УДК 004.232, 004.274, 004.318

М. О. Березовський, Р. Б. Дунець
Національний університет “Львівська політехніка”,
кафедра спеціалізованих комп’ютерних систем

ОСОБЛИВОСТІ ПРОГРАМУВАННЯ ДИНАМІЧНО РЕКОНФІГУРОВАНИХ ПРОЦЕСОРІВ

Розглянуто проблеми, пов’язані з програмним забезпеченням динамічно реконфігурованих процесорів. Запропоновано способи представлення машинних команд таких процесорів, метод адаптації програм від RISC-процесора, а також методи стиснення машинних команд процесора.

Ключові слова: динамічна реконфігурація, адаптація програмного забезпечення, стиснення машинних команд.

PROGRAMMING FEATURES DYNAMICALLY RECONFIGURABLE PROCESSORS

Considered problems associated with software for the dynamically reconfigurable processors. Suggested ways of representing machine instructions for such processors, the method of adaptation programs of RISC-processor and processor`s machine instructions compression methods.

Key words: dynamic reconfiguration, software adaptation, machine instructions compression.

Вступ

Підвищення продуктивності процесорів універсальних комп’ютерів є актуальною задачею, яка розв’язується декількома шляхами – підвищенням тактової частоти, збільшенням кількості процесорних ядер, введенням додаткових співпроцесорів, а також застосуванням реконфігурації на всіх рівнях. З розвитком мікроелектроніки та забезпеченням достатньої кількості елементів на кристалі стало можливим застосовувати методи реконфігурації, зокрема методи динамічної реконфігурації, які дають змогу налаштовувати апаратуру комп’ютера у процесі його роботи відповідно до операцій програми.

Використання процесорів з реконфігурованими ядрами дає змогу оптимізувати апаратну структуру процесора для виконання конкретної програми, а динамічна реконфігурація таких ядер, здійснювана перед виконанням кожної машинної команди, дає можливість найкраще оптимізувати структуру процесорного ядра для цієї машинної команди.

Аналіз останніх досліджень та публікацій

Сьогодні найпопулярнішими системами машинних команд процесорів є: IA-32 – для персональних комп’ютерів та ноутбуків [1], ARM – для портативних пристроїв, таких як нетбуки, планшети та смартфони [2]. Для процесорів, які мають набори машинних команд, сумісні з цими переліченими системами машинних команд, розроблено безліч компіляторів, які в поєднанні з кросплатформним кодом на мові програмування високого рівня забезпечують переносність програмного забезпечення між різними архітектурами машинних команд [3].

Відповідно, відомі архітектури реконфігурованих процесорів, як правило, передбачають використання ядра процесора однієї з поширених архітектур машинних команд для керівництва реконфігурованими елементами процесора [4]. У разі розроблення процесора з новим набором машинних команд необхідним етапом його впровадження повинно бути створення засобів для програмування під таку архітектуру машинних команд [5]. До таких засобів можуть належати як компілятори, що забезпечують формування машинних кодів з мови високого рівня, так і бінарні транслятори, які адаптують код програм, розроблених для процесорів з іншими наборами машинних команд, для виконання на заданому процесорі [6, 7].

Постановка завдання

Описаний у [8] процесор забезпечує виконання за допомогою функціональних блоків операцій як над регістрами, так і над виходами інших функціональних блоків, наприклад, над результатами попередньої операції, причому послідовність з великої кількості операцій, обмеженої лише кількістю наявних функціональних блоків, можна виконати протягом одного такту однієї машинної команди. Враховуючи відмінності апаратної структури та набору машинних команд процесора від поширених архітектур, актуальним завданням для забезпечення роботи програмно-апаратних засобів, побудованих на цьому процесорі, є адаптація програмного забезпечення для його виконання на динамічно реконфігурованому процесорі.

Способи представлення машинних команд

Машинні команди описаного процесора визначають зв'язки між функціональними блоками, які здійснює матриця зв'язків. На відміну від процесорів, які виконують операції над регістрами, для представлення машинної команди цього процесора неможливо використати класичну форму запису, в якій задають операцію та регістри чи інші операнди, а також приймач результату операції.

Для запропонованого процесора, серед інших, можна використати такі форми представлення машинної команди:

1. Текстовий запис зв'язків між входами та виходами функціональних блоків із розподілом між машинними командами, наприклад у формі запису

x<n> <= y<m>,

де замість <n> записується унікальний ідентифікатор входу, а замість <m> – унікальний ідентифікатор виходу. Такий запис можна використати як аналог мови асемблера звичайних процесорів, оскільки він забезпечує наочний опис команд у текстовому форматі.

2. Табличний запис зв'язків, за якого в кожному стовпці записується значення, що вказує на вихід, під'єднаний до відповідного цьому стовпцю входу, а кожен рядок відповідає окремій команді. Цей запис можна застосувати як аналог представлення машинних кодів звичайних процесорів.

3. Матричний запис зв'язків за допомогою матриці суміжності, за якого рядки відповідають входам, а стовпці – виходам функціональних блоків. Така матриця може містити значення "1", що відповідає наявності зв'язку, та "0" – відсутності зв'язку. Оскільки до одного входу може бути підключено не більше від одного виходу, в кожному стовпці може бути не більше від однієї одиниці. Для запису послідовності команд необхідно використовувати послідовність таких матриць (тривимірний масив). Такий запис може використовуватися для внутрішнього представлення машинних кодів у програмах, таких як компілятори чи дизасемблери.

4. Схематичне представлення функціональних блоків та зв'язків між ними забезпечує наочне представлення сукупності операцій, що виконується машинною командою. Може використовуватися також для графічної побудови чи зміни машинної команди, що найактуальніше для критичних до швидкодії ділянок коду.

Метод адаптації програм RISC-процесора

Одним із методів отримання програм для описаного процесора може бути конвертування наявних програм. Такий підхід забезпечує спрощення переходу на новий процесор без необхідності розроблення нового прикладного програмного забезпечення, за рахунок можливості конвертації наявних програм.

Розроблений метод передбачає використання програм RISC-процесора як вихідного коду, що пояснюється їх простотою, а також перейменуванням регістрів на стадії компіляції, а не під час виконання, як у CISC-процесорів. Конвертування здійснюється в такій послідовності:

1. Отримання структури зв'язків. Машинні команди асемблерного коду діляться на секції в місцях розміщення міток, за якими здійснюються умовні та безумовні переходи. Кожному типу машинної команди RISC-процесора відповідає певна схема під'єднання одного або декількох функціональних блоків певних типів, що виконують відповідну операцію. Наприклад, команда

add <регістр1>, <регістр2>, <регістр3>

відповідає вибору суматора із набору функціональних блоків реконфігурованого процесора та підключення до його входів двох регістрів, а до виходу – третього регістра.

Для кожної секції здійснюється послідовне опрацювання команд одна за одною і для кожної із них з переліку функціональних блоків вибирається необхідний. Вибраний блок, крім блоків регістрів, позначається як зайнятий. Процес триває доти, доки не буде вистачати функціональних блоків. Сформовані у результаті зв'язки між функціональними блоками забезпечують виконання однієї команди реконфігурованого процесора.

У межах множини вибраних функціональних блоків однієї команди реконфігурованого процесора задаємо зв'язки між входами та виходами кожного із блоків та регістрів. Сукупність цих зв'язків для подальшого автоматичного опрацювання подамо у вигляді матриць суміжностей [9].

Розглянемо конкретний приклад. Нехай наш реконфігурований процесор має функціональні блоки з пронумерованими входами та виходами, наведені в табл. 1.

Таблиця 1

Набір функціональних блоків динамічно реконфігурованого процесора

Блок	Тип	Входи	Виходи
$\varphi_1 \ldots \varphi_{32}$	Регістр r1…r32	x1…x32 – значення для запису	y1…y32 – значення регістра
$\varphi_{33} \ldots \varphi_{36}$	Суматор	x33 та x34, x35 та x36, x37 та x38, x39 та x40 – пари доданків для кожного функціонального блока	y33…y36 – сума
φ_{37}	Помножувач	x41, x42 – множник та множене	y37 – добуток
φ_{38}	Інтерфейс оперативної пам'яті	x43 – адреса x44 – дані для запису	y38 – вихід зчитаних даних
φ_{39}	Блок умовного переходу	x45 – умова x46 – адреса переходу за умови = 0 (якщо не підключений – перехід на наступну команду) x47 – адреса переходу за умові ≠ 0 (якщо не підключений – повторення поточної команди)	–

Необхідно конвертувати асемблерний код програми RISC-процесора DLX множення матриць для виконання на цьому процесорі:

```
@matrix_multiply:
  xor  r0, r0, r0
  lw   r1, [m]
  lw   r2, [n]
  lw   r3, [q]
  addi r9, r0, C
  addi r10, r0, A
  addi r4, r1, 0
```

```
@for_r:
  beqz r6, @end_r
    lw   r13, [r7]
    lw   r14, [r8]
    mult r15, r13, r14
    add  r12, r12, r15
    addi r7, r7, 1
    add  r8, r8, r3
```

```
@for_i:
  beqz r4, @end_i
  addi r11, r0, B
  addi r5, r3, 0
@for_j:
  beqz r5, @end_j
  addi r7, r10, 0
    addi r8, r11, 0
    addi r11, r11, 1
    xor  r12, r12, r12
    addi r6, r2, 0
    addi r6, r6, -1
    jr    @for_r
@end_r:
    sw    r12, [r9]
    addi r9, r9, 1
    addi r5, r5, -1
    jr    @for_j
@end_j:
    add  r10, r10, r2
    addi r4, r4, -1
    jr    @for_i
@end_i:
```

Код програми розділяємо на секції відповідно до міток. Далі розглянемо лише секцію, виділену жирним шрифтом. Результат вибору функціональних блоків для нашої секції разом із зв'язками їх входів та виходів зведено у табл. 2. Числові константи 1, -1 та адреси асемблерного коду @for_i та @end_r реалізовуються попереднім записом значень з ОЗП у вільні регістри r16…r19 відповідно під час ініціалізації програми.

Таблиця 2

Отримання структури зв'язків між функціональними блоками

№	Асемблерна команда RISC-процесора	Використані функціональні блоки	Зв`язки входів з виходами функціональних блоків
1	beqz r6, @end_r	φ39	x45 <= y6 x47 <= y19=const(@end_r)
2	lw r13, [r7]	φ38	x43 <= y7 x13 <= y38
3	lw r14, [r8]	φ38	x43 <= y8 x14 <= y38
4	mult r15, r13, r14	φ37	x41 <= y14 x42 <= y13 x15 <= y37
5	add r12, r12, r15	φ33	x33 <= y12 x34 <= y15 x12 <= y33
6	addi r7, r7, 1	φ34	x35 <= y7 x36 <= y16=const(1) x7 <= y34
7	add r8, r8, r3	φ35	x37 <= x8 x38 <= x3 x8 <= y35
8	addi r6, r6, -1	φ36	x39 <= y6 x40 <= y17=const(-1) x6 <= y36
9	jr @for_r	φ39	x45 <= y16=const(1) x47 <= y18=const(@for_i)

З наведеної таблиці можна визначити, що перші дві асемблерні команди будуть реалізовані однією командою реконфігурованого процесора, а решта – другою, оскільки третя асемблерна команда потребує функціонального блока φ_{38}, що вже використаний другою асемблерною командою.

Матриця суміжності для цієї команди реконфігурованого процесора матиме вигляд:

	φ_3	φ_6	φ_7	φ_8	φ_{12}	φ_{13}	φ_{14}	φ_{15}	φ_{16}	φ_{17}	φ_{18}	φ_{33}	φ_{34}	φ_{35}	φ_{36}	φ_{37}	φ_{38}	φ_{39}
φ_3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
φ_6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
φ_7	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
φ_8	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
φ_{12}	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
φ_{13}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
φ_{14}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
φ_{15}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
φ_{16}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
φ_{17}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
φ_{18}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
φ_{33}	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
φ_{34}	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
φ_{35}	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
φ_{36}	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
φ_{37}	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
φ_{38}	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
φ_{39}	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0

У ній рядки позначено функціональними блоками, входи яких з'єднуються із виходами функціональних блоків, якими позначені стовпці матриці. Наявність зв'язку між цими блоками фіксується записаною у відповідній клітинці матриці одиницею.

З метою наочності реалізації другої команди реконфігурованого процесора побудуємо для неї відповідну схему з'єднань між функціональними блоками (див рисунок).

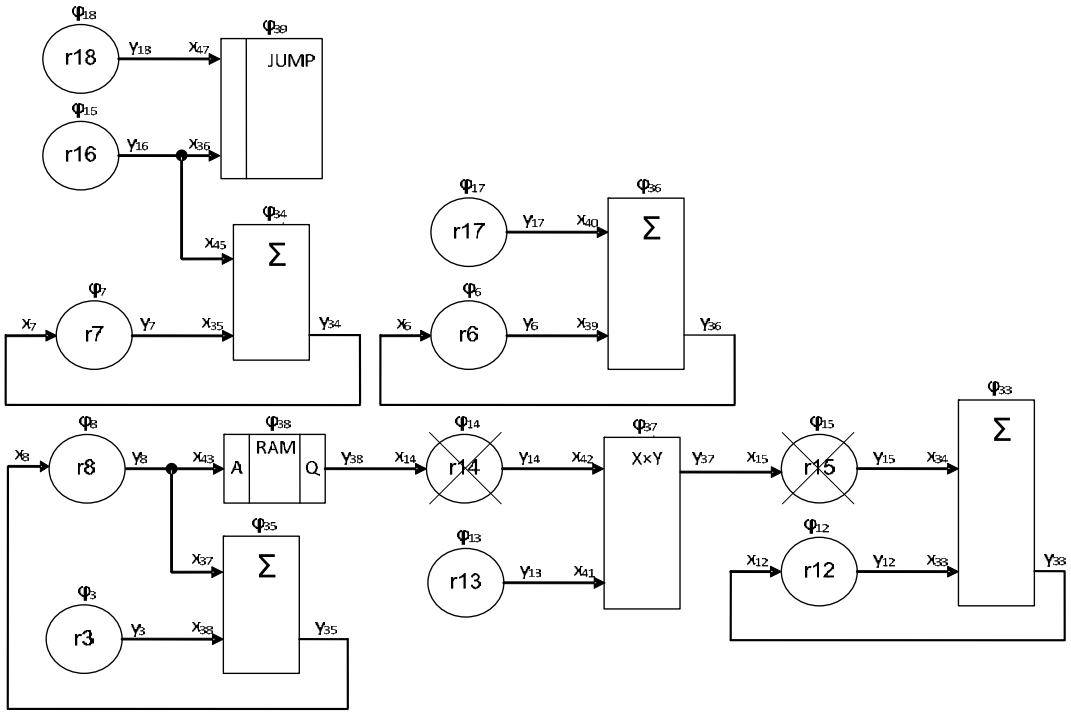

Схема з'єднань між функціональними блоками

2. Мінімізація кількості регістрів. Для кожної команди реконфігурованого процесора проводиться пошук регістрів, що можна вилучити зі схеми з'єднань. Такими регістрами можуть бути ті, що своїми входами і виходами з'єднують два функціональні блоки, які не відносяться до регістрів. Такі регістри виконують функцію додаткової буферизації даних між двома функціональними блоками, а тому ці регістри можна вилучити, з'єднавши функціональні блоки безпосередньо. Всі інші регістри, у яких входи не задіяні, а також ті, що входять до складу циклічних з'єднань, не підлягають вилученню. У розглянутому прикладі це регістри r14, r15 (на рис. 1 вони перекреслені).

Для автоматичного виявлення таких регістрів можна застосувати методи топологічного аналізу, що базуються на матричному представленні зв'язків [9–11].

Спочатку виявимо регістри, входи яких незадіяні. Для цього потрібно відомими методами [9] проаналізувати утворену матрицю суміжності та виявити рядки матриці, у яких є лише нулі, та отримати матрицю-рядок M_I. Елементи цієї матриці поіменовано назвами функціональних блоків-регістрів. Якщо елемент матриці містить одиницю, то відповідний йому регістр є з незадіяним входом. Для нашого прикладу така матриця матиме вигляд:

$$\begin{array}{c} \varphi_3\ \varphi_6\ \varphi_7\ \varphi_8\ \varphi_{12}\ \varphi_{13}\ \varphi_{14}\ \varphi_{15}\ \varphi_{16}\ \varphi_{17}\ \varphi_{18} \\ M_I = \left|1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 1\ \ 1\ \ 1\right| \end{array}.$$

Отже, такими регістрами є φ_3, φ_{13}, φ_{16}, φ_{17}, φ_{18} і їх не можна вилучити. Подальшим кроком є виявлення регістрів, що входять у цикли. Методом [10, 11], що передбачає застосування логічного множення матриць, отримаємо матрицю-рядок M_C, одиничні елементи якої відповідатимуть тим регістрам, що входять у контури. В результаті для нашого прикладу одержимо:

$$\begin{array}{c} \varphi_3\ \varphi_6\ \varphi_7\ \varphi_8\ \varphi_{12}\ \varphi_{13}\ \varphi_{14}\ \varphi_{15}\ \varphi_{16}\ \varphi_{17}\ \varphi_{18} \\ M_C = \left|0\ \ 1\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\right| \end{array}.$$

Отже, регістрами, що входять у контури, є такі: φ_6, φ_7, φ_8, φ_{12}. Щоб виявити регістри, які можна вилучити, потрібно отримати матрицю MD, у якій одиничні елементи відповідатимуть таким регістрам. Для цього виконаємо такі логічні операції над матрицями MI та MC, а саме:

$$M_D = \overline{M_I \vee M_C} =$$

$$= \overline{\left|1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 1\ \ 1\ \ 1\right| \vee \left|0\ \ 1\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\right|} =$$

$$= \overline{\left|1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 1\ \ 1\right|} = \left|0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\right|,$$

$$\begin{array}{c} \varphi_3\ \varphi_6\ \varphi_7\ \varphi_8\ \varphi_{12}\ \varphi_{13}\ \varphi_{14}\ \varphi_{15}\ \varphi_{16}\ \varphi_{17}\ \varphi_{18} \\ M_D = \left|0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\right|. \end{array}$$

Отже, регістри φ_{14}, φ_{15} потрібно вилучити, а в матриці суміжності виконати відповідні дії – вилучити рядки та стовпці, помічені як φ_{14}, φ_{15}, та вказати у ній зв'язки між блоками φ_{38}, φ_{37} та φ_{37}, φ_{33}.

3. Формування розгорнутих машинних команд. Кожному входу відповідає одне поле в машинній команді, що задає номер виходу, до якого під'єднано цей вхід. Вся необхідна інформація міститься в табл. 2 та у відкоригованій матриці суміжності. Аналогічно утворюються всі команди програми реконфігурованого процесора.

Для розглянутого прикладу машинна команда динамічно реконфігурованого процесора, представлена у формі табличного запису, матиме вигляд, як у табл. 3. В полях, номери яких не вказані в таблиці, записується значення 0, що відповідає непідключеному входу.

Таблиця 3

Машинна команда динамічно реконфігурованого процесора

№ поля	8	12	33	34	35	36	37	38	39	40	41	42	43	45	47
Значення	35	33	12	37	7	16	8	3	6	17	13	38	8	16	18

4. Стиснення машинної команди. Цей етап передбачає застосування, залежно від методу стиснення машинних команд для конкретного процесора, відповідного методу стиснення.

5. Методи стиснення машинних команд

Можуть застосовуватись такі методи стиснення машинної команди.

1. Табличний метод з фіксованою довжиною машинної команди, який полягає у попередньому завантаженні наборів декодованих машинних команд у буферну пам'ять, наприклад, за допомогою спеціальної машинної команди, і подальшому виборі їх машинною командою, значення якої містить адресу в цій пам'яті.

Розрядність машинної команди, крім команд завантаження машинних команд у буфер, у разі застосування такого методу можна визначити виразом

$$R' = \log_2 \mathrm{S} + \mathrm{C},$$

де R′ – розрядність оптимізованої машинної команди, бітів; S – ємність буферної пам'яті, що відповідає кількості машинних команд, які можуть одночасно зберігатися в ній; C – розрядність додаткових полів; якщо результат логарифма не є цілим числом, він заокруглюється до більшого цілого значення.

Перевагами такого методу є фіксована невелика розрядність машинної команди та простота апаратної реалізації. До недоліків можна зарахувати значну розрядність для операцій з невеликою кількістю задіяних функціональних блоків та необхідність попереднього завантаження декодованих команд у буферну пам'ять.

2. Табличний метод з префіксним кодуванням. Для зменшення розрядності часто використовуваних машинних команд можна застосувати алгоритми префіксного кодування, наприклад. Розрядність оптимізованої машинної команди змінна і залежить як від застосованого алгоритму кодування, так і від розподілу ймовірностей появи різних команд у програмі. На відміну від попереднього, цей метод вимагає складнішої апаратури та більших витрат часу, необхідних для декодування програми.

3. Використання формату з пропуском незадіяних входів, що є подальшим удосконаленням описаного автором в [12] методу. В такому випадку машинна команда складається з полів двох типів – полів, що задають збільшення номера поточного входу, та полів, що задають номер виходу, який під'єднується до поточного входу. Тип поля визначається однобітним префіксом на початку поля. Якщо між полями, що задають номери виходів, немає поля збільшення номера входу, поточним стає наступний вхід. Для визначення кінця машинної команди може використовуватися як спеціальне значення для останнього поля, так і поле з довжиною команди в бітах чи полях перед іншими полями машинної команди. Розрядність машинної команди для цього методу визначатиметься виразом

$$R' = n_1 \times (\log_2 \mathrm{m} + 1) + \mathrm{n}_2 \times (\log_2 (i - 1) + 1) + \mathrm{r} + \mathrm{C},$$

де R′ – розрядність оптимізованої машинної команди, бітів; m – кількість виходів; n_1 – кількість задіяних входів; n_2 – кількість полів, що відповідають проміжкам у нумерації входів; i – максимальне збільшення номера входу за одне поле, яке задає збільшення номера поточного входу; r – розмір поля, яке позначає кінець або задає розмір машинної команди; C – розрядність додаткових полів; результати логарифма заокруглюються до більшого цілого значення.

Для зменшення середньої довжини машинної команди входи та виходи функціональних блоків слід нумерувати в послідовності зменшення ймовірності їх використання, а також групувати входи з попередніми, якщо вони використовуються лише разом, наприклад, входи арифметичних та логічних пристроїв, присвоюючи групі лише один номер входу, який відповідає першому з входів у групі. Переваги цього методу – невелика розрядність машинних команд типових операцій та пропорційна до складності машинної команди довжина. Основним недоліком є значна апаратна і часова складність, необхідна для декодування машинної команди.

Конкретний метод оптимізації розрядності машинних команд вибирають під час розроблення процесора на основі критерію найменшої середньої довжини машинної команди, враховуючи наявні обмеження на апаратну та часову складність. При цьому також враховують особливості апаратної реалізації дешифратора команд, який здійснює їх декомпресію під час роботи програми.

Висновки

У роботі запропоновано способи представлення машинних команд динамічно реконфігурованих процесорів, метод адаптації програм RISC-процесора та методи стиснення машинних команд такого процесора.

Розроблені методи адаптації програмного забезпечення та стиснення машинних команд уможливлюють застосування динамічно реконфігурованих процесорів як універсальних обчислювачів для високопродуктивних комп'ютерних систем.

1. Intel 64 and IA-32 Architecture Optimization Reference Manual (March 2014) [Електронний ресурс]. – Режим доступу: http://www.intel.ru/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf 2. Таненбаум Э. Архитектура компьютера / Э. Таненбаум, Т. Остин. 6-е изд. – СПб.: Питер, 2013. – 816 с. 3. Bringing Portability to the Software Process / West Virginia University, Dept. of Statistics and Computer Science – Morgantown, 1997 – 9 p. 4. Клятченко Я. М. Soft-процесорний пристрій на базі сучасних ПЛІС для реалізації алгоритму адаптивного порівняння інформаційних об'єктів / Я. М. Клятченко, В. П. Тарасенко, О. В. Тарасенко-Клятченко, О. К. Тесленко // Вісник Нац. ун-ту "Львівська політехніка": Комп'ютерні системи та мережі. – 2011. – № 717. − С. 64–69. 5. Штейнберг Б. Новым процессорам – новые компиляторы / Б. Штейнберг, М. Юрушкин // Открытые системы: журнал. – 2013, № 01. 6. G. Haber. Introduction to Binary Translation [Електронний ресурс] // Intel, 2010, – Режим доступу: http://moodle.technion.ac.il/plug infile.php/373059/mod_resource/content/3/ IntroductionToBinary Translation-ver2. pdf. 7. Baraz L. IA-32 Execution Layer: a two-phase dynamic translator designed to support IA-32 applications on Itanium-based systems / Baraz, Leonid; Devor, Tevi; Etzion, Orna. – Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture MICRO 36 – Washington, DC, USA: IEEE Computer Society, 2003. 8. Березовський М. Динамічно реконфігуроване процесорне ядро / М. Березовський, Р. Дунець // Сучасні комп'ютерні системи та мережі: розробка та використання: матер. 5-ї міжнар. наук.-техн. конф. ACSN-2011. – С. 204–205. 9. Дунець Р. Б. Аналіз та синтез топологій комп'ютерних видавничо-поліграфічних систем: монографія. – Львів: НВФ "Українські технології", 2003. – 192 с. 10. Dunets R. Method of determination the elements of connected simple cycles of topologies // Proc. Advanced Computer Systems and Networks: Design and Application (ACSN-2003). – Lviv: Publishing House of Lviv Polytechnic National University, 2003. – P. 131–133. 11. Дунець Р.Б. Визначення часу та маршрутів критичних шляхів топологій спеціалізованих комп'ютерних систем // Вісник Хмельницького нац. ун-ту. – Хмельницький, 2007. – Т.1. – № 2. – С. 70–74. 12. Березовський М. Оптимізація формату машинних команд динамічно реконфігурованого процесорного ядра / М. Березовський, Р. Дунець // Радіоелектронні і комп'ютерні системи. – 2012. – № 5 (57). – С. 120–124.