KTU KOMPIUTERIŲ KATEDRA

Kompiuterių architektūros pirmas laboratorinis darbas

Kazimieras Bagdonas
Stasys Maciulevičius

Turinys

Pratarmė

Šis laboratorinis darbas yra skirtas supažindinti studentus su konceptais reikalingais suprasti žemiausio lygio skaitmeninių įrenginių programavimo pagrindus. Laboratorinis darbas tęsia ir plėtoja Skaitmeninės logikos pradmenyse [$P175B100$] įgytas žinias ir įgūdžius. Laboratorinis darbas gali būti atliekamas bet kokiu VHDL simuliatoriumi, tačiau buvo pritaikytas darbui su Lattice Diamond Active HDL paketu.

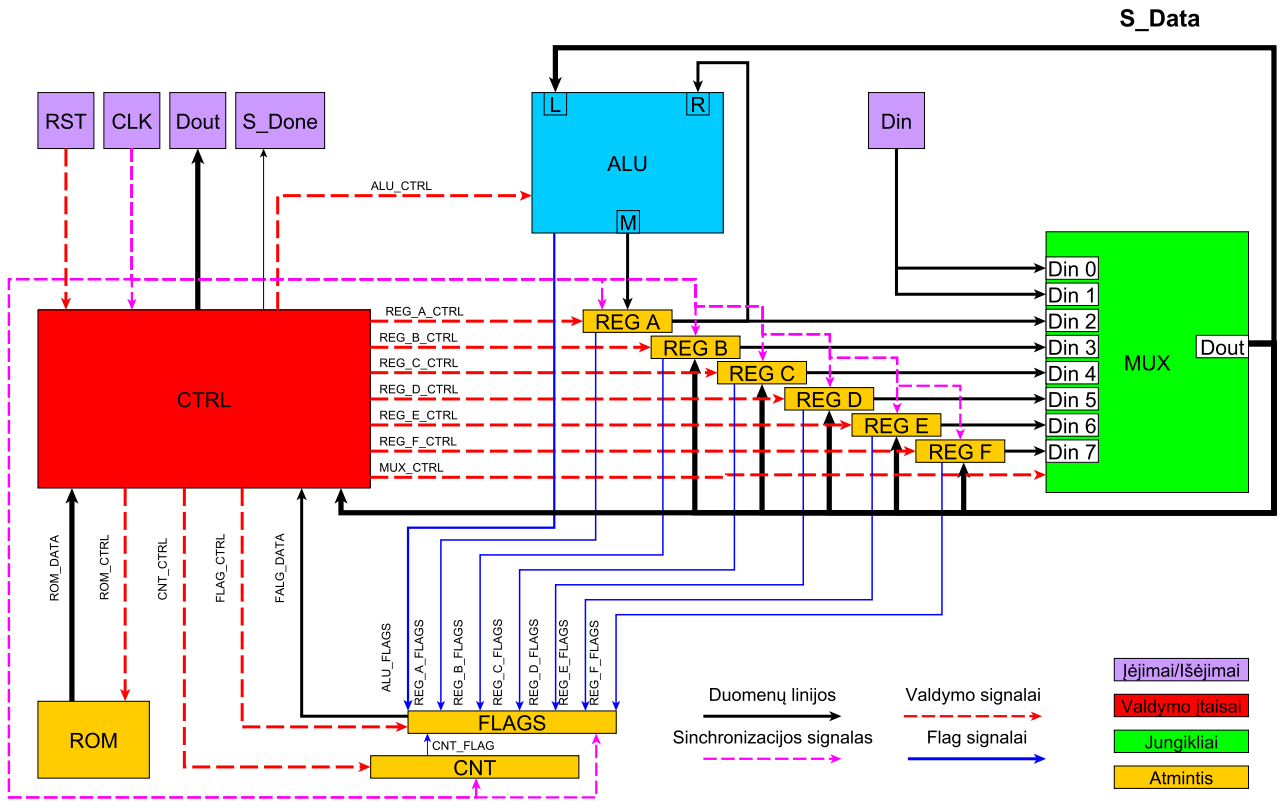
Pagrindiniai laboratorinio darbo fokuso taškai yra:

- Skaitmeninę informaciją apdorojančių įrenginių struktūra
- Informacijos perdavimo tarp įrenginių būdai
- Skaitmeninės informacijos apdorojimo algoritmų kūrimas ir realizacija
- Algoritmų adresavimo tipai

1 skyrius

Procesorius

1.1. Struktūra



1.1 pav. Procesoriaus struktūrinė schema

Paveikslėlyje 1.1 yra pateikiama struktūrinė procesoriaus schema, kurioje išdėstyti komponentai ir parodyti jų sąryšiai kontrolės ir duomenų kanalais. Procesorius turi du įėjimus: sinchro signalą **CLK** ir 16 bitų duomenų įvestį **Din**. **Dout** yra 16 bitų duomenų išvesties kanalas, o **S_Done** – programos pabaigos signalas išoriniams įrenginiams.

Procesorių sudaro šie komponentai:

1. Valdiklis – **CMD** 1.1.1
2. Aritmetinis loginis įrenginys – **ALU** 1.1.2
3. Pastovioji atmintis – **ROM** 1.1.3
4. 6 bendros paskirties registrai – **REG A—F** 1.1.4
5. Skirstytuvas – **MUX** 1.1.5
6. Skaitiklis – **CNT** 1.1.6
7. Įvykių registras – **FLAG** 1.1.7

1.1.1 Valdiklis

Valdiklis, paveikslėlyje 1.1 žymimas **CTRL** simboliu organizuoja procesoriaus darbą, pagal vartotojo sukurtą programą, esančią pastoviojoje atmintyje **ROM**. Kiekvieno takto pradžioje valdiklis nuskaito vieną Mikro Komandą (MK) iš ROM ir ją įvykdo. MK struktūra priklauso nuo naudojamos adresavimo sistemos 3. Priverstinėje adresacijoje MK yra sudaryta iš Operacinių Komandų (OP), Loginės Sąlygos numerio (LS) ir Naujo Adreso (NA) blokų 3.1. Natūralioje adresacijoje egzistuoja dviejų tipų MK. Pirmoji sudaryta komandos tipo lauko $P = 0$ ir OP 3.2 ir yra skirta atlikti užprogramuotus veiksmus, antroji – iš komandos tipo lauko $P = 1$, LS ir NA 3.3, skirta įvertinti logines sąlygas ir nustatyti sekančią instrukciją. Lentelėje 1.1 yra pateiktos ir aprašytos visos valdiklio įvestys ir išvestys.

1.1.2 Aritmetinis loginis įrenginys

Aritmetinis Loginis Įrenginys (ALU) 1.1 yra kombinacinė loginė schema, turinti du duomenų įėjimus: kairįjį (L) ir dešinįjį (R), bei rezultato išėjimą (M) 1.2. Pilnas ALU įėjimų ir išėjimų sąrašas pateikiamas lentelėje 1.2.

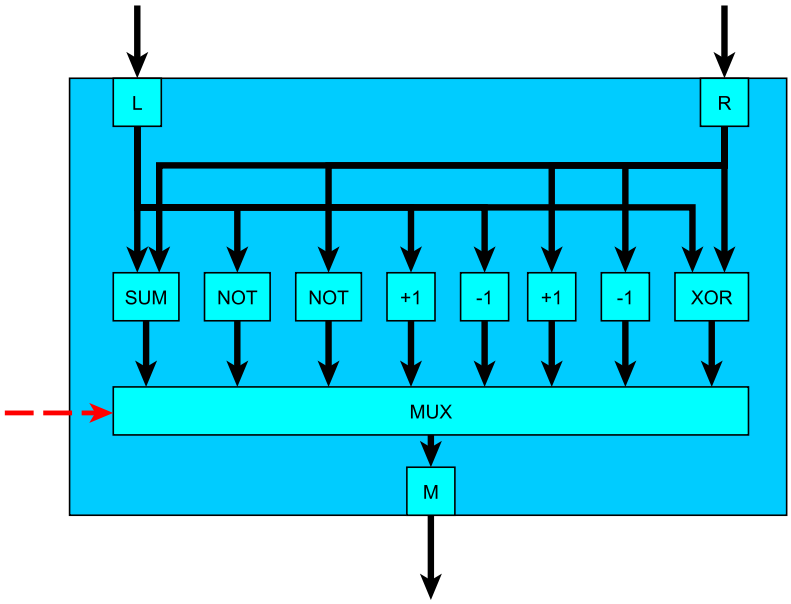
1.1 lentelė. Valdiklio įėjimai ir išėjimai

Įvestys	**Komanda**
CLK	5 Sinchronizavimo signalas
RST	16 Reset signalas
ROM_Din	MK komandos įvestis iš ROM komponento
Din	5 16 bitų duomenų įvestis
$FlagV$	5 Įvykių registro vektoriaus įvestis
Išvestys	**Komanda**
CNT_CMD	Skaitiklio valdymo signalas
MUX_CMD	3 bitų MUX valdymo signalas
ALU_CMD	5 ALU valdymo signalas
CMD	16 bitų duomenų išvestis
REG_A_CMD	3 A registro valdymo signalas
REG_B_CMD	3 B registro valdymo signalas
REG_C_CMD	3 C registro valdymo signalas
REG_D_CMD	3 D registro valdymo signalas
REG_E_CMD	3 E registro valdymo signalas
REG_F_CMD	3 F registro valdymo signalas
RST_COMP	9 reset signalai į individualius komponentus
$Done$	programos pabaigos signalas
$CTRL\_Dout$	16 bitų rezultato išvedimas

1.2 lentelė. ALU įėjimai ir išėjimai

Įvestys	**Komanda**
ALU_CMD	5 bitų baitmetinės komanda
ALU_Din_L	16 bitų kairioji duomenų įvestis
ALU_Din_R	5 bitų dešinioji duomenų įvestis
Išvestys	**Komanda**
FLG_ALU_CMD	5 bitų vektorius išvedamas į įvykių registą
ALU_Dout	16 bitų duomenų išvestis

ALU galima vizualizuoti kaip aštuonių atskirų kombinaciunių schemų rinkinį, prijungtą prie vidinio skirstytuvo (MUX) 1.2. Priklausomai nuo į MUX'ą ateinančių valdančiųjų signalų, vienos iš imties schemų rezultatas yra išvedamas iš ALU.



1.2 pav. ALU vidinė struktūra

ALU atliekamos operacijos yra apibrėžtos lentelėje 1.3. Vieno ciklo metu ALU gali atlikti tik viena aritmetinę operaciją. Jei vienoje MK bus pažymėta daugiau nei viena ALU operacija, ALU įvykdys tą, kurios kodas yra didžiausias.

Pavyzdyje realizuojamas ALU gali priimti penkių bitų instrukciją. Lentelėje 1.3 yra naudojami tik paskutiniai trys bitai, leidžiantys turėti 8 individualias operacijas. Naudojant visus 5 bitus, ALU galėtų atlikti 32 skirtingas operacijas. Ši išplėtimo galimybė yra skirta studentams, kurie norėtų papildyti ALU operacijų imtį pagal individualius poreikius.

1.1.3 Pastovioji atmintis

Pastovioji atmintis (ROM) 1.1, yra sistemos elementas, kuriame patalpinta vykdomoji programa. ROM turi **RST_ROM**, **ROM_CMD** įėjimus ir **ROM_DATA** duomenų išėjimą. Aukšto lygio **RST_ROM** signalas nustato ROM elementą į pradinę būseną.

1.3 lentelė. ALU atliekamos aritmetinės operacijos

ALU_CMD	Komanda	Veiksmas
000	$M = L + R$	Suma
001	$M = \overline{L}$	Invertuojamas L
010	$M = \overline{R}$	Invertuojamas R
011	$M = L + 1$	L padidinamas vienetu
100	$M = L - 1$	L sumažinamas vienetu
101	$M = R + 1$	R padidinamas vienetu
110	$M = R - 1$	R sumažinamas vienetu
111	$M = L \oplus R$	L xor R

ROM_CMD priima dvejetainio kodo komandas, kurios nustato kelinta atminties eilutė bus išvesta į **ROM_DATA** išėjimą.

ROM matricos plotis priklauso nuo MO skaičiaus ir naudojamos adresavimo sistemos. Jei naudojama Priverstinė 3.0.1 adresavimo sistema, matricos plotis yra lygus MO, LS ir NA koduojančių bitų skaičiui. Natūralios 3.0.1 adresavimo sistemos atveju ROM matricos plotis yra lygus MO skaičiui +1.

ROM matricos ilgis priklauso nuo MK skaičiaus. ROM eilutės numeris sutampa su MK adreso numeriu.

1.1.4 Registrai

Pateiktame procesoriuje yra integruoti 6 universalšūs 16 bitų registrai 1.1, indentifikuojami raidėmis **A**, **B**, **C**, **D**, **E**, **F**. Registrų įėjimai ir išėjimai yra apibrėžti lentelėje 1.4.

Registras **A** atlieka akumuliatoriaus funkciją. Į jį duomenys gali būti įrašomi tik iš ALU išėjimo. Registro **A** išėjimas yra tiesiogiai sujungtas su dešiniuoju (**L**) ALU įėjimu ir MUX 2 įėjimu. Registrų **B**—**F** duomenų įėjimai yra prijungti prie pagrindinės duomenų magistralės, o išėjimai atitinkamai prie MUX 3—7 įėjimų.

Kiekvienas registras gali atlikti 8 operacijas, jos ir jas inicijuojančios kontrolės signalų kombinacijos yra pateiktos lentelėje 1.5.

1.1.5 Skirstytuvas

Skirstytuvas (MUX) yra įrenginys, turintis 8 duomenų ir 1 valdymo įėjimus, bei 1 duomenų išvestį. Priklausomai nuo valdymo signalo kombinacijos, vienas iš įėjimų yra perduodamas į išėjimą. Visi MUX įėjimai ir išėjimai yra papibrėžti lentelėje 1.6. Valdymo

1.4 lentelė. Registrų įėjimai ir išėjimai

Įvestys	**Komanda**
CLK	Sinchronizavimo signalas
RST	Reset signalas
REG_Din	16 bitų duomenų įvestis
REG_CMD	3 bitų komanda
Išvestys	**Komanda**
REG_FLAG_H	15o bito reikšmė išvedama į įvykių registą
REG_FLAG_L	0o bito reikšmė išvedama į įvykių registą
REG_Dout	16 bitų duomenų išvestis

1.5 lentelė. Registrų atliekamos operacijos

REG_CMD	**Komanda**	**Veiksmas**
000	$A = A$	Saugojimas
001	$A = Din$	Duomenų įvedimas
010	$A = LL1(A)$	Loginis postūmis į kairę per 1 bitą
011	$A = LR1(A)$	Loginis postūmis į dešinę per 1 bitą
100	$A = AL1(A)$	Aritmetinis postūmis į kairę per 1 bitą
101	$A = AR1(A)$	Aritmetinis postūmis į dešinę per 1 bitą
110	$A = CL1(A)$	Ciklinis postūmis į kairę per 1 bitą
111	$A = CR1(A)$	Ciklinis postūmis į dešinę per 1 bitą

1.6 lentelė. Skirstytuvo įėjimai ir išėjimai

Įvestys	Komanda
MUX_CMD	Sinchronizavimo signalas
MUX_Din0	16 bitų duomenų įvestis
MUX_Din1	16 bitų duomenų įvestis
MUX_Din2	16 bitų duomenų įvestis
MUX_Din3	16 bitų duomenų įvestis
MUX_Din4	16 bitų duomenų įvestis
MUX_Din5	16 bitų duomenų įvestis
MUX_Din6	16 bitų duomenų įvestis
MUX_Din7	16 bitų duomenų įvestis
Išvestys	**Komanda**
MUX_Dout	16 bitų duomenų išvestis

1.7 lentelė. Skirstytuvo atliekamos operacijos

MUX_CMD	Komanda	Veiksmas
000	$Dout = Din00$	$S\_Data = Din$
001	$Dout = Din01$	$S\_Data = Din$
010	$Dout = Din02$	$S\_Data = REG\_A\_Dout$
011	$Dout = Din03$	$S\_Data = REG\_B\_Dout$
100	$Dout = Din04$	$S\_Data = REG\_C\_Dout$
101	$Dout = Din05$	$S\_Data = REG\_D\_Dout$
110	$Dout = Din06$	$S\_Data = REG\_E\_Dout$
111	$Dout = Din07$	$S\_Data = REG\_F\_Dout$

signalų kombinacijos yra pateiktos lentelėje 1.7

1.1.6 Skaitiklis

Skaitiklis (**CNT**)yra 16 bitų registras, kuris gavęs kontrolės signalą, sumažina savo vidinę reikšmę vienetu. Skaitiklio įėjimai ir išėjimai yra apibrėžti lentelėje 1.8. Skaitiklis yra naudojamas sukurti programoje ciklus. Kai skaitiklio vidinė reikšmė pasiekia kritinę reikšmę, skaitiklis išduoda įvykio signalą į 13 įvykių registro bitą. Pradinė ir kritinės reikšmės yra nurodomos skaitiklio programine kode.

1.8 lentelė. Skaitiklio įėjimai ir išėjimai

Įvestys	**Komanda**
CLK	Sinchronizavimo signalas
RST	Reset signalas
Išvestys	**Komanda**
CNT_Flag	Įvykio signalas

1.9 lentelė. Įvykių registro bitų reikšmės

RST_COMP bitas	**Komponentas**
$FLAG(1)$	REG_A_H
$FLAG(2)$	REG_A_L
$FLAG(3)$	REG_B_H
$FLAG(4)$	REG_B_L
$FLAG(5)$	REG_C_H
$FLAG(6)$	REG_C_L
$FLAG(7)$	REG_D_H
$FLAG(8)$	REG_D_L
$FLAG(9)$	REG_E_H
$FLAG(10)$	REG_E_L
$FLAG(11)$	REG_F_H
$FLAG(12)$	REG_F_L
$FLAG(13)$	CNT
$FLAG(14)$	ALU pernaša
$FLAG(15)$	$1 = 0$
$FLAG(16)$	$1 = 0$
$FLAG(17)$	$1 = 0$
$FLAG(18)$	$1 = 0$

1.10 lentelė. Įvykių registro įėjimai ir išėjimai

Įvestys	**Komanda**
CLK	Sinchronizavimo signalas
RST	Reset signalas
Xin	18 bitų įvestis
Išvestys	**Komanda**
FLG_Dout	18 bitų išvestis

1.1.7 Įvykių registras

Įvykių registras (**FLAG**) yra komponentas fiksuojantis kritinius įvykius ir suteikiantis galimybę valdikliui lengvai prieiti prie šių duomenų, kai yra vertinamos loginės sąlygos. Įvykių registro bitų reikšmės yra pateiktos lentelėje 1.9. $FLAG(1)$ bitas parodo, jog A registre, 15 (kairiausiame – REG_A_H) bite yra 1, kas ženklina, jog registre yra įrašytas neigiamas skaičius ir yra naudingas, jei taikomas algoritmas tikrina skaičiaus aukščiausią skiltį. $FLAG(1)$ bitas parodo, jog A registre, 0 (dešiniausiame – REG_A_L) bite yra 1. Šis bitas yra naudingas, jei naudojamas algoritmas tikrina žemiausią skaičiaus skiltį.

Įvykių registro įėjimo ir išėjimo signalai apibrėžti lentelėje 1.10

1.2. Valdymo signalai

Laboratoriniame darbe naudojamas procesorius yra pagrįstas moduliniu dizainu ir apdoroja duomenis tarp komponentų naudodamas kontroliniais signalais. Valdiklis **CTRL** priima išorinius **RST** ir **CLK** valdymo signalus, ir interpretuodamas **ROM** esančias instrukcijas, generuoja vidinius kontrolės signalus likusiems komponentams.

1.2.1 Reset signalai

Išorinis **RST** signalas nustato valdiklį į pradinę būseną, kurioje visi jo vidiniai kintamieji yra nunulinami. Šioje būsenoje valdiklis išduoda RST signalus likusiems komponentams. Priklausomai nuo programos esančios ROM, likę komponentai gali būti resetinami programuotojui patogiu metu pasinaudojant RST_COMP signalais apibrėžtais lentelėje 1.11.

1.11 lentelė. Individualūs komponentų reset signalai

RST_COMP bitas	**Komponentas**
$RST\_COMP(0)$	REG_A
$RST\_COMP(1)$	REG_B
$RST\_COMP(2)$	REG_C
$RST\_COMP(3)$	REG_D
$RST\_COMP(4)$	REG_E
$RST\_COMP(5)$	REG_F
$RST\_COMP(6)$	CNT
$RST\_COMP(7)$	ROM
$RST\_COMP(8)$	FLG

1.2.2 Sinchronizacijos signalas

Sinchronizacijos signalas yra naudojamas siekiant padidinti sistemos apibrėžtumo laipsnį. Komponentai, turintys savyje atminties elementus, atlieka savo operacijas lygiagrečiai, reaguodami į kylantį sinchronizacijos signalo frontą. ALU ir MUX nenaudoja sinchronizacijos signalo, nes yra sudaryti iš kombinacinės logikos ir keičia savo išėjimus, kai sulaukia kontrolės ar duomenų signalų pasikeitimo.

Dėl sinchronizacijos signalo ir natūralaus elementų vėlinimo nėra įmanoma vieno takto metu modifikuoti vieno komponento išėjimo ir panaudoti šiuos naujus duomenis kitame komponente. Šis apribojimas verčia duomenis paruošti naudojimui bent vienu sinchronizacijos signalo taktu anksčiau, nei kad juos reikės apdoroti kitam komponentui.

1.2.3 Valdiklio išduodami kontrolės signalai

Kontrolės signalai valdiklyje yra generuojami atsižvelgiant į programos kodą, patalpintą ROM komponente. Gavęs einamąją programos MK, valdiklis ją pabičiui interpretuoja ir radęs 1, sugeneruoja to bito pozicijai atitinkamą kontrolės signalą. Reikia atkreipti dėmesį, jog MK struktūra yra pagrįsta rastro (*bitmap*) principu, kuri leidžia viename takte lygiagrečiai perduoti kontrolės signalus nors ir visiems komponentams. Tačiau komponentams siunčiami valdymo signalai yra užkoduoti dvejetainiu kodu, taip sutaupant siunčiamos informacijos kiekį, tačiau apribojant galimybę komponentams atlikti daugiau nei vieną operaciją vieno takto metu.

1.2.4 Įvykių signalai

Įvykių signalai yra signalų grupė, kuriuos automatiškai generuoja periferiniai komponentai (registrai, ALU, skaitiklis). Šie signalai nustatinėja įvykių registro reikšmes. Kiekvienas įvykių signalas yra priskirtas individualiam įvykių registro bitui 1.9. Programos vykdymo eigoje, atėjus loginės sąlygos įvertinimui, valdiklis kreipiasi reikiamą į įvykių registro bitą ir nustato ar konkretus įvykis yra užfiksuotas.

1.3. Procesoriaus programavimas

Norint sėkmingai užprogramuoti procesorių, pirmiausiai reikia susidaryti algoritmą, kurio pagalba bus galima parašyti programą iš atskirų mikro komandų. Atsižvelgiant į asmeninę užduotį, reikia sudaryti abstraktų algoritmą 1.3, pažingsniui vėliau konkretizuojant atskirus mazgus, pasiekiamas lygis, leidžiantis užprogramuoti procesorių ir atskiri medžiai apjungiami į vientisą programą.

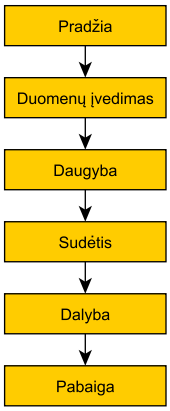
1.3.1 Algoritmo kūrimo pavyzdys

Norint pradėti formuoti algoritmą, pirmiausiai reikia atsižvelgti į užduoties formulę. Kaip pavyzdį naudokime formulę 1.1, informaciją gaunant tiesioginiu(T) kodu, ir operandų ženklams būnant $+/-/+$.

$$\frac{N1 + N2}{N3^2} \tag{1.1}$$

Bendras veiksmų planas galėtų būti atvaizduotas kaip paveikslėlyje 1.3, skaidant veiksmus į duomenų įvedimą ir paruošimą, dviejų operandų daugybą, sudėtį ir dalybą. Kad būtų patogiau, kiekvienas iš šių blokų gali būti atvaizduojamas kaip individualus algoritmas, kurie vėliau yra apjungiami į vientisą medį.

Pirmieji žingsniai programoje yra skirti duomenų įvedimui ir apdorojimui. Kadangi procesorius 1.1 turi tik vieną duomenų įvestį, operandus tenka įvedinėti nuosekliai. Kadangi procesoriaus A registras duomenis gali įrašyti tik iš ALU, pirmąjį operandą $N1$ įrašysime į B registrą, $N2$ į registrą C, o $N3$ į registrą D. Šiuos veiksmus galime užrašyti psiaudokodu



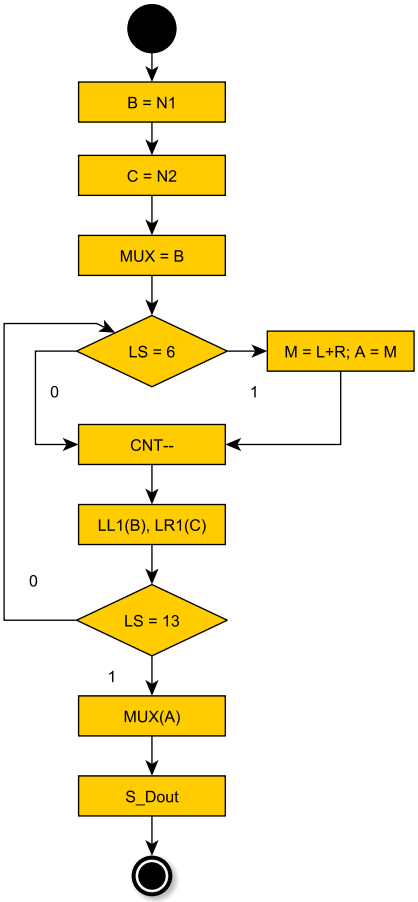
1.3 pav. Abstraktaus programos algoritmo pavyzdys

1.12 lentelė. Individualūs komponentų reset signalai

Veiksmas	**Psiaudokodas**
Išvesti Din reikšmę per MUX į S_Data magistralę	$S\_Data <= Din$
Įrašyti Din reikšmę į B registrą	$Reg\_B <= S\_Data$
Įrašyti Din reikšmę į C registrą	$Reg\_C <= S\_Data$
Įrašyti Din reikšmę į D registrą	$Reg\_D <= S\_Data$

1.3.2 Daugyba

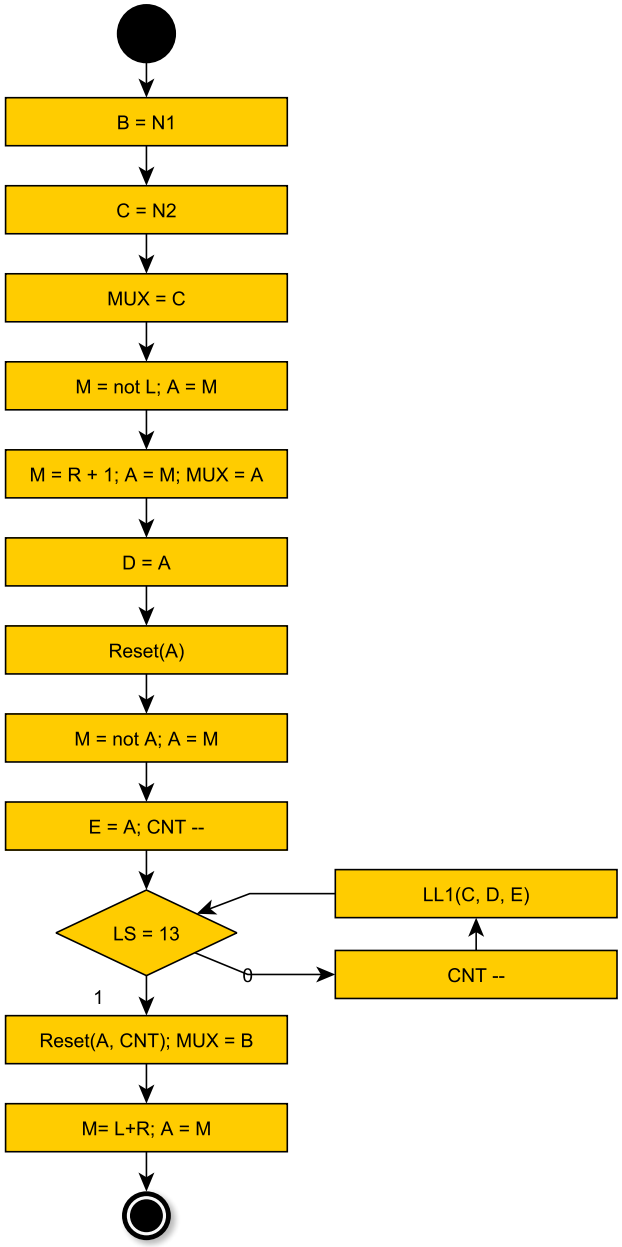
Pav. 1.4 yra pateiktas teigiamų skaičių daugymos algoritmo medis. Daugybos algoritmas pavaizduotas pav. 1.4 atitinka knygoje "Operacijos su sveikaisiais skaičiais ir jų modeliavimas" 24 p. aprašytą 2.2 Daugybos algoritmas" algoritmą. Rekomenduojame pasirinkti jums tinkamiausią algoritmą iš aprašytų knygoje.



1.4 pav. Teigiamų skaičių daugybos algoritmas

1.3.3 Dalyba

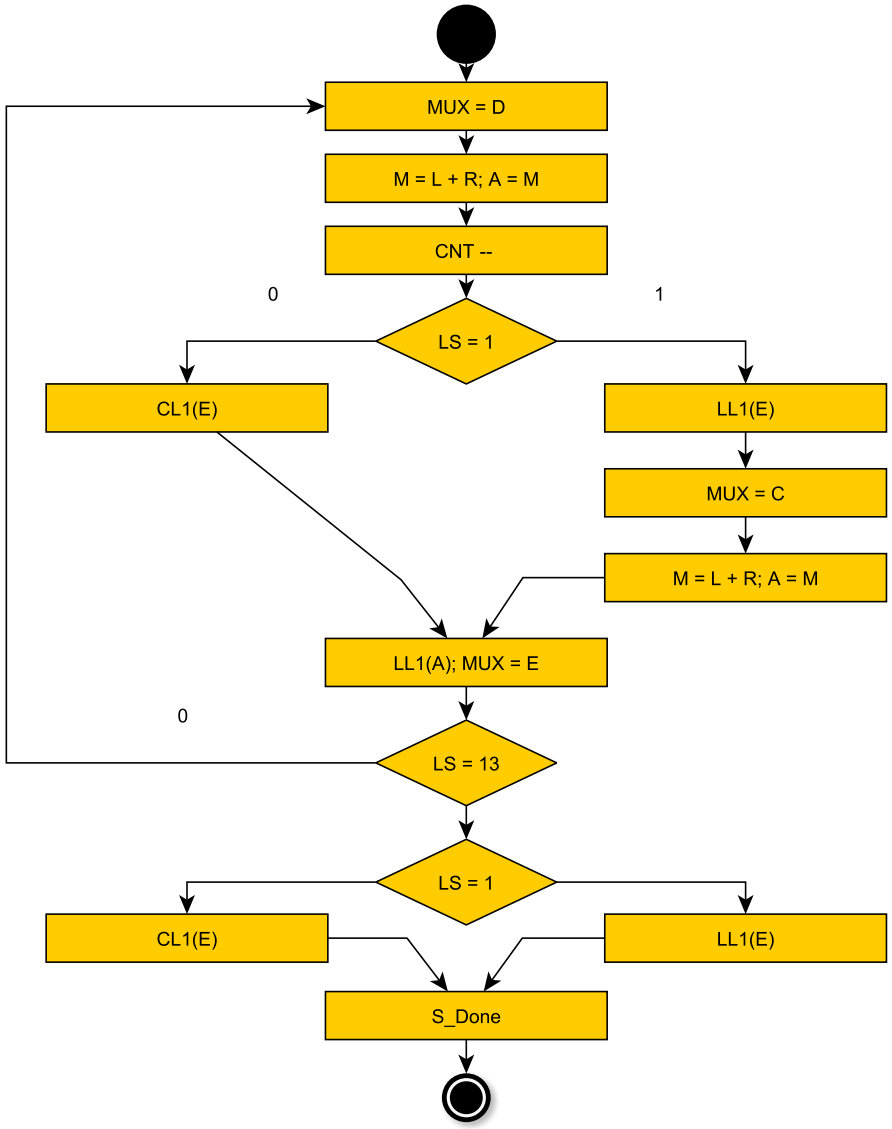
Pav. 1.5 yra pateiktas pasiruošimas pasiruošimo dalybai algoritmo medis. Dalybos algoritmas pavaizduotas pav. 1.6 atitinka knygoje "Operacijos su sveikaisiais skaičiais ir jų modeliavimas" 50 p. aprašytą 4.1.1 "Dalyba grąžinant liekaną ir stumiant liekanas" algoritmą.



1.5 pav. Pasiruošimas dalybai

- A registre bus patalpintas skaitiklis
- C registre saugoma teigiama vardiklio reikšmė
- D registre saugoma neigiama vardiklio reikšmė
- E registre formuojamas atsakymas

Prieš atliekant dalybos algoritmą C, D ir E registrai yra paslenkami 7 kartus į kairę (atkreipti dėmesį, jog CNT– vieną kartą yra panaudojama prieš slinkimo ciklą.)



1.6 pav. Teigiamų skaičių dalyba

Trečioji loginė sąlyga pav. 1.6 yra reikalinga, nes 8 taktų neužtenka atsakymui suformuoti. Kai skaitiklis pasiekia kritinę ribą, dar kartą reikia atlikti vieną postūmį atsakymo registre, atitinkamai: $CL1$ – norint įstumti 1 ir $LL1$ – norint įstumti 0.

2 skyrius

Adresacija

2.0.1 Priverstinė adresacija [F]

Panagrinėsime MK adresacijos būdą, vadinamą priverstine adresacija su vienu adreso lauku (F). Tokiu atveju paskesnės MK adresas nustatomas pagal formulę 3.1, o MK turi tokį formatą 3.1

$$A_{t+1} = A + xLS \tag{2.1}$$

Mikro komandos struktūra

2.1 lentelė. Priverstinės adresacijos MK struktūra

Y_1	Y_2	Y_3	...	Y_k	LS	A

Čia $Y1$, $Y2$, ..., Yk – laukai mikro operacijoms nurodyti, LS – loginės sąlygos numerio laukas, A – adreso laukas, LS_X – tikrinamosios sąlygos reikšmė. Sutarsime, kad besąlyginio perėjimo atveju $LS = 0$, o $A_{t+1} = A$.

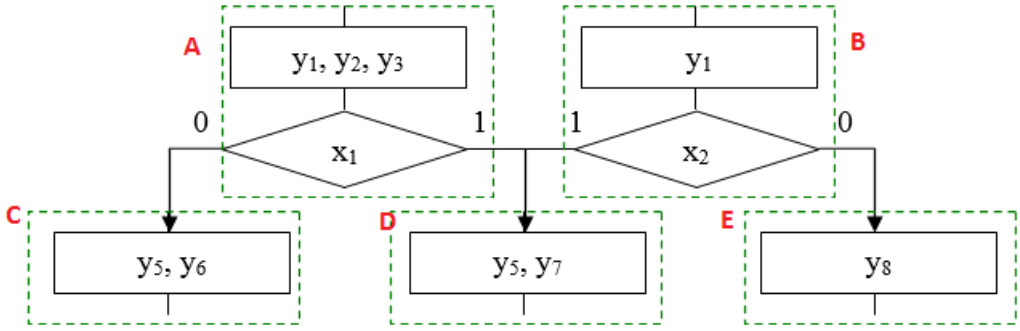
Priverstinės adresacijos atveju mikrokomandą gali atitikti:

1. viena operatorinė viršūnė, kurioje nurodytas vykdomų mikrooperacijų sąrašas:
2. operatorinė viršūnė, kurioje nurodytas vykdomų mikrooperacijų sąrašas, ir po jos esanti sąlygos tikrinimo viršūnė:

3. tik sąlygos tikrinimo viršūnė:

Galimos probleminės situacijos

Tarkime, turime mikroprogramos grafą, kuris parodytas žemiau. Priverstinės adresacijos atveju MK adresai gali būti priskirti kaip pavaizduota paveiksliuke3.1



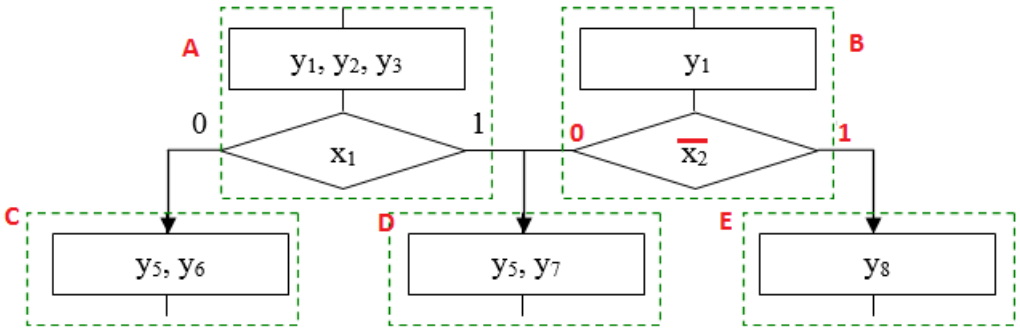
2.1 pav. Galimas adresų konfliktas priverstinės adresacijos algoritmo medyje

Šiame paveiksle MK išskirtos rėmeliais, o adresai užrašyti šalia.

Pradėkime nuo perėjimo iš MK A. MK C ir D adresai turi būti gretimi, be to, D adresas vienetu didesnis nei C adresas. Tarkime, C adresas bus 6, o D adresas bus 7. Dabar nagrinėkime perėjimą iš MK B. MK D ir E adresai taip pat turi būti gretimi, be to, E adresas vienetu mažesnis nei D adresas. Kadangi D adresas jau priskirtas ir lygus 7, tai E adresas turi būti vienetu mažesnis nei D adresas ir lygus 6. Tačiau toks adresas suteiktas MK C.

Konfliktas gali būti išspręstas dviem būdais.

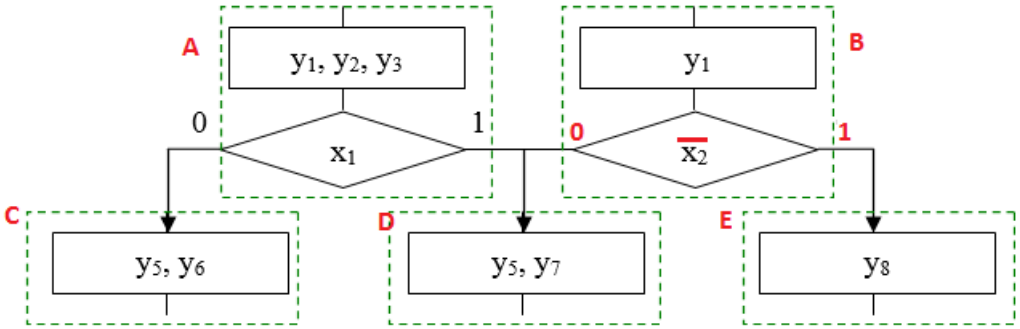
1. 1) Pakeiskime vieną iš sąlygų priešinga; tuomet pasikeis vietomis ir tos viršūnės išėjimų reikšmės 3.2.



2.2 pav. Priverstinės adresacijos adresų konflikto sprendimo strategija 1

Tarkime, C adresas bus 6, o D adresas bus 7. Dabar E adresas turi būti vienetu didesnis nei D adresas ir lygus 8. Adresavimo problema išspręsta.

2. 2) Dubliuokime MK D; tuomet perėjimai bus atskirti 3.3.



2.3 pav. Priverstinės adresacijos adresų konflikto sprendimo strategija 1

Tarkime, C adresas bus 6, o D1 adresas bus 7. Dabar E adresas turi būti siejamas su D2 adresu; tarkime D2 adresas bus 8, o E adresas lygus 9. Adresavimo problema taip pat išspręsta.

2.0.2 Natūrali adresacija [N]

Panagrinėsime MK adresacijos būdą, vadinamą **natūralia adresacija** (N); šiuo atveju MK turi du formatus. MK formato tipas nurodomas lauke **P**.

Operacinė MK turi tik laukus mikrooperacijoms nurodyti 3.2

Mikro komandos struktūra natūralioje adresacijoje

2.2 lentelė. Natūralios adresacijos operacinės MK struktūra

P	Y_1	Y_2	Y_3	...	Y_k

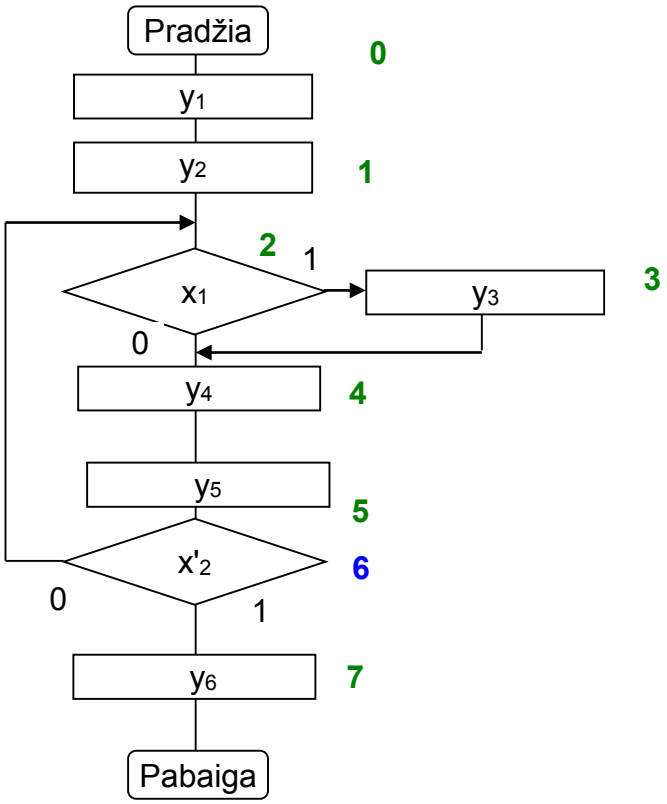
Adresinė (perėjimo) MK turi tik laukus **LS** ir **A** 3.3

2.3 lentelė. Natūralios adresacijos sąlyginės MK struktūra

P	LS	A

Paskesnės MK adresas nustatomas pagal tokias taisykles:

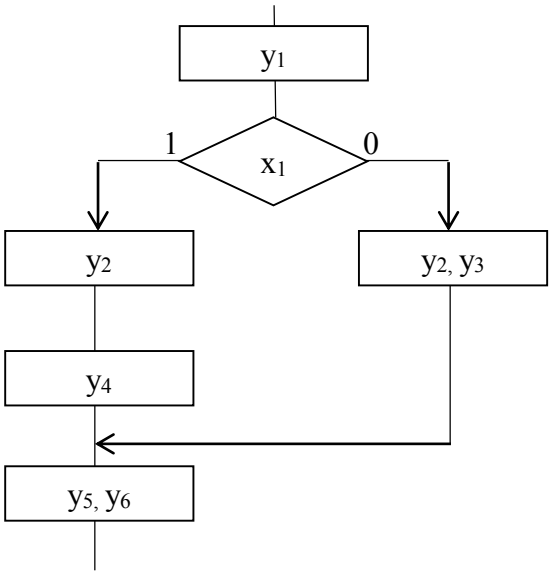
1. po operacinės MK $A_{t+1} = A_t + 1$ (t.y., bus išrenkama sekanti iš eilės MK);
2. po adresinės MK, jei tikrinamos sąlygos reikšmė lygi 1 — $A_{t+1} = At + 1$;
3. po adresinės MK, jei tikrinamos sąlygos reikšmė lygi 0 – $A_{t+1} = A$. Besąlyginio perėjimo atveju $LS = 0$, o $A_{t+1} = A$.



2.4 pav. Adresų priskyrimas natūralioje adresacijoje1

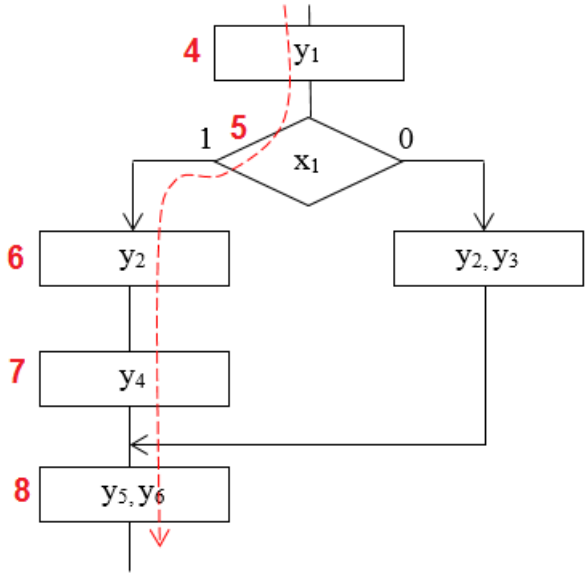
MK adresus mes priskyrėme nuosekliai eidami punktyrine linija pažymėtu keliu 3.4. Atkreipkite dėmesį, kad tikrinant loginę sąlygą kelias pasuka per tą tikrinimo viršūnės išėjimą, kuris pažymėtas `1ı. Jei palyginsite šį mikroprogramos grafą su pateiktu priverstinės adresacijos atveju, pastebėsite, kad loginė sąlyga $x2$ pakeista į x‘2. Tai padaryta tam, kad punktyrine linija pažymėtas kelias pasuktų žemyn. Loginės sąlygos tikrinimo viršūnės išėjimų reikšmės pasikeičia priešingomis, invertuojant sąlygą (pavyzdžiui, jei $x2$ atitinka sąlygą $Sk = 0$, tai x‘2 atitiks sąlygą $Sk <> 0$). Kadangi punktyrine linija pažymėtas kelias aina per visas viršūnes, adresavimas baigtas.

Natūralios adresacijos atveju taip pat gali iškilti adresacijos sunkumų. Panagrinėkime tokį mikroprogramos struktūros fragmentą 3.5



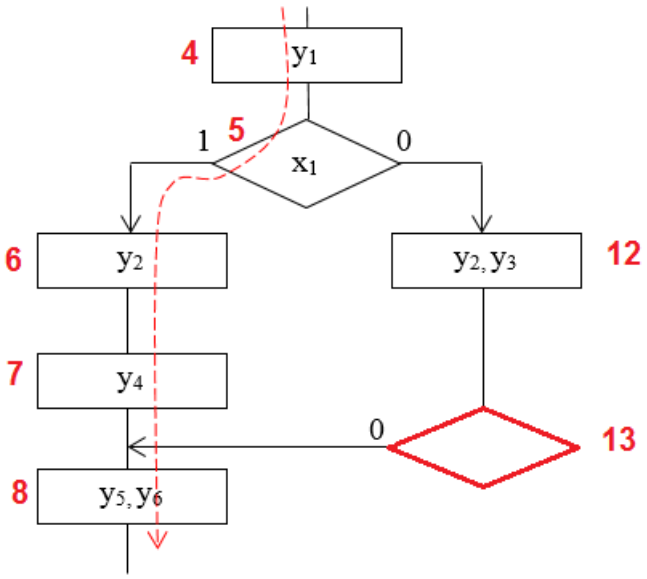
2.5 pav. Adresų konfliktas natūralioje adresacijoje 1

Adresavimui nubrėžiame kelią pagal minėtą taisyklę ir eidami tuo keliu priskiriame adresus, pradedant, tarkime, 4 3.6



2.6 pav. Adresų konfliktas natūralioje adresacijoje 2

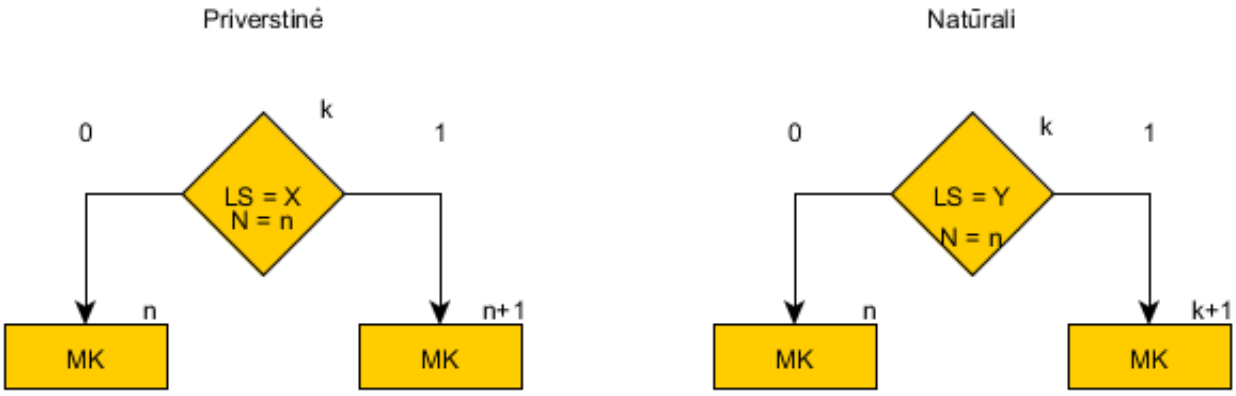
Kelio nuošalyje liko viršūnė su mikrooperacijomis $y2$, $y3$. Tarkime, jai suteikiame adresą 12. Kad po 12–os MK galėtume pereiti prie 8–os MK, įterpsime besąlyginio perėjimo MK adresu 13 3.7



2.7 pav. Adresų konfliktas natūralioje adresacijoje 3

Adresavimo problema sėkmingai išspręsta.

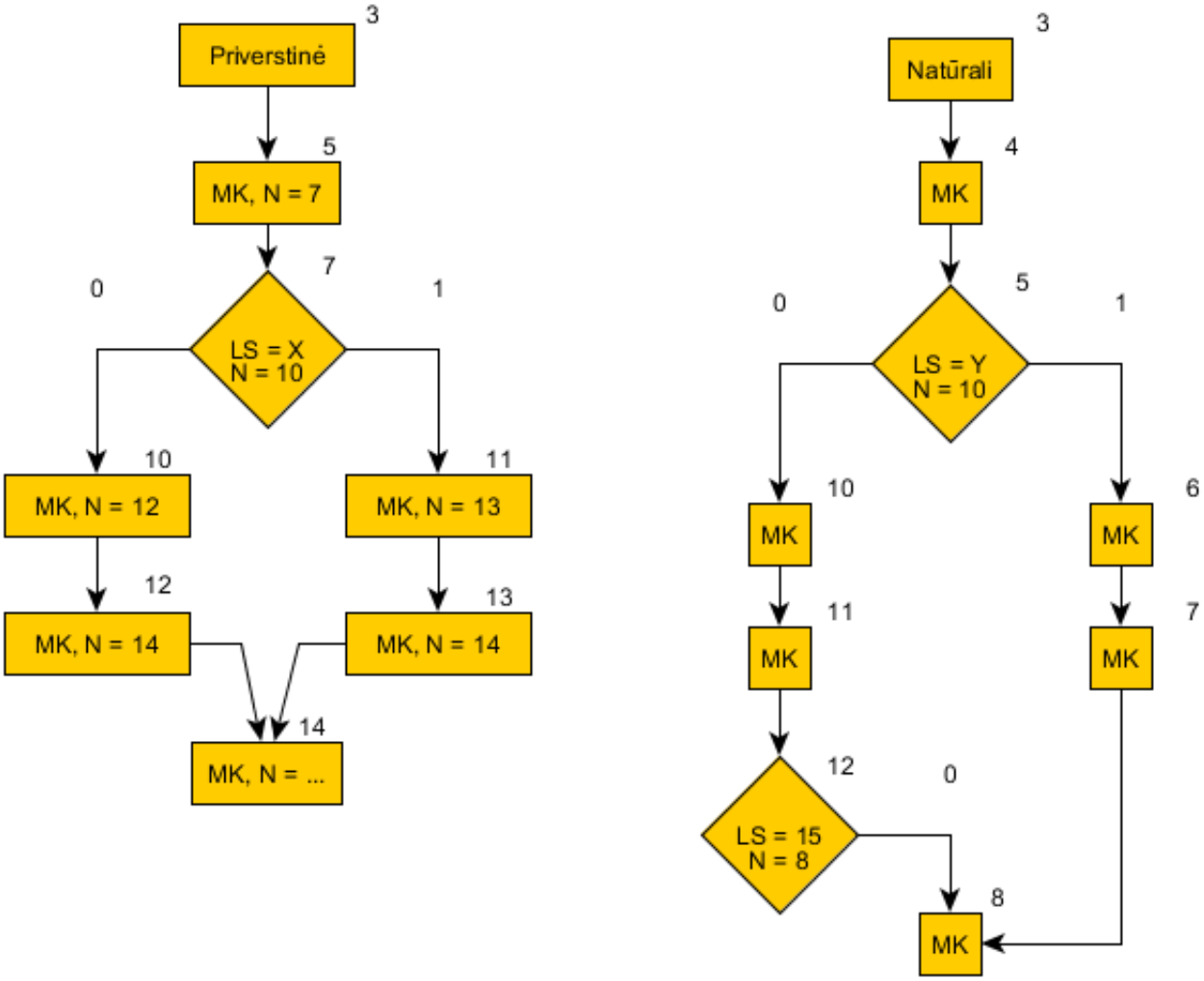
Paveikslėlyje 3.8 yra sulyginamos priverstinės ir natūralios adresacijos.



2.8 pav. Priverstinės ir Natūralios adresacijos skirtumas, nustatant sekantį adresą loginės sąlygos komandoje

Paveikslėlyje 3.9 pateiktos priverstinės ir natūralios adresacijos algoritmų ištraukos, ku-

riose N atspindi sekančios būsenos numerį, LS - tikrinamos loginės sąlygos numerį (kelinto procesoriaus flag'o reikšmė bus tikrinama), o skaičius dešinėje, virš kiekvienos algoritmo grafo viršūnės - kelintoje ROM'o eilutėje ši mikro komanda yra įrašyta. MK atspindi mikro operacijų imtį, kuri bus vygdoma duotajame algoritmo žingsnyje.

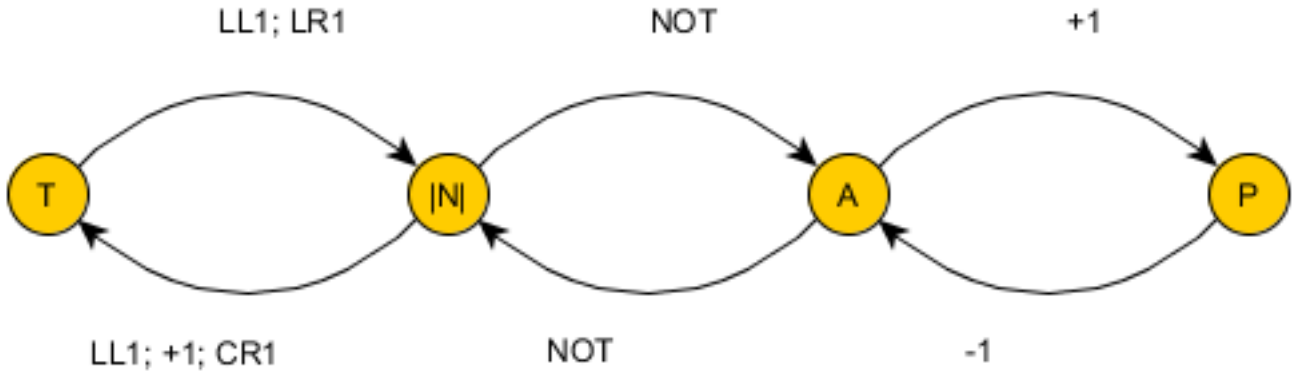


2.9 pav. Palyginimas Priverstinės ir Natūralios adresacijos algoritmų skirtumų

3 skyrius

Skaičių kodų transformacijos

Norint transformuoti skaičius iš vieno kodo į kitą, pateikiame transformacijų grafą 4.1, kuriame atvaizduotas galimas transformacijos kelias tarp skirtingų kodų. Naudojamos tiek ALU, tiek vidinės registrų funkcijos.



3.1 pav. Parėjimai tarp skirtingų skaičių kodų, naudojantis procesoriaus funkcijomis, kur T yra neigiamas skaičius Tiesioginiame kode, $|N|$ yra teigiamas skaičius (skaičiaus modulis), A yra neigiamas skaičius atvirkštiniame kode, o P yra neigiamas skaičius papildomame kode.

3.1. Pavyzdys

$$\frac{N1 - N2^2}{N3} \tag{3.1}$$

Pateiktoje lygtyje 4.1 operandai per įvestį yra atsiunčiami tiesioginiu kodu ir jų atitinkami ženklai yra $(-/-/-)$. Procesorius gali sudėti neigiamus skaičius tarpusavyje, arba su teigiamais, tik tada, kai jie yra papildomame kode. Norint atlikti $(-N1) - N2^2$ operaciją, $N1$ operandas turi būti transformuotas į papildomą kodą. Pagal transformacijų grafą 4.1, galime sudaryti algoritmą, kuris realizuos ją:

1. LL1(B);
2. LR1(B); MUX = B;
3. M = not L; A = M;
4. M = R + 1; A = M; MUX = A;
5. B = A;
6. Reset A;

Procesoriaus mikro operacijos yra grupuojamos į mikro komandas įvertinant paralelinę procesoriaus įrenginių veikimo prigimtį, pvz.: vienoje komandoje galima atlikti dvi mikro operacijas skirtinguose registruose ir ALU, tačiau tik po vieną mikro operaciją per vieną įrenginį - registras gali tik įrašyti duomenis, arba tik atlikti poslinkį. Atlikus šiuos veiksmus, pradinė $N1$ vertė tiesioginiame kode, yra pakeičiama į neigiamą skaičių išreikštą papildomu kodu.

Norint pakelti $N2$ operanda kvadratiniu laipsniu, mums reikia jį sudauginti su savimi viena kartą. Tam mes turime turėti jo modulį dviejuose registruose (galima tiesiogiai sudauginti du neigiamus skaičius, naudojantis pvz Berkso, Goldsteinoo ir Neimano metodu, tačiau tada operandas turėtų būti pervestas į papildomą kodą). Darant prielaidą, jog $N2$ operandas yra patalpintas į C ir D registrus, konvertuoti juos į skaičiaus modulį mums užteks 2 mikro komandų:

1. LL1(C,D)
2. LR1(C,D);