УДК 519.67

МАШИНАЛЫҚ КОДТЫ АЛГОРИТМДЕУ ӘДІСІМЕН ТЕЛЕКОММУНИКАЦИЯЛЫҚ ҚҰРЫЛҒЫЛАРДЫҢ БАҒДАРЛАМАЛЫҚ ЖАСАҚТАМАСЫНДАҒЫ ОСАЛДЫҚТАРДЫ ІЗДЕУГЕ АРНАЛҒАН УТИЛИТА

Сагинбаев Аскар Маргуланович
Қарағанды мемлекеттік туниверситетінің магистранты
Қарағанды Қ., Қазақстан

Аннотация

Зерттеу пәні. Мақала бастапқы код болмаған жағдайда осалдықтарды іздеуге мүмкіндік беретін телекоммуникациялық құрылғылардың машиналық кодын алгоритмдеудің авторлық әдісіне арналған. Әдістің негізгі кезеңі утилита түрінде арнайы бағдарламалық құралды қолдана отырып, оны автоматтандыру үшін зерттеледі. **Әдісі.** Утилитаны іске асырудың негізгі алғышарттары компиляция мен декомпиляция кезеңдеріне жақын тәсілдерді, сондай-ақ әдісте қолданылатын модельді талдау арқылы алынды. **Негізгі нәтижелер.** Утилитаның функционалды архитектурасы ұсынылған, оның ішінде оны кезеңдерге бөлу және функционалды Модульдер. Модульдердің жұмысы мен өзара әрекеттесуі үшін пайдаланылатын ішкі қызметтік деректер көріністерінің түрлері анықталды. Сынақ деректеріндегі кезеңдер мен жеке модульдердің жұмыс мысалдарын гипотетикалық модельдеу жүргізілді. **Практикалық маңыздылығы.** Ұсынылған архитектураны алгоритмдеуге жақын есептерді шешетін құралдардың бүкіл тобын жүзеге асыру үшін қолдануға болады.

Түйінді сөздер: телекоммуникациялық құрылғылар, бағдарламалық қамтамасыз ету, осалдықтарды іздеу, машиналық код, Алгоритмдеу әдісі, функционалдық архитектура.

Кіріспе

Қазіргі заманғы телекоммуникациялық құрылғылардың функционалы бағдарламалық кодтың көмегімен толық іске асырылған, ондағы осалдықтар өңделетін ақпараттың құпиялылығын, тұтастығын және/немесе қолжетімділігін бұзуы мүмкін [1]. Оларды бейтараптандыру жеткізілетін бағдарламалық жасақтаманың меншікті сипатымен күрделене түседі, бұл байланыс операторында осы құрылғылардың бастапқы бағдарламалық кодының болмауына әкеледі [2]. Мұнда мүмкін болатын тәсіл-осалдықтарды олардың машиналық кодынан тікелей іздеу, ол үшін кейбір әдістер мен бағдарламалық құралдар жиынтығы бар. Алайда соңғылары бірқатар себептер бойынша (жұмыстың жылдамдығы мен еңбек сыйымдылығы, осалдықтарды анықтаудың субъективтілігі, мамандардың

жоғары талап етілетін біліктілігі) осы міндеттің қанағаттанарлық шешімін қамтамасыз етпейді, ол үшін алгоритмдеудің авторлық әдісі (бұдан әрі - әдіс) ұсынылады.

Әдістің мәні-бастапқы кодты құрастыру арқылы алынған оның машиналық көрінісі бойынша кодтың жұмыс алгоритмдерін қалпына келтіру. Мұндай алгоритмделген көріністі (бұдан әрі-көрініс) арнайы белгілеуде [3, 4] кодтың осалдықтарын қолмен іздеу үшін пайдалануға болады. Әдістің жұмыс принципі декомпиляцияға ұқсас болғанымен, айтарлықтай айырмашылықтарға ие; атап айтқанда, бастапқы кодты қалпына келтіру тек негізгі мақсат емес, оны қолдану тиімділігін одан әрі нашарлататын еді. Сондай - ақ, бұл әдіс архитектурадағы және Код алгоритмдеріндегі қателіктер үшін жауап беретін орташа деңгейлі және жоғары деңгейлі осалдықтарды іздеуге арналған, ал төмен деңгейлі-жеке код операцияларындағы қателіктерге байланысты [5, 6].

Бұл әдіс бастапқы кодтың құрылымдық ерекшеліктері-архитектура, Модульдер, Кіші бағдарламалар және олардың алгоритмдері туралы ақпаратты қамтитын ішінара қалпына келтірілген деректерді көрсететін белгілі бір S моделіне [7, 8] машиналық кодтың нақты данасын түрлендіреді. Осыған байланысты метадеректер мен модель құрылымдық деп аталады (СМД, құрылымдық метадеректер).

[9, 10] - ден қажетті нәтижелерді алудың сәттілігіне әсер ететін әдістің негізгі (негізгі) кезеңі "2 кезең. Алгоритмдеудің арнайы бағдарламалық құралымен (бұдан әрі - утилитамен) іске асырылатын ассемблерлік көрсетілімді Алгоритмдеу".

Утилита архитектурасын жобалау, оны іске асырудан бұрын оның көрінісін әртүрлі позициялардан жасауға мүмкіндік береді: функционалды-негізгі модульдерді және олардың өзара қоңыраулар мен деректерді беру деңгейіндегі өзара әрекеттесуін ашады; Ақпараттық - утилитаның ішкі деректерінің көрінісін және оларды түрлендіруді анықтайды; және бағдарламалық – осы арнайы құралды іске асыратын бағдарламалық кодтың құрылымын сипаттайды.

Әрі қарай, біріншісі ретінде, ішкі көріністерді және нақты бағдарламалық жасақтаманы егжей-тегжейлі көрсетпей, утилитаның бүкіл процесін анықтайтын функционалды архитектура қарастырылады.

Архитектура элементтері және функционалдық архитектура схемасы

Алгоритмдеу PS жұмысы келесі кезеңдерден тұрады (құралдың негізгі функционалдығын дәйекті түрде орындайды), компиляторларды әзірлеу саласында келесі ағылшын тіліндегі атаулар бар: Front-End, Middle-End және Back-End. Бірінші кезеңде талдау грамматиканы қолдану және оның ішкі көрінісін құру арқылы жүзеге асырылады. Front-End жұмыс істеуі үшін сізге өзіңіздің лексикалық анализаторыңызды енгізу,

кеңейтілген ассемблердің ресми грамматикасын жасау, дерексіз синтаксис ағашын құру үшін грамматика ережелерін және ағашты платформалық тәуелсіз ішкі көрініске айналдыру алгоритмдерін енгізу қажет. Екінші кезеңде салынған ағаштарды өңдеу, СМД бөлу, осалдықтар туралы ақпарат жинау және S-моделін құру (графтар, кестелер, хэштер, қосымша ағаштар және т.б. жиынтығы арқылы) жүргізіледі. Сондай-ақ, жалпы бағдарламалау практикасы негізінде жоғалған СМД-ны қалпына келтіруге болады; берілген модельдеу түзетулерінің негізгі бөлігі осы жерде ескеріледі. Middle-End жұмыс істеуі үшін графиктерді айналып өту алгоритмдерін орындау және олардың элементтерінің осалдық үлгілеріне сәйкестігін іздеу қажет. Үшінші фазада Шығыс синтаксисіне сәйкес мәтіндік түрге айналдырылған АК алгоритмдік көрінісі жасалады. Егер Middle-End-де жиналған ақпарат болса, мұндай көріністі құру және оны құру технологиялық тұрғыдан маңызды емес.

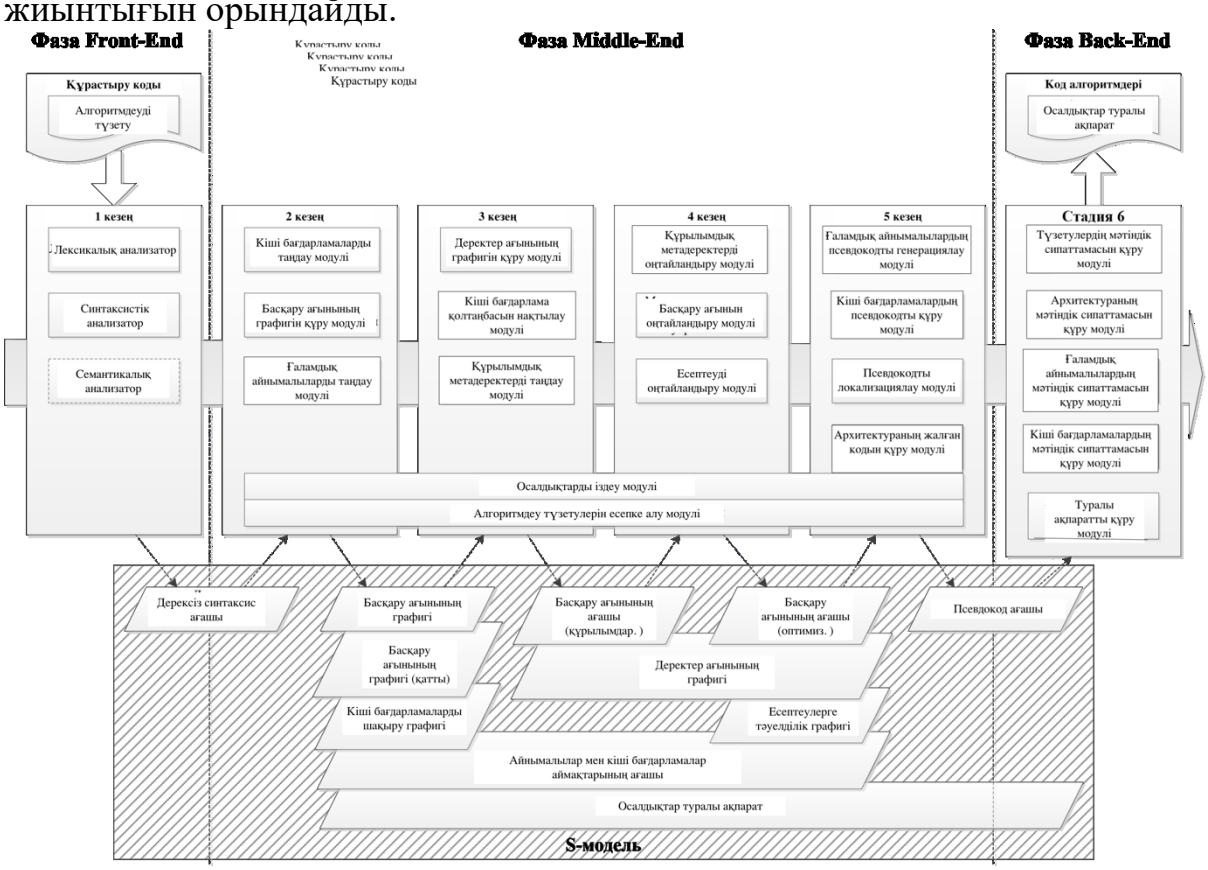
Утилитаның мақсаты мен функционалдығының компиляция мен декомпиляция кезінде қолданылатын тәсілдерге жақындығы айқын болғандықтан, оның архитектурасын ұқсас етіп жасаған жөн. Сонымен қатар, архитектураның негізгі элементі s-модель болып саналады, оның айналасында қалған функциялар салынады. Утилита консоль қосымшасы сияқты көрінуі мүмкін, өйткені пайдаланушының барлық өзара әрекеттестігі кіріс пен Шығыс талдауына дейін азаяды. Утилитаны жөндеу үшін оның ішкі көріністерінің күйін келтіру нәтижесін пайдалану жеткілікті. Утилитаның жұмыс принципі әдіс кезеңіне сәйкес әрекеттерді дәйекті (фазалық) орындауға дейін азайтылуы мүмкін (мысалы, хабар алмасуға немесе ішкі күйлер арасындағы ауысуларға байланысты жұмыс принципі бар графикалық интерфейсі бар қосымшалардан айырмашылығы). Бұл Утилитада сыртқы нысандармен (басқа PS және дерекқорлармен) деректер алмасудың тікелей қажеттілігі жоқ. Мұндай архитектурадағы S-модель әртүрлі құрылымдардың жиынтығымен анықталады – қызметтік модульдердің алгоритмдері анықталған және АК кірісінің көрінісін анықтайтын ішкі (аралық) қызметтік деректер. Утилитаның функционалды архитектурасы схемалық түрде 1-суретте көрсетілген .

1-суретте келесі архитектура элементтері бар. Біріншіден, Бұл утилитаның жоғары қаралған фазалары. Екіншіден, бұл архитектура модульдері және олардың өзара әрекеттесуі. Үшіншіден, бұл-кіру және шығу мәліметтері Утилиттер. Төртіншіден, бұл S моделін анықтайтын ішкі қызметтік деректер. Көріп отырғанымыздай, архитектура-бұл АК-ны оның алгоритмдік көрінісіне айналдырудың фазалық көп сатылы орындалуы. Әр кезең оның функционалдығын жүзеге асыратын модульдерден тұрады. Модульдер ішкі деректерді түрлендіреді, сонымен қатар олардың арасында алмасу әдісі ретінде қолданылады. Мұндай мәліметтердің жиынтығы s-модельдің ішкі көрінісін құрайды, ол МК бойынша бірінші кезеңде

құрылады және барлық кейінгі сатыларда өңделеді; сонымен қатар, соңғы кезең әлеуетті осалдықтар туралы ақпараты бар кіріс АК алгоритмдері түрінде алгоритмделген көрініс жасайды.

Модульдік құрылым

Барлық архитектура модульдері тәуелсіз орындалу бірлігі болып табылады. Қызметтік бағдарламаның әртүрлі кезеңдеріндегі Модульдер арасындағы алмасу қызметтік бағдарламаның ішкі көріністері арқылы жүзеге асырылады; осылайша, бір сатыдағы модульдердің әр жиынтығы жаңа немесе ескі көріністі өзгерту үшін нақты белгіленген әрекеттер жиынтығын орындайды.



Сурет 3.1-қызметтік бағдарламаның функционалды архитектурасы (Алгоритмдеу бағдарламалық құралы)

Модульдердің кезеңдер бойынша өзара әрекеттесуі келесідей: 1-кезең модульдері АК кірісіне қабылданады және дерексіз синтаксис ағашын жасайды; 2-кезең модульдері абстрактілі синтаксис ағашын қабылдайды және басқару ағынының графигін жасайды және кіші бағдарламалардың қоңырау графигі, айнымалылар мен кіші бағдарламалар аймағын; 3-кезең деректер ағынының графигін жасайды және басқару ағынының ағашы; 4-кезең модульдері бұл ағашты оңтайландырады; 5-кезең модульдері оған жалған код ағашын салады, ол 6-сатыдағы Модульдер АК алгоритмдерінің мәтіндік сипаттамасын құру үшін қолданылады. Сондай-ақ, 4-кезеңде

есептеулер тәуелділігінің графигі құрылады, сол кезеңде оңтайландыру үшін қолданылады.

Сатылар арасындағы модульдердің тікелей өзара әрекеттесуі іс жүзінде жоқ, өйткені мұндай модульдердің әрқайсысы қызметтік деректерді (график немесе ағаш) кезеңге енгізу немесе одан шығу үшін өзінің шектеулі функционалдығын жүзеге асырады. Модульдерді мұндай ұйымдастыру, атап айтқанда, ағаштарды оңтайландыру және қысқарту үшін айтарлықтай еңбек шығынсыз жаңа функционалдылықты қосуға мүмкіндік береді. Соңғысы, оңтайландырудан айырмашылығы, кодтың субъективті ерекшеліктерін жақсартуға арналған, мысалы: ықшам және адамның қабылдауы, алгоритмдерді неғұрлым ықшам өрнектерді, арнайы тілдік құрылымдарды, аталған мекен-жайларды және т. б. қолдану арқылы. Ерекшелік екі модуль болып табылады – Алгоритмдеу түзетулерін есепке алу және осалдықтарды іздеу, өйткені олар соңғы модульдердің орындалу сатысына қарамастан, барлық басқа модульдерге деректер мен өз функционалдығын ұсынады.

Біз утилитаның жұмыс кезеңдерін және оларды іске асыратын модульдерді толығырақ сипаттаймыз.

1-кезең

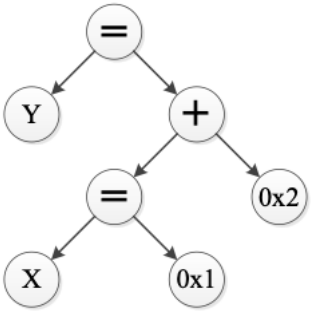
Кезең кіріс АК-ны талдауға және оның ішкі көрінісін Front-End фазасында құруға арналған. Кезең компилятор фазасының классикалық мысалы болып табылады және 3 модульден тұрады: лексикалық, синтаксистік және семантикалық анализаторлар. Біріншісі АК таңбаларының кіріс ағынын жеке таңбалауыштарға бөлуге арналған (сөздерге әріптер жинауға ұқсас), осылайша негізгі формализацияны жүзеге асырады. Екіншісі алдын-ала белгіленген ережелермен (ауызша сөйлемдерге ұқсас) салыстыра отырып, жеке таңбалауыштарды жинайды. Мұндай талдау үшін рекурсивті ережелерді қолдана отырып, кіріс тілінің синтаксисімен берілген ресми грамматика қолданылады; соңғылары ережелерді құрастыру кезінде орындалатын қолданушылық әрекеттермен (қолданылатын бағдарламалау тілінде) толықтырылады – яғни.кіріс таңбалауыштарының ережесіне сәйкес. Нәтижесінде кіріс кодын толық формаланған және құрылымдалған түрде көрсететін дерексіз синтаксис ағашы жасалады. Үшінші кезең модулі – семантикалық анализатор ережелердің семантикалық мағыналарын талдауға арналған. Классикалық компиляторларда оған маңызды орын берілсе де, қазіргі архитектурада модульді шартты деп санауға болады, өйткені ол тек көмекші әрекеттерді орындайды, мысалы, ағашқа процессордың тиісті нұсқаулары үшін салыстыру нәтижелерін сақтайтын түйіндердің абстрактілі синтаксисін қосу. Модульдер жұмысының нәтижесінде келесі кезеңдерде өңдеуге жарамды кіріс АК-нің дерексіз синтаксис ағашы жасалады.

Айта кету керек, кезең орындалу процессорына тәуелді болса да (ол кіріс АК синтаксисін қолданғандықтан), бірақ анализаторларды іске асыруды олар жасаған дерексіз ағаш процессордың нұсқауларын пайдаланбайтындай етіп жасауға болады, бірақ дерексіз операциялар мен айнымалылармен жұмыс істейді. Мысалы, PowerPC процессоры үшін келесі АК:

```
LI R0, 0x1 ; R0 = 1
ADDI R1, R0, 0x2 ; R1 = R0 + 0x2
```

синтаксистік талдау барысында процессордың тәуелсіз кодына ұқсас келесі дерексіз ағашқа (2-сурет) айналдыруға болады:

```
X = 0x1;
Y = X + 0x2
```



Сурет 2-Қарапайым өрнектің дерексіз ағашының мысалы

Front-End-ден кейінгі деңгейдің процессорлық өзгергіштігін қамтамасыз ету жалпыланған (шаблондық) алгоритмдерді жасауға мүмкіндік береді, бұл кодтың тасымалдануын да, оның сенімділігін де арттырады.

2-кезең

Кезең Middle-End фазасындағы бірінші болып табылады және абстрактілі синтаксис ағашын бастапқы өңдеуге және оны ішкі қабаттар, басқару ағындары, Ғаламдық және жергілікті айнымалылар және т.б. туралы ақпаратты қамтитын әртүрлі қабаттарға бөлуге арналған. Біріншіден, ішкі бөлу модулі абстрактілі синтаксис ағашын талдайды, ішкі бағдарламаларды анықтайды және олар туралы ақпаратты SCT-ге енгізеді. Екіншіден, сәйкестендіру нәтижелерін қолдана отырып, басқару ағынының графигін құру модулі ішкі бағдарламаның ішіндегі барлық өтулерді анықтайтын тиісті график жасайды (яғни алгоритмнің мәні). Басқару ағынының графигін бұрынғыға қарағанда формалды құрылымы бар "қатаң" деп аталатын (графикте заңдылықтарды қатаң белгілейтін қызметтік түйіндерді қосу арқылы) түрлендірген жөн – бұл оны өңдеу алгоритмдерінің орындалуын жеңілдету үшін қажет. Онда, мысалы, филиалда әрдайым кем дегенде бір түйін болуы үшін шартты тармақтардың басында қызметтік түйіндер қосылады – "қатты емес" бағанда бос бұтақта түйіндер болмайды, бұл оны өңдеу алгоритмдерін түйіндердің болуын

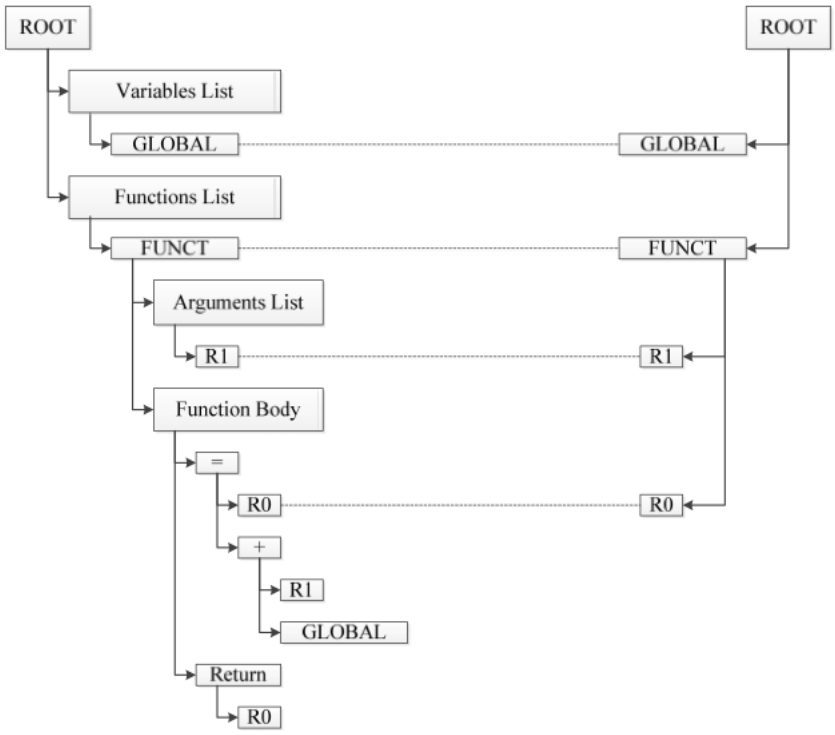
қажетті тексерулермен қиындатады және т.б. басқару ағынын құру процесінде сыртқы кіші бағдарламалардың қоңыраулары талданады, бұл олардың қоңырауларының графигін параллель құруға мүмкіндік береді. Үшіншіден, абстрактілі синтаксис ағашын Ғаламдық айнымалыларды бөлу модулімен талдау соңғы ақпаратты SCT-ге сәйкестендіреді және енгізеді. Бұл ағаш компиляторларда қолданылатын көріну аймағының жеңілдетілген аналогы болып табылады. Осылайша, SCT барлық айнымалылар туралы да, кіші бағдарламалар туралы да ақпаратты сақтайды, соның ішінде соңғысының басқару ағынының жанама бағандары (сілтемелер арқылы). С-тәрізді псевдокодтың келесі мысалына ұқсас дерексіз синтаксис ағашын өңдеу:

```
var GLOBAL; FUNCT(R1) {
R0 = R1 + GLOBAL ; return R0;
}
```

3-суретте көрсетілгендей, басқару ағынының графигін салады және SCT-ге деректерді енгізеді.

3-кезең

Кезең ішкі көріністе СМД бөлуге арналған Smодельдер алдыңғы сатыларда алынған. Ол үшін, атап айтқанда, деректер ағынының графигін құру модулі айнымалылардың өмір сүру уақыты, олардың мәні, алғашқы/соңғы пайдалану нүктелері және т.б. туралы басқару ағынының графигіне қосылған ақпаратты жасайды.



3-сурет-абстрактілі синтаксис ағашы мен айнымалылар мен кіші бағдарламалар аймақтарының ағашының өзара байланысының мысалы

Қолтаңбаны нақтылау модулі басқару ағынының графигін талдайды және деректер ағынының графигін қолдана отырып, кіріс және шығыс параметрлері сияқты қолтаңбаның қасиеттерін болжайды. АК аясында ішкі бағдарламаның параметрлері регистрлер жиынтығын білдіреді, олардың көмегімен ішкі бағдарлама сыртқы мәндерді алады және есептеу нәтижелерін береді. Мысалы, с-тәрізді жалған кодтағы ассемблерлік кіші бағдарламаның келесі мысалы үшін:

```
??? FUNCT(???) { R0 = R1 + R2 ; return R0;
}
```

funct() кіші бағдарламасы екі Регистр – R1 және R2 арқылы параметрлерді енгізуі мүмкін, өйткені олар ешқандай нақты инициализациясыз қолданылады және R0 регистрі арқылы есептеу нәтижесін қайтарады, өйткені оған кейінгі пайдаланусыз мән беріледі; қазіргі заманғы компилятордың субоптималды кодты құру нұсқасы іс жүзінде алынып тасталады. Сонымен, осы логиканы қолдана отырып, жоғарыда келтірілген мысал үшін қолтаңбаны нақтылау модулі ішкі қолтаңбаны келесідей анықтайды:

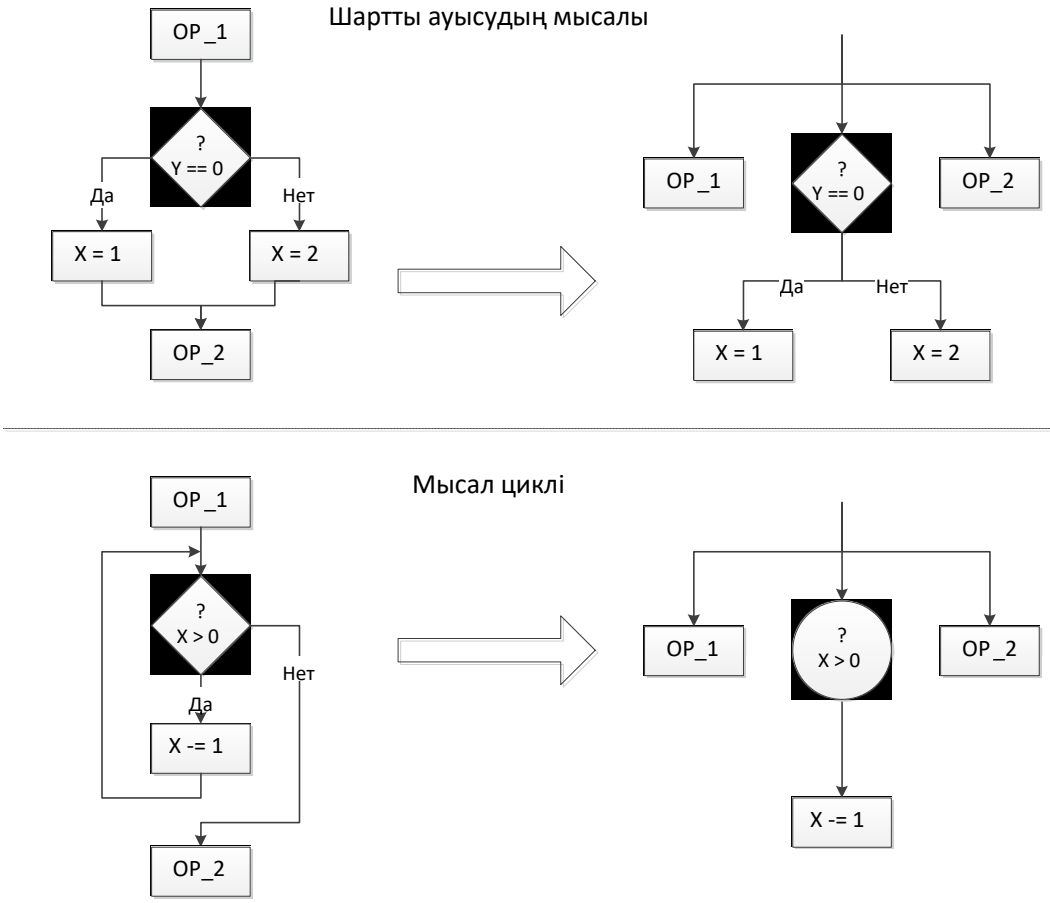
```
( R0 ) FUNCT( R1, R2 );
```

Айта кету керек, ішкі бағдарламаның аяқталған қолтаңбалары үшін (мысалы, с тілі тұрғысынан) дәлелдердің түрлері мен қайтарылатын мәндер де қажет, алайда алгоритмдердің логикасын сипаттау үшін олар маңызды емес және бұл модуль қалпына келмейді. Алайда мұндай қалпына келтіру мүмкін.

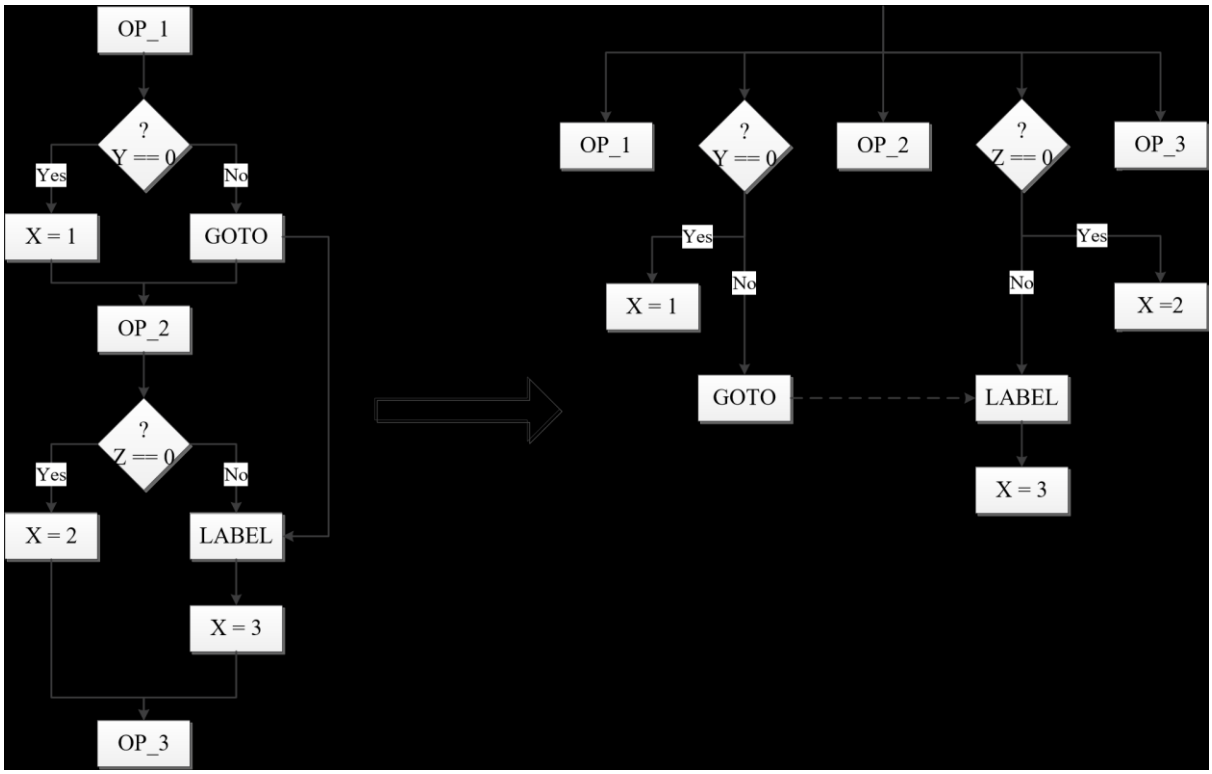
Негізгі кезең модулі (және, белгілі бір мағынада, бүкіл утилита) - бұл ішкі бағдарламаның графикалық басқару ағынын нассишнейдерман диаграммаларына ұқсас ағаш пішініне параллель аударатын SMD таңдау модулі. Модульдің негізгі функциялары барлық шартты өтулерді (басқару конструкцияларын, тармақтарды және аяқтау белгілерін қоса алғанда) және басқару ағынының графигіндегі циклдерді (циклден шығу және итерациялар бойынша өту шарттарын қоса алғанда) бөлуден тұрады. Талдау модулі рекурсиялар мен графиктерді бояу арқылы жүзеге асырылады; содан кейін олар Насси-Шнайдерман құрылымдарын пайдалануға қайта құрылады. Қарапайым шартты ауысу және цикл үшін мұндай қайта құру мысалдары В. 3 суретте көрсетілген.

Басқару ағынының графигін ағаш құрылымына дейін азайту мүмкін емес жағдай (мысалы, goto сөзсіз ауысу операторы болған кезде) дегенеративті болып саналады және басқару ағынының ағашында қосымша байланыс енгізу арқылы сипатталады (Б.4-сурет).

Мысалдарда айқын көрініп тұрғандай (В. 3 және В. 4 суреттері), басқару ағынының ағаш тәрізді ішкі көрінісіне көшу код формасының құрылымын арттырады, бұл оның адамның қабылдауына оң әсер етуі керек.



4-сурет-Шартты ауысу және цикл үшін Насси-Шнайдерман диаграммаларын құру мысалдары



5-сурет-ағаш құрылымына қысқартылмаған сөзсіз ауысудың мысалы

4-кезең

Кезең оның мөлшерін азайту және Us жеңілдету үшін кодты ұсынуды оңтайландыруға арналған; ол 3 модульден тұрады:

1) келесі әрекеттерді орындайтын СМД оңтайландыру модулі:

- пайдаланылмаған белгілерді жою, сондай-ақ тармақталғаннан кейін келесі операцияға, циклдің басына және циклден кейінгі бірінші нұсқаулыққа сөзсіз көшу;

- белгілерді, сондай-ақ шартты ауысулар мен циклден шығудың бірнеше тармағын біріктіру;

- шартты өткелдердің тармақтарын қайта сұрыптау;

- циклден тыс ішкі бағдарламадан шығу.

2) циклден сөзсіз шығуды BREAK конструкциясына ауыстыруды және келесі цикл итерациясының сөзсіз басталуын NEXT конструкциясына ауыстыруды жүзеге асыратын басқару ағынының ағашын оңтайландыру модулі;

3) келесі әрекеттерді орындайтын есептеулерді оңтайландыру модулі: − аралық мәндерді қоса, өрнектердің мәндерін есептеу;

- өрнектердің есептелген мәндерін тиісті тұрақтыларға, сондай-ақ айнымалыларға бит қатынасын VAR түріндегі арнайы конструкцияларға ауыстыру.N_bit; - айнымалылардың инициализациялық мәндерін алмастыру арқылы өрнектерді жеңілдету;

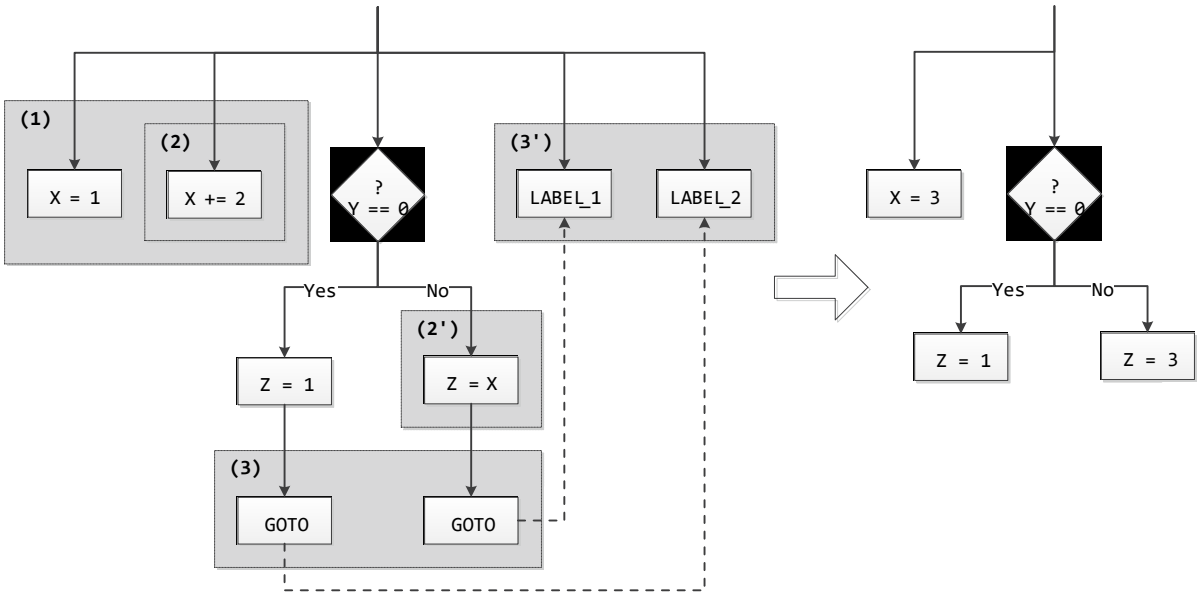
- логикалық операцияларды, салыстыру операцияларын, сондай-ақ теріс сандарды қосу/азайту операцияларын жеңілдету;

- әсері жоқ операцияларды, сондай-ақ әсері бағдарламаның күйін өзгертпейтін операцияларды жою.

Басқару ағынының ағашын оңтайландыру мысалы 6-суретте көрсетілген.

Кезең барысында есептеулердің тәуелділік графигі құрылады, ол кодтағы өрнектерді және есептеулер тәуелді айнымалыларды көрсетеді. График оңтайландыру модульдері сатысында ғана қолданылады және басқару ағынының ағашын қалпына келтірудің көптеген ірі операцияларынан кейін жаңартылады.

Басқару ағынын оңтайландыру мысалы



6-сурет-басқару ағынын оңтайландыру мысалы

Ескерту. 6-суретте келесі оңтайландырулар қолданылады: (1) - өрнектердің мәндерін есептеу, (2)+(2') – өрнек мәндерін алмастыру, (3)+(3') – тармақталғаннан кейін операцияға ауысуларды алып тастау.

Барлық Модульдер Бірнеше өтулерде орындалады, олардың бірде – бір оңтайландыруы нәтиже бермейді-яғни.утилитаның ішкі көріністерінде қандай-да бір өзгерістер жасаңыз. Кезеңді айналдыру модульдердің барлық оңтайландырулары әрқашан код көрінісін жеңілдететін немесе өзгертпейтін етіп жүзеге асырылатындығына жол бермейді.

5-кезең

Кезең алгоритмдердің псевдокодын тиісті ағаш түрінде құруға арналған (оның ішінде кіші және ғаламдық айнымалылар да бар), содан кейін оны лаконикалық көрініске келтіреді және келесі модульдермен жүзеге асырылады.

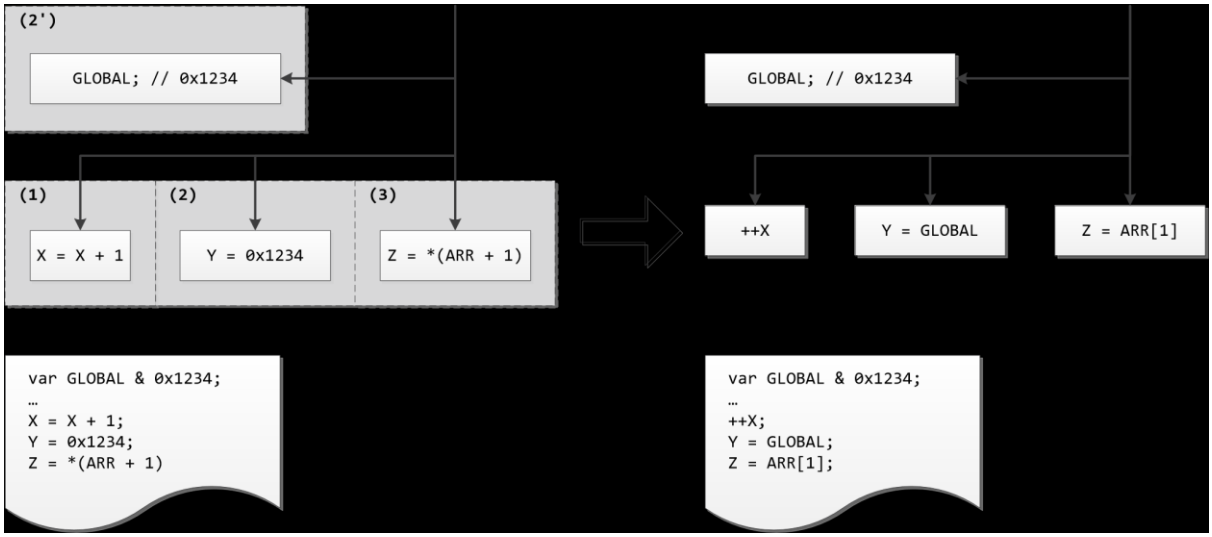
Ғаламдық айнымалылардың псевдокодын құру модулі оларды орналастыру мекен-жайларын (бар болса) көрсете отырып, РСТ-ге қосады. Псевдокодты генерациялау модулі РСТ-де әр Ішкі бағдарламаның ішкі көрінісін іс жүзінде өзгертпейді. Содан кейін, осы ағашта псевдокодтың лаконизация модулі адамның кодты қабылдауын жақсарту үшін оны өзгертеді, бұл келесі әрекеттерді қамтамасыз етеді:

- коммутативті операцияларды сұрыптау және біріктіру;

- абсолютті адрестерді тиісті Ғаламдық айнымалыларға ауыстыру, сондай-ақ арнайы конструкцияларға есептеу операциялары-көбейту/декреттеу (++ ,--), тағайындау операциясы([+-*/]=), массив элементіне кіру.

Pct лаконизациясының мысалы 7-суретте көрсетілген.

Басқару ағынын оңтайландыру мысалы

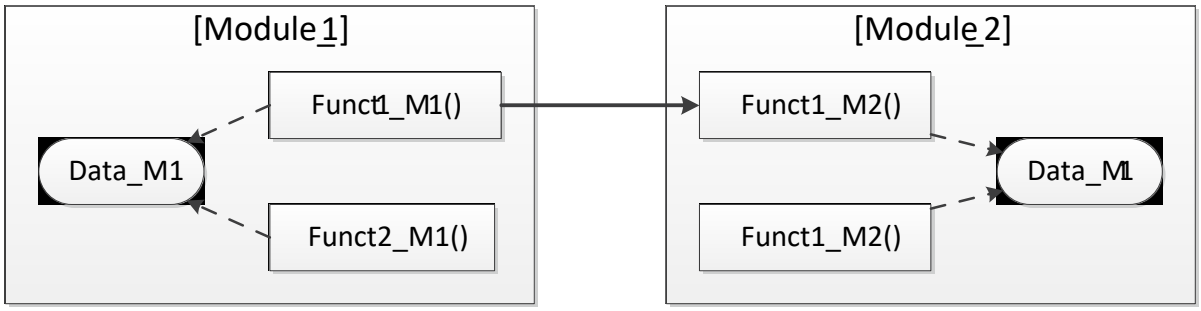


Сурет 7-жалған код ағашының лаконизациясының мысалы

Ескерту. 7-суретте келесі оңтайландырулар қолданылады: (1) - арнайы құрылымға ауыстыру: көбейту, (2)+(2') – мекен-жайдың орнына Ғаламдық айнымалыны алмастыру, (3) - арнайы құрылымға ауыстыру: массив элементіне қол жеткізу.

Модуль лаконизация бойынша оның бірде-бір әрекеті өзгерістер енгізуді тоқтатпайынша бірнеше өту кезінде орындалады.

Архитектураның жалған кодын құру модулі PCT-ге PC архитектурасы туралы ақпаратты модульдерге бөлу арқылы қосады. Осындай бөлу нәтижесінде архитектураның мысалы 8- суретте көрсетілген.



8– сурет-бағдарламалық код архитектурасының мысалы

Суретке сәйкес, архитектура екі модульден тұрады (Module_1 және Module_2), бірінші модульде екі функция бар (Funct1_M1() және Funct1_M2 (); екінші модульде Funct1_M2() және Funct2_M2() модуль деректерін қолданады (Data_M1 және Data_M2; оларды қолдану нүктелі сызықтармен көрсетілген). Бұл жағдайда бір модуль функциясы екіншісінің функциясын шақырады (Funct1_M1 () Funct1_M2 () шақырады; үздіксіз сызықпен көрсетілген).

Осылайша, кезең іс жүзінде кодтың көлемін азайтпайды, тек оны арнайы құрылымдарды қолдану және Код элементтерін ұсынудың басқа

формаларын таңдау арқылы көбірек қабылдайды. Кезең 4-сатыға ұқсас себептерге байланысты тоқтай алмайды; ол Middle-End фазасындағы қорытынды болып табылады.

6-кезең

Кезең құрылған, оңтайландырылған және ықшамдалған ішкі мәліметтер негізінде Back-End фазасындағы код алгоритмдерінің Шығыс сипаттамасын құруға арналған. Кезең-компилятор фазасының классикалық мысалы және келесі модульдерден тұрады.

Алгоритмдеу түзетулерінің мәтіндік сипаттамасын құру модулі кіріс ассемблеріне енгізілген Алгоритмдеу түзетулерін Шығыс кодына қосады. Бұл алынған алгоритмдік презентация (оның ішінде осалдықтар туралы ақпарат) негізделген деп есептелуі үшін қажет, өйткені соңғысы енгізілген түзетулерге байланысты. "Readonly" қатынау режимі бар айнымалыны көрсететін қызметтік бағдарламаның Шығыс кодындағы түзетулердің мысалы (тек оқу үшін) кіріс кодымен толық сәйкес келеді:

```
user_control {
secret_for_read: p_readonly;
}
```

Архитектураның мәтіндік сипаттамасын құру модулі Ғаламдық айнымалылар мен функцияларды модульдерге бөледі, олардың интерфейстерін көрсетеді. Сонымен, 8-суретіндегі мысалдағы архитектура жағдайында утилитаның тұжырымы келесі түрде болады:

```
module Module_1 { var Data_M1;
() Funct1_M1() {
...
Funct1_M2();
}
() Funct2_M1() { ...
} }
module Module_2 {
/* Funct1_M2() - Interface (nCall = 1) */; var Data_M2;
() Funct1_M2() { ...
}
() Funct2_M2() { ...
} }
```

Сондай-ақ, алгоритмдік көрініске сәйкес, Утилита Funct1_M2 () функциясын Module_2 модулі үшін интерфейс ретінде анықтады, өйткені ол Module_1 функциясынан шақырылады.

Ғаламдық айнымалылардың мәтіндік сипаттамасын құру модулі, егер мүмкін болса, олардың мекен-жайларын көрсете отырып, АК арқылы қалпына келтірілген барлық Ғаламдық айнымалылардың сипаттамасын жасайды. Global_1 және global_2 Ғаламдық айнымалылар үшін осындай тұжырымның мысалы, соңғысы 0x1234 мекен-жайы бойынша орналастырылған:

```
var global_1; var global_2 & 0x1234;
```

Кіші бағдарламалардың мәтіндік сипаттамасын құру модулі мәтіндік түрде барлық кіші бағдарламалардың (олардың қолтаңбалары мен алгоритмдерін қоса) сипаттамаларын жасайды. Параметр мәнін көбейтетін және қайтаратын (R0 регистрі арқылы) FUNCT () кіші бағдарламасы үшін осындай нәтиженің мысалы келесідей болады:

```
(r0) FUNCT(r0) { ++ r0; return r0;
}
```

Осалдық туралы ақпаратты құру модулі осалдықтарды іздеу модулі жинаған деректерді қолдана отырып, ықтимал осалдықтардың белгілерін қосады. Алгоритмнің құрылымы бұзылған Destructed_Funct () кіші бағдарламасы үшін осындай тұжырымның мысалы келесідей болады:

```
Destructed_Funct() {
/* ATTENTION!!! Possible, destruction of the structure. */; }
```

Қорытынды

Middle-End фазасының барлық сатыларында (2-5-кезеңдер) екі модуль жұмыс істейді: Алгоритмдеу түзетулерін есепке алу және осалдықтарды іздеу. Біріншісі, кіріс ассемблерін талдау кезінде алынған алгоритмдік түзетулер туралы ақпаратты қолдана отырып, осы кезеңдегі барлық модульдердің жұмысын басқарады – мысалы, көрсетілген ассемблер кіші бағдарламасы үшін оның дәлелдері ретінде пайдаланылатын регистрлерді нақты көрсету. Осылайша, модуль утилитаның ішкі көріністерінің топологиясына да, жеке сипаттамаларына да, алгоритмдердің соңғы мәтіндік түріне де әсер етеді. Екіншісі ықтимал осалдықтар туралы ақпаратты бөлу үшін әртүрлі шаблондар мен алгоритмдерді ұсынады, оларды ақырында оларды шығару үшін осалдықтар туралы ақпаратты құру модулі пайдаланады. Сатылардың сызықтық орындалуы тұрғысынан бұл модульдерді ортогональды деп атауға болады, өйткені олар қандай да бір жолмен басқалармен әрекеттесе алады.

Әдебиет:

1. Ақпараттық қауіпсіздікті қамтамасыз етудің стратегиялық құралы ретінде телекоммуникациялық құрылғылардың машиналық кодын Алгоритмдеу / Израиль / / Ұлттық қауіпсіздік және стратегиялық жоспарлау. - 2013. - № 2 (2). - С. 28-36.

2. Буиневич, м.в. телекоммуникациялық құрылғыларға арналған машина кодын Алгоритмдеу әдісі / М. в. Буиневич, К. Е. Израильов // Телекоммуникациялар. - 2012. - № 12. - С. 2-6.

3. Буиневич, м.в. телекоммуникациялық құрылғылардың машиналық кодын алгоритмдеудің автоматтандырылған құралы / М. в. Буиневич, К. Е. Израильов // Телекоммуникациялар. - 2013. - № 6. - С. 2-9.

4. Израиль, К. Е. кодты қалпына келтіруге арналған прототиптік утилитаның ішкі көрінісі / К. Е. Израиль / / Қазіргі әлемдегі іргелі және қолданбалы зерттеулер: Матер. II Халықаралық ғылыми-практикалық конференция. конф. - СПбГПУ, 2013. - Б.79-90.

5. Деректерді шифрлау стандарты (DES) // Уикипедия: Энциклопедия. URL: http://wikipedia.org/wiki/Data_Encryption_Standard (15.03.2014).

6. Израиль, К. Е. бағдарламалық жасақтаманың архитектуралық осалдығы / К. Е. Израиль / / "ЭНГЕКОН-2013": докл тезистері. Студенттер мен аспиранттардың VI ғылыми съезі. СПБГИУ. 18-19 сәуір 2013-Санкт-Петербург, 2013 .-- Б. 35.

7. Буиневич М., Израильов к. осалдықтарды табуға арналған телекоммуникациялық құрылғылардың кодтық алгоритмдерін қалпына келтіру әдісі мен утилитасы / М. Буиневич, Израиль к. / / 16-шы халықаралық коммуникациялық технологиялар конференциясы (ICACT - 2014), Bongpyeong-myeon, Корея (Оңтүстік), 16-19 ақпан 2014 ж. 172-176 жж.

8. Машиналық кодты қалпына келтіру бағдарламасының онлайн нұсқасы [Израиль К. Е. жеке сайты]. URL: http://demono.ru/onlineDemono.aspx-C.2-6.