaitymo.

Kitas sprendimas yra sudaryti tokias mikroprogramas, kad būtų galima užtikrinti lygiagretų komandų interpretavimą. Pradėti vykdyti sekančią komandą, nebaigus vykdyti ankstesnės, yra gana keblus uždavinys ir yra prasmingas tik tada, kai tokio uždavinio sprendimo greitis yra sulyginamas su skaitymo iš atminties greičiu. Pavyzdžiui, nėra prasminga lygiagretinti tokia mikroprogramą:
$L = I + Y \cdot K$
PUSH I;
PUSH Y;
PUSH K;
MUL;
ADD;
POP L;

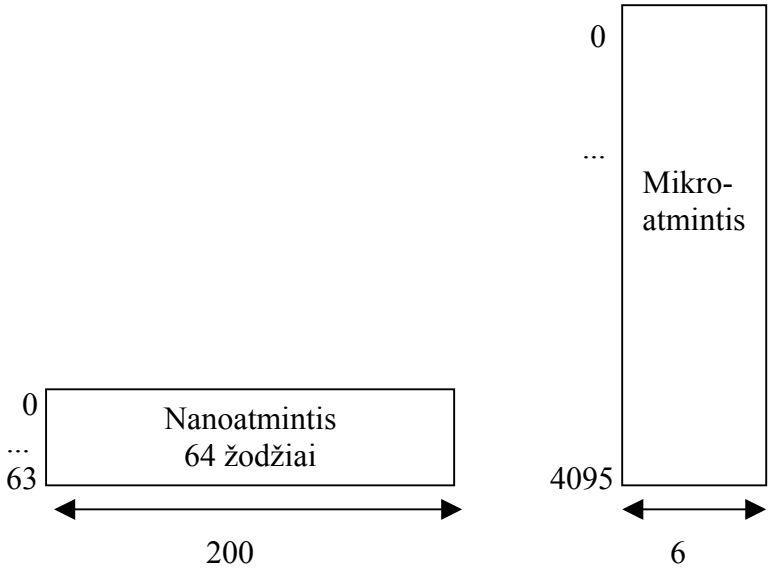
Lygiagretinti reikia turint sąlyginį perėjimą. Jo atveju atsiranda dvi šakos. Dar nebaigus skaičiuoti sąlygos reikšmės ir nežinant, kurią šaką iš tikrųjų reikės vykdyti, galima pradėti atlikti veiksmus, nurodytus kuria nors iš jų.
IF $A \cdot B \cdot C \cdot D \cdot E < 1000$ THEN CALL SUB;
Jeigu pasirinkta šaka nepasitvirtina, reikia organizuoti grįžimą, nes buvo ivykdyti nereikalingi priskyrimai. Statistiškai sąlyga paprastai nepatenkinama, o patenkinama tik vieną kartą, nes sąlyga paprastai įeina į ciklą.

Nanoatmintis

Horizontalioje architektūroje paprastai naudojamas nedidelis ventilių skirtingų kombiancijų kiekis, tačiau tos kombinacijos, kurios naudojamos, yra naudojamos daugelį kartų (pvz., skaitymas iš steko). Tos skirtingos kombinacijos užduodamos mikrokomandomis (jų ilgis gali būti keli šimtai bitų, o jų skaičius 64 – 256), kurios saugomos nanoatmintyje. Tuomet mikroprograma yra tokių nanoatminties mikrokomandų adresų seka.



Jeigu mikroprogramoje yra ne daugiau 64 skirtingų žodžių, tada architektūra galėtų būti tokia, kad mikroatmintyje yra adresai komandų, kurios tais adresais yra saugomos nanoatmintyje:



64 · 200 + 4095 · 6 = 12800 + 24576 = 37376 bitų

Taip susidaro 5% buvusio atminties dydžio.
Čia išsaugomos horizontalios struktūros perdavimų lygiagretumas ir vertikalios struktūros atminties taupymas. Padidėja procesoriaus ciklas, nes jis susideda iš mikroatminties ir nanoatminties ciklų

Turint tikslą sumažinti skirtingų mikrokomandų skaičių, labai svarbu turėti valdymo perdavimą santykiniais adresais.
IF BIT(15, MBR) = 1 THEN SKIP n;
vietoj
IF BIT(15, MBR) = 1 THEN GOTO L;

Mikroprogramines lygis gali būti orientuotas konkretaus interpretuojamo lygio efektyviam interpretavimui arba gali būti universalus mikroprogramines lygis, skirtas įvairių architektūrų interpretavimui.

Pagal nusistovėjusias tradicijas kalbų transliatoriai (ALGOL, FORTRAN,DL11, COBOL, PASCAL, C) perveda programą į tradicinį asemblerinį lygį, kuris vėliau interpretuojamas. Maža yra pagrindo manyti, kad tradicinis asembleris yra optimali tarpinė kalba tarp kompiliatoriaus ir interpretatoriaus. Kompiliatoriai naudoja apie 20% komandų sugeneruotame kode.

Mikroprogramavimo trūkumas yra tas, kad efektyvumas yra mažesnis apie 10 kartų, o norint gauti duoto galingumo procesorių, naudojant mikroprogramavimą, jis turi būti 10 kartų galingesnis, lyginant su aparatūrine komandų sistemos realizacija.