ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ РЕСУРСІВ ПРИ СИНТЕЗІ КОДІВ ПРОГРАМ

Салапатов В.І.
Кафедра кібернетики, ЧНУ
mpzas@cdu.edu.ua

Abstract

Salapatov V.I. The information resources using on program code synthesis. *It's proposed the information resources using on the last compilation phase. The new method of the creating program code is designed. This method enables to make the effective program code on the machine–depended level.*

Вступ

Проблема створення ефективних кодів програм як за кількістю пам'яті під код програми, так і за швидкістю його виконання все ще залишається актуальною. В цьому напрямку працюють кілька фірм – виробників програмного забезпечення таких як WatCom, TopSpeed та деякі інші. Згідно інформації, що надходить у відкритий друк, ці фірми створюють найефективніші компілятори. Наводяться технічні характеристики продуктів компіляції, але, на жаль, не розкривається технологія створення ефективних кодів програм. Тому ця стаття є логічним продовженням серії статей [1, 2, 3], що присвячені створенню ефективних кодів програм. Головною метою її є висвітлення формального підходу щодо формування інформаційних ресурсів, які певний час утримуються у регістрах та стеку, з метою їхнього подальшого багаторазового використання у процесі синтезу кодів програм [2]. Процес формування інформаційних ресурсів відбувається одночасно із синтезом коду програм [3], тобто під час синтезу коду одночасно оновлюється база даних (БД) інформаційних ресурсів.

1. Виникнення та використання інформаційних ресурсів

Інформаційні ресурси при компіляції накопичуються при синтезі кодів програм [1]. Спробуємо формалізувати процес обліку та оновлення інформаційних ресурсів.

При синтезі чергової порції коду програми, що, до речі, використовує інформаційні ресурси, треба враховувати розміщення результату операції машинної команди у відповідну змінну. При цьому у БД з інформаційними ресурсами [2] треба занести значення відповідної змінної з її атрибутами, що розміщується у програмно доступних регістрах. Така БД використовується як для запобігання непотрібних команд пересилок або збереження та відновлення при підготовці операндів в разі їхньої наявності в цій БД, так і для обрання найоптимальнішого формату певної команди. Йдеться про зайві команди пересилок у регістри, а також команди збереження певних даних у стеку та відповідні команди відновлення із стеку. Стосовно формату машинної команди розглядається найоптимальніший їхній формат згідно інформації про систему команд ЦЕОМ [3].

2. Постановка задачі по використанню інформаційних ресурсів

Через обмеженість програмно доступних регістрів виникає проблема їхнього заміщення, тобто вибору таких регістрів із наявних і, можливо, повністю зайнятих. Інформація в них або не буде використовуватися, або буде використовуватись рідко у порівнянні з іншими регістрами. Таким чином, стратегія заміщення в нашому випадку повинна визначати, який з регістрів має бути вивантаженим (збереженим) або просто зайнятим, щоб використати його для синтезу коду програми. З цією метою треба спрогнозувати подальше використання тих чи інших даних у конкретній програмі. Тому метою даної роботи є створення методики ефективного використання інформації у регістрах під час синтезу кодів програм.

3. Використання інформаційних ресурсів в процесі синтезу кодів програм

Для отримання ефективних машинних кодів програм необхідно в повній мірі використовувати інформацію, що знаходиться в регістрах. Для цього необхідно в таблиці змінних та констант, що створюються під час синтаксичного аналізу, ввести додаткові атрибути, які враховують місце та спосіб використання змінних та констант. Перший атрибут являє собою порядковий номер оператора початкової програми, а другий має визначати лінійну ділянку програми, цикл або розгалуження. Останні два значення вимагають збереження змінної та подальшого його відновлення.

3.1. Обрання інформаційної структури даних для змінних та констант

З урахуванням наведених вище вимог структура елементу таблиці змінних та констант має вигляд, який показано на малюнку 1 нижче.

Ім'я	Тип	Вид	Знач	Ід_ном	Ід_сп

Мал. 1. Структура елементу таблиці змінних та констант

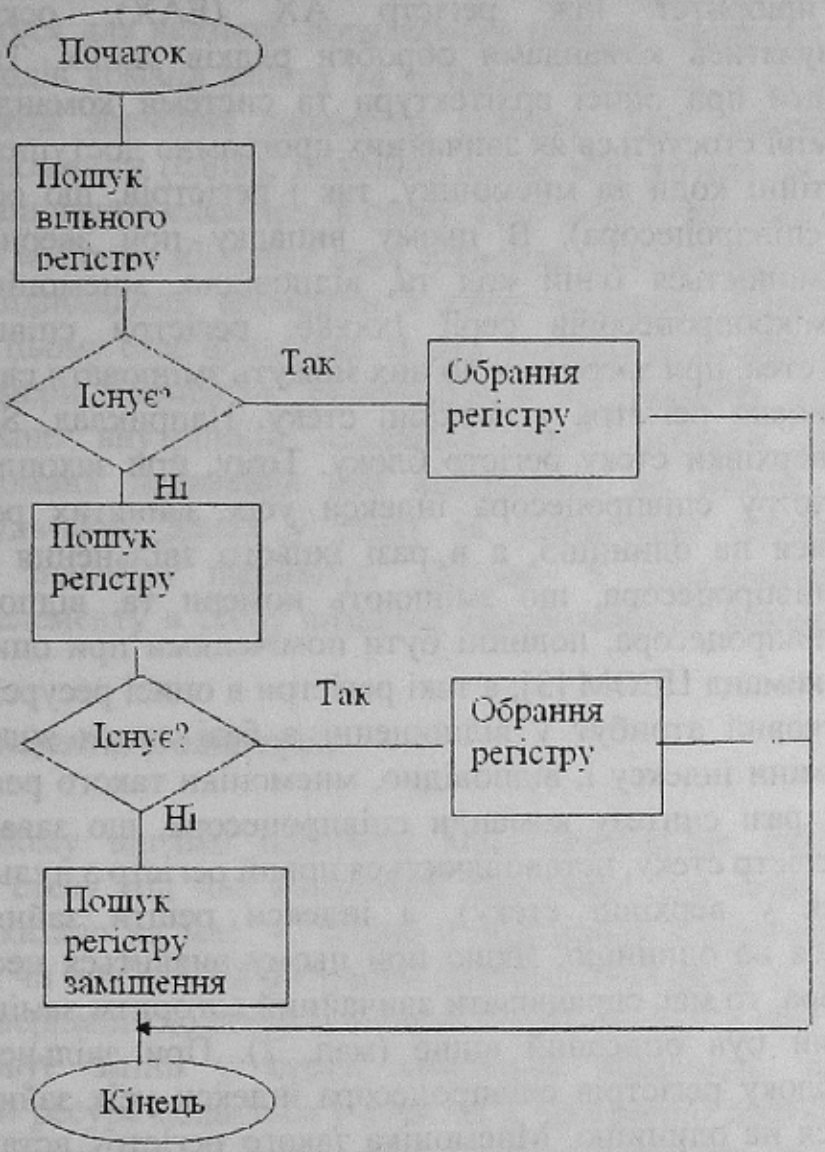
Атрибут «Тип» визначає тип простої змінної, константи або будь-якої структури даних. Атрибут «Вид» визначає вид тієї чи іншої змінної, тобто визначає її як просту змінну, або як масив. Атрибут «Значення» містить безпосереднє значення простої змінної або посилання на неї (адреса), що має місце для рядкових типів, структур даних або масивів. Атрибути «Ід_Номер» та «Ід_Спосіб» використовуються алгоритмом заміщення при синтезі кодів програм і є ключовими для відповідних таблиць, що являють собою списки номерів операторів, де використовується та чи інша змінна або константа, а також спосіб її використання.

3.2. Заміщення програмно доступних регістрів

Алгоритм заміщення програмно доступних регістрів при синтезі коду програми зводиться до перевірки номера оператора, де використовується змінна чи константа, та способу її використання. Якщо номер поточного оператора, для якого виконується синтез коду, перевищує будь-який номер із списку у таблиці з номерами операторів, де використовується та чи інша змінна, то для заміщення обирається програмно доступний регістр з цією змінною. В разі відсутності таких регістрів спочатку здійснюється заміщення для констант, оскільки їхнє значення зберігати не потрібно. В разі заміщення змінних в першу чергу необхідно зберігати адреси змінних (за відповідним типом), а не самі значення. Якщо стоїть вибір, яку саме змінну треба заміщувати у відношенні з інформаційними ресурсами, то для цього треба обирати таку змінну, що має найменшу кількість використань після синтезу коду відносно поточного оператора програми. Укрупнена блок-схема алгоритму заміщення інформаційних ресурсів у регістрах подана на малюнку 2.

Як бачимо, алгоритм заміщення інформаційних ресурсів являє собою звичайний перебір альтернативних можливих варіантів. Причому, порядок перебору визначає пріоритетність дій, які і дозволяють здійснювати синтез

оптимальних кодів програм. Наведені дії повинні виконуватись у тому випадку, коли значення змінної не змінюється деяким іншим оператором від поточного оператора, для якого здійснюється синтез відповідного коду, до тих операторів, що зазначені у таблиці з номерами наступних операторів, де вони використовуються. Тому, при кожному присвоєнні нового значення змінній необхідно коректувати номери операторів у списку, де вони використовуються.



Мал. 2. Блок-схема алгоритму заміщення інформаційних ресурсів

На початку роботи алгоритму в першому лінійному блоці здійснюється пошук вільного регістру. Перший умовний блок на мал. 2 перевіряє наявність такого регістру із списку регістрів і при наявності захоплює його для синтезу низки кодів команд для поточного оператора програми (лінійний блок праворуч). Якщо потрібного регістру немає, то виконується пошук регістрів (лінійний блок нижче), що вже не використовуються в подальшому змінними у регістрах для їхнього заміщення. При наявності такого регістра (другий умовний блок) він

обирається, як і в попередньому випадку, для синтезу низки кодів команд для поточного оператора програми (лінійний блок праворуч). В разі відсутності відповідного регістру виконується пошук такого регістру, що використовується оператором програми з найбільшим номером, тобто подалі від поточного оператора, для якого здійснюється синтез коду. В разі наявності кількох таких регістрів вибір можна здійснювати за їхніми пріоритетами, які можуть визначатися у відповідності за їхнім призначенням. Наприклад, регістри SI та DI (ESI, EDI) Ixxx86 матимуть більший пріоритет ніж регістр AX (EAX), оскільки можуть використовуватись командами обробки рядків даних. Такий пріоритет можна задати при описі архітектури та системи команд ЦЕОМ. Це в однаковій мірі стосується як звичайних програмно доступних регістрів, що мають постійні коди та мнемоніку, так і регістрів, що об'єднані в стек (регістрів співпроцесора). В цьому випадку при зверненні до таких регістрів змінюється їхній код та, відповідно, мнемоніка. Як відомо, стосовно мікропроцесорів серії Ixxx86, регістри співпроцесора, що об'єднані в стек, при зверненні до них можуть змінювати свій номер, тобто індекс, відносно регістра у верхівці стеку. Наприклад, ST(2) вказує на третій від верхівки стеку регістр блоку. Тому, при захопленні чергового нового регістру співпроцесора індекси усіх зайнятих регістрів мають збільшуватися на одиницю, а в разі їхнього звільнення зменшуватися. Команди співпроцесора, що змінюють номери та, відповідно, індекси регістрів співпроцесора, повинні бути поміченими при описі архітектури та системи команд ЦЕОМ [3], а такі регістри в описі ресурсів ЦЕОМ [2, 6] мати додатковий атрибут у відношенні в базі даних значення індексу. Алгоритм зміни індексу і, відповідно, мнемоніки такого регістра є досить простим. В разі синтезу команди співпроцесора, що завантажує дані у поточний регістр стеку, встановлюється новий регістр з нульовим індексом (знаходиться у верхівці стеку), а індекси решти зайнятих регістрів збільшуються на одиницю. Якщо при цьому виявиться нестача регістрів співпроцесора, то має спрацювати звичайний алгоритм заміщення, схожий на той, який був описаний вище (мал. 2). При звільненні чергового регістру з блоку регістрів співпроцесора індекси усіх зайнятих регістрів зменшуються на одиницю. Мнемоніка такого регістру встановлюється за значенням індексу з описі ресурсів із спеціального відношення.

3.3. Обрання інформаційної структури даних для стеку

При заміщенні треба зберігати відповідний регістр, що заміщується, у загальному стеку (команда PUSH) та відновити його зі стеку (команда POP) перед наступним його використанням.

Для ведення обліку даних, що зберігаються в стеку, треба мати спеціальне відношення, структура якого схожа на ту, що представлена

вище на мал. 2. Структура даних елементу такого відношення для обліку даних в стеку представлена на малюнку 3.

Ім'я	Тип	Вид	Знач	Місце

Мал. 3. Структура елементу обліку даних в стеку

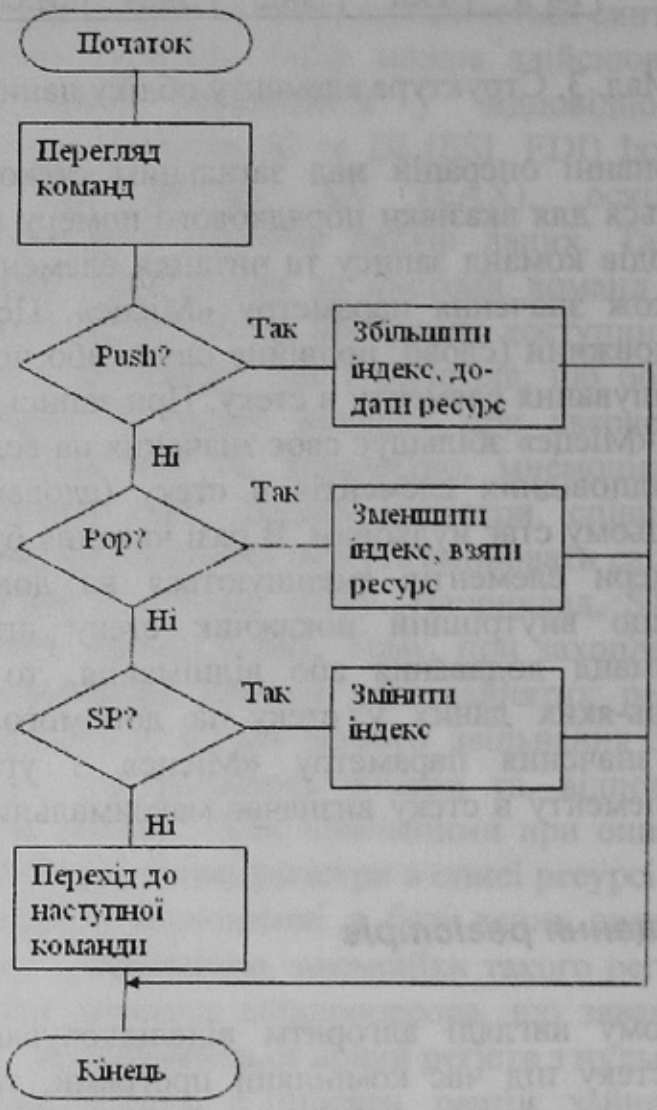
При виконанні операцій над загальним стеком параметр «Місце» використовується для вказівки порядкового номеру елементу в стеку. Під час синтезу кодів команд запису та читання елемента зі стеку необхідно змінювати також значення параметру «Місце». Цей параметр вказує з урахуванням довжини (слово, подвійне слово або чотири чи більше слів) на місце розташування елементу в стеку. При записі чергового елементу в стек параметр «Місце» збільшує своє значення на величину його довжини у одиницях відповідних елементів в стеку (словах). Номер поточного елементу при цьому стає нульовим. В разі читання будь-якого елементу зі стеку усі номери елементів зменшуються на довжину елементу, що читається. Якщо внутрішній покажчик стеку штучно змінювати за допомогою команд додавання або віднімання, то можна забезпечити доступ до будь-яких даних у стеку за допомогою стекових команд. Максимальне значення параметру «Місце» з урахуванням довжини відповідного елементу в стеку визначає максимальний розмір пам'яті під стек.

3.4. Заміщення регістрів

В загальному вигляді алгоритм відслідковування інформації, яка змінюється в стеку під час компіляції програми, так само, як і при її виконанні, показано на мал. 4. Згідно з наведеним алгоритмом спочатку перевіряється, чи пов'язаний поточний оператор програми зі стеком. Якщо виконується звернення до стеку (команди PUSH або POP), то у відношенні, яке відслідковує зміни у стеку, вносяться відповідні зміни. Тобто інформаційний ресурс додається у стек чи у програмно доступний регістр в разі запису туди певної інформації (значення змінної або константи, певної адреси, тощо) або звільняється, якщо цей ресурс замінюється в регістрі або виштовхується зі стеку. При цьому покажчик стеку збільшується або зменшується на відповідний розмір в ньому при запису операнду в стек або його читання зі стеку. В разі штучного зменшення або збільшення покажчика стеку на певний розмір (зміна SP, ESP за допомогою додавання чи віднімання) відбувається лише заміна покажчика стеку.

Треба зазначити, що подібні алгоритми застосовуються при програмній реалізації віртуальної пам'яті в разі заміщення сегментів або

сторінок пам'яті. З точки зору наукової новизни йдеться про розділ програмування, де алгоритми заміщення використовуються. Саме при створенні систем компіляції алгоритми заміщення дають можливість здійснити машинно-залежну оптимізацію кодів програм.



Мал. 4. Блок-схема алгоритму контролю стеку

Запропоновані алгоритми заміщення інформаційних ресурсів у регістрах та стеку дозволяють відслідковувати їхні зміни та місце розташування в апаратних ресурсах ЦЕОМ. Такий облік ресурсів дозволяє багаторазово використовувати інформацію, яка враховується при синтезі кодів програм. При цьому різко зменшується кількість зайвих команд пересилок у регістри, а також команд збереження відновлень інформації, що мають місце при звичайній генерації коду. При синтезі кодів програм кількість таких команд зводиться до мінімуму, оскільки інформація у регістрах буде враховуватись при синтезі кодів програм для конкретної ЦЕОМ в максимальній мірі. Як відомо, саме неефективне використання інформації у регістрах є основною причиною надмірності коду програм при їхній компіляції.

Висновки

За матеріалами виконаної роботи, що викладена в даній статті можна сформулювати такі основні висновки.

1) Запропонована структура елементів змінних і констант програм у вигляді відношення, що формується під час синтаксичного аналізу.
2) Розроблено алгоритм заміщення інформаційних ресурсів, який дозволяє найбільш повно та багаторазово використовувати інформацію в регістрах та стеку. Запропонований алгоритм заміщення ресурсів застосовується як для створення програм користувача (режим користувача), так і для системних програм (привілейований режим).
3) Запропоновано алгоритм відслідковування заповнення інформації в стеку при виконанні програми, за допомогою якого можна визначити максимально необхідний розмір пам'яті під стек.
4) Облік інформаційних ресурсів у спеціальній базі даних та налаштування компіляторів на особливості архітектури та системи команд ЦЕОМ, що знаходяться в спеціальній базі даних, дозволяє отримувати високоефективні коди програм

Література

1. В.И.Салапатов. Синтез кода программ с использованием данных о системе команд целевой ЭВМ и информационных ресурсов. ж. Электронное моделирование № 1. 1998 с. 33-38. 2. В.І.Салапатов. Формування інформаційних ресурсів у процесі синтезу коду. Вісник ЧІТІ. № 2., с. 43-48. Черкаси - 2001. 3. В.И.Салапатов. Структура данных для описания семантики и синтаксиса команд целевой ЭВМ. Вісник національного технічного університету України «Київський політехнічний інститут». Інформатика, управління та обчислювальна техніка. № 41. К. – 2004. с. 191-198. 4. К.Дж.Дейт. Введение в системы баз данных. М.: Вильямс, 2001. – 1072 с. 5. Ал. Ахо, Р. Сети, Дж. Ульман. Компиляторы: принципы, технологии и инструменты. – М.: «Издательский дом», 2001. -768 с. 6. Э. Таненбаум. Архитектура компьютеров. – СПб.: Питер, 2002. - 864 с.

Дата надходження до редакції 12.06.2005 р.