УДК 004.335

Г.Е. Маргиев, магистр,
А.В. Тищенко, студент,
А.Н. Мирошкин, канд. техн. наук, доц.,
Донецкий национальный технический университет, г. Красноармейск
margievge@gmail.com, tishka.ms95@gmail.com, miroshkinan@gmail.com

Исследование реализации ассоциативных запоминающих устройств

В статье рассмотрены вопросы внутренней организации ассоциативных запоминающих устройств (АЗУ), поиск в которых реализуется по алгоритму бинарного дерева и хеш-таблицы. Рассмотрены аспекты программной реализации АЗУ, длительности инициализации таких устройств, длительности процесса поиска и вероятности коллизий. Исследования проводились при решении задачи поиска ключевых слов в тексте. На основе исследований выработаны характеристики основных типов АЗУ по критериям скорости работы, нагрузки на ЦП и пикового объема оперативной памяти.

Ключевые слова: ассоциативное запоминающее устройство, хеш-таблица, бинарное дерево, коллизия.

Введение

Стремительное развитие информационных технологий обусловлено универсальностью функциональных возможностей ЭВМ и проникновением ЭВМ во все сферы человеческой деятельности. При этом постоянно растущие скорость вычислений и объем информации, хранящейся в ЭВМ, не могут в полной мере удовлетворить потребности практики применения информационных технологий. Тем более, возможности современной элементной базы и классической архитектуры аппаратных средств ЭВМ по быстродействию и производительности приближаются к физическим пределам. Выход за эти пределы возможен только при реализации новых идей в сфере аппаратно-программных средств и применения новых технологий, в частности, путем разработки методов и средств искусственного интеллекта, сочетающие в себе формальные подходы с эвристическими приемами, присущими человеку, основанными на принципах адаптации к окружающим условиям. Анализ показывает, что решение интеллектуальных задач основывается на специфических методах обработки огромного объема информации в условиях неполной или нечетко заданной информации. Попытки применения классических методов обработки данных в этих условиях требуют все большего повышения быстродействия процессора и емкости памяти ЭВМ. Методы ассоциативной организации памяти позволяют в определенной степени решать эти проблемы, поскольку в этом случае на элемент памяти возлагаются также и некоторые функции обработки информации [1-6].

Ассоциативный способ основан на установлении некоторого соответствия, ассоциации между хранящейся в запоминающем устройства (ЗУ), и поисковыми аргументами. Запоминающие устройства, в которых использован такой способ доступа, получили название ассоциативных ЗУ (АЗУ).

АЗУ характеризуются двумя основными признаками:

- доступ к хранимой информации в АЗУ осуществляется исходя из ее содержания;
- при каждом обращении к АЗУ возможен параллельный доступ ко всей хранящейся информации.

Эти два признака определяют свойство «интеллектуальности» АЗУ. Первый из них определяет возможность обработки данных непосредственно в логико-запоминающем среде АЗУ, а второй обеспечивает возможность параллельной обработки информации.

Виды АЗУ

Существует два основных способа адресации по содержанию. Первый из них базируется на распределении памяти в зависимости от содержания данных и реализуется с помощью программных средств. Второй способ адресации опирается на применение специальных аппаратных средств, предназначенных для хранения и поиска элементов данных. Следует отметить, что оба метода начали разрабатываться практически одновременно в середине 50-х гг. Учитывая это, можно сделать вывод, что потребность в создании памяти с адресацией по содержанию стала ощущаться уже тогда, когда только появились первые вычислительные системы коммерческого назначения. Хотя с тех пор прошло уже много лет, указанные методы продолжают развиваться параллельно, и пока нет оснований для окончательного выбора в пользу одного из них. Поэтому здесь мы обсудим оба принципа построения ассоциативной памяти в их современной трактовке.

Аппаратные реализации АЗУ

Память с адресацией по содержанию имеет три основные базовые типы построения:

- АЗУ параллельного типа;
- АЗУ с поиском, параллельным по словам и последовательным по разрядам;
- АЗУ с поиском, последовательным по словам и параллельным по разрядам.

Параллельный способ организации АЗУ позволяет достигать высокой производительности, при которой время доступа к памяти не превышает нескольких наносекунд. На этом основании АЗУ параллельного действия, как правило, используются в качестве быстродействующих буферных ЗУ.

На рис. 1 изображена функциональная схема одной ячейки АЗП параллельного действия. Все логические цепи, представленные на рис. 1 построены на обычных вентилях И-НЕ. Исключение сопоставляют только вентили, подключении к линиям M_i і $\overline{B}_j$, которые реализуют функцию «Встроенное И». Такая реализация схемы сравнения и обеспечивает наибольшее быстродействие за счет отсутствия последовательных цепей итеративного сравнения, благодаря чему каждая ячейка обрабатывается параллельно независимо друг от друга, а общий результат формируется по выходным потенциалам ячеек.

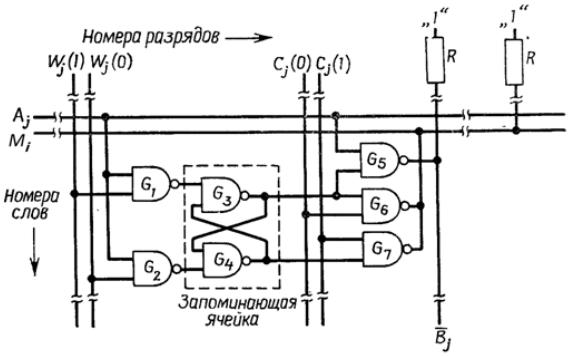

Рисунок 1 - Одноразрядная ячейка АЗУ параллельного действия с цепью сравнения слов на основе функции «Встроенное И»

На рис. 2 приведена блок-схема общей реализации АЗУ данного типа.

АЗУ второго типа по своей структуре проще, чем предыдущие, что позволяет использовать сравнительно недорогие элементы памяти. Данная структура организации имеет достаточно широкий круг применения, например, в системе проектирования XILINX в набор модулей CORE Generator входит IP-Core Content Addressable Memory (CAM). Один из вариантов реализации генерирует CAM-модуль на основе памяти на сдвиговых регистрах. Блок-схема структуры такой памяти приведена на рис. 3.

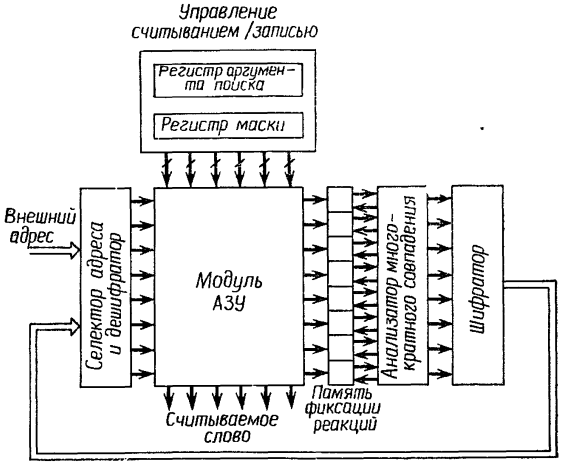

Рисунок 2 - Организация АЗУ параллельного действия

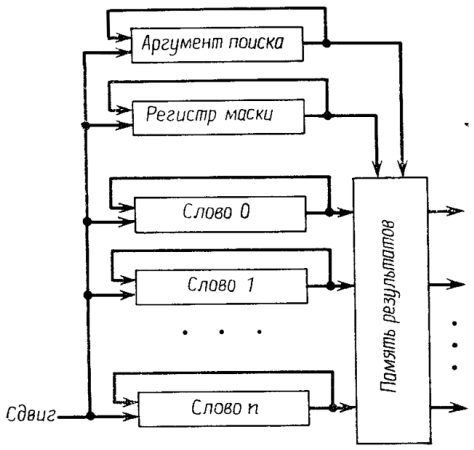

Рисунок 3 - АЗУ на регистрах сдвига

В качестве ячеек памяти в таком CAM-модуле применяются регистры сдвига. Поисковый аргумент и слово маскирования хранятся в отдельных регистрах, аналогичных по строению ячейкам памяти. На все ячейки, а также на регистры аргумента и маскирования постоянно подаются последовательность тактовых импульсов. В каждом такте содержание памяти синхронно циклически сдвигается на разряд вправо; при этом значение самого правого разряда всех ячеек поступает в память результатов, они сравниваются с соответствующим битом маскируемого аргумента поиска. Память результатов в данной схеме необходима, так как операции сравнения выполняются с отдельными разрядами, и конечный результат рассчитывается рекурсивно.

В АЗУ описанного типа разряды слов, которые хранятся в памяти, могут нумероваться как слева направо, так и наоборот. Порядок нумерации определяется выбранным алгоритмом сравнения. К недостаткам данной схемы относится невозможность временного прекращения цирку-

ляции содержания некоторых ячеек и получения заданной комбинации битов.

АЗУ последнего типа реализуют последовательный поиск информации в памяти. Такой подход может использоваться в тех случаях, когда время необходимое для получения результатов от АЗУ сравнимо с продолжительностью других вычислительных операций. Преимущество данной методики заключается в том, что все операции реализованы аппаратно, и не требуют использования каких-либо программных средств. На рис. 4 приведена одна из упрощенных структур организации АЗП с последовательным поиском по словам и параллельным по разрядам.

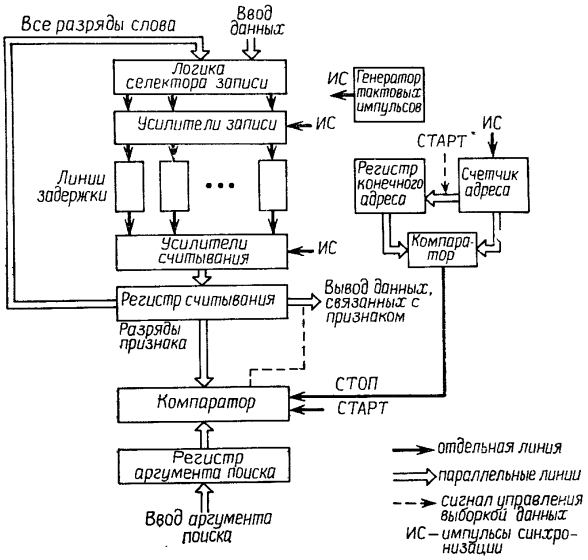

Рисунок 4 - Структура АЗУ с поиском, последовательным по словам и параллельным по разрядам

Для хранения информации в системе используется набор линий задержки, по которым синхронно циркулируют серии импульсов. Процесс циркуляции управляется специальными приемными и передающими цепями, подключенными к концам линий. В принципе динамическую память можно реализовать на регистрах сдвига в интегральном исполнении, однако стоимость такого ЗУ будет слишком высокой.

Программные реализации АЗУ

Существует два основных подхода к программной реализации АЗУ:

- АЗУ на основе древовидных структур поиска;
- АЗУ на основе таблиц хеширования.

Применение дерева поиска в качестве основы для построения АЗУ обусловлено тем, что поиск по дереву является более эффективным алгоритмом, чем линейный алгоритм поиска. К достоинствам древовидных структур можно отнести упорядоченность хранимой информации, что облегчает организацию поиска минимального и максимального элемента, а также поиск со сравнением величин. К недостаткам данных алгоритмов можно отнести зависимость времени поиска от позиции в цепочке узлов: чем дальше расположен узел, тем дольше время выполнения алгоритма. Для выравнивания времени поиска по ветвям разрабатываются алгоритмы балансировки дерева, целью которых является в общем случае перераспределение цепочек узлов таким образом, чтобы они имели минимальные длины.

Хеширование является самым быстродействующим из известных методов программного поиска; эта его черта особенно проявляется при работе с наборами данных большого размера. Данный метод достаточно удобен тем, что не требует ни какого-либо упорядочения, ни сортировки ключевых слов.

Высокая скорость выполнения операций хеширования обусловлена тем, что элементы данных запоминаются, а затем выбираются из ячеек памяти, адреса которых являются простыми арифметическими функциями от содержания соответствующих ключевых слов. В современных ЭВМ вычисления подобных функций занимает очень малое время по сравнению с продолжительностью доступа к памяти (особенно в памяти большой емкости).

Набор всех допустимых ключевых слов принято называть пространством имен, а набор адресов памяти, в которые превращаются ключевые слова, – адресным пространством. Адреса, получаемые из ключевых слов методом хеширования, называются хеш-адресом. В общем случае нельзя гарантировать, что новые адреса будут отличаться от уже задействованных, так как ключевые слова выбираются произвольно и, кроме того, невыгодно хранить список всех ранее использованных адресов. Поэтому всегда существует отличная от нуля вероятность того, что два различных ключевых слова будут преобразованы в один и тот же адрес в памяти. Подобная ситуация называется коллизией (или конфликтом) имен. При коллизии для элемента, который претендует на уже занятую ячейку памяти, отводится новое место в памяти, расположение которого легко получить, зная адрес точки коллизии. Однако в любом случае при наличии одной или нескольких резервных ячеек с одинаковым хеш-адресом необходимо иметь возможность идентифицировать элементы, которые туда попали по ключевым словам, так как неизвестно, как эти элементы распределены по ячейкам. Обычно для этого вместе с данными записывается соответствующее ключевое олово (или результат его однозначного преобразования), а на этапе выборки оно сравнивается с ключом, используемым в качестве аргумента поиска (рис. 5).

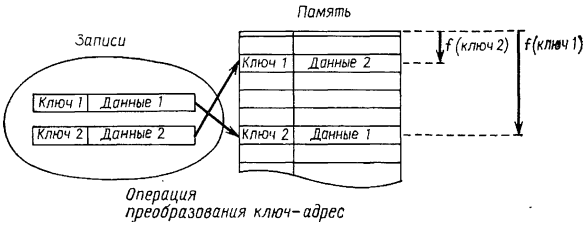

Рисунок 5 - Распределение записей при хешировании (формирование хеш-таблицы)

Если расположение элемента данных нужно установить исходя из нескольких различных ключей, исчисляется нужное количество независимых хеш-адресов, и по каждому из них записывается его отдельная копия. Чтобы избежать чрезмерного расхода памяти на запись многочисленных копий одного элемента в хэш-таблице он заменяется более коротким идентификатором, например, адресной ссылкой-указателем на область памяти, где хранится сам элемент. С их помощью можно упростить манипуляции с данными, не прибегая к их фактической выборке.

Наиболее широкое распространение методы хеширования получили в системном программировании. Хеш-таблицы стали применяться в ассемблерах фирмы IBM начиная примерно с 1954 г., а в настоящее время практически все символьные таблицы Ассемблеров и компиляторов строятся по принципу хеширования. Это снижает затраты на проведение вычислений, объем которых в значительной мере определяется продолжительностью этапа компиляции. Современные персональные компьютерные построены по архитектуре фон Неймана, а старые ограничения в виде малого объема памяти и отсутствия возможностей организации параллелизма обработки уже преодолены. Это делает целесообразным проведение исследований в направлении разработки алгоритмов на основе программных подходов. Исходя из вышесказанного, целью исследования стала разработка программных реализаций АЗУ и их исследование в процессе решения тестовой задачи поиска ключевого слова по тексту.

Выбор алгоритмов реализаций

В качестве базовой иерархической структуры выбрано бинарное дерево поиска. Двоичное (или бинарное) дерево поиска - двоичное дерево, в котором каждой вершине x сопоставлено определенное значение $val[x]$. При этом такие значения должны удовлетворять условию упорядоченности:

- пусть x – произвольная вершина двоичного дерева поиска, если вершина y находится в левом поддереве вершины x, то val [y] $\leq$ val [x];
- если y находится в правом поддереве x, то val [y] $\geq$ val [x].

Бинарные деревья поиска гораздо эффективнее в операциях поиска, чем линейные структуры, в которых затраты времени на поиск пропорциональны $O(n)$, где n – размер массива данных, тогда как в полном бинарном дереве это время пропорционален в среднем $O(log_2 n)$ или $O(h)$, где h - высота дерева.

Хеш-таблица – структура данных, которая реализует интерфейс ассоциативного массива данных, а именно, она позволяет хранить пары (ключ, значение) и осуществлять три операции:

- операцию добавления новой пары;
- операцию поиска;
- операцию удаления с ключом.

Существует два основных варианта хэш-таблиц: с цепочками и с открытой адресацией. Хеш-таблица содержит в себе некоторый массив H, элементами которого являются пары (хеш-таблица с открытой адресацией) или списки пар (хеш-таблица с цепочками) [7, 8].

Выполнение операций в хэш-таблице начинается с вычисления хэш-функции от ключа. Полученное хеш-значение $i = hash(key)$ играет роль индекса в массиве H. После этого операция (добавление, удаление, поиск) перенаправляется объекту $H[i]$, который хранится в соответствующей ячейке массива.

Однако, велика вероятность коллизии. Например, при добавлении в хеш-таблицу размером 365 ячеек, всего лишь 23-х элементов, вероятность коллизии уже превышает 50 процентов (если каждый элемент может с одинаковой вероятностью попасть в любую ячейку) – см. парадокс дней рождения [9]. Поэтому механизм разрешения коллизий – важная часть любой хеш-таблицы.

В некоторых особых случаях удается избежать коллизий. Например, если все ключи элементов известны заранее (или очень редко меняются), то для них можно найти некоторую совершенную хеш-функцию, которая распределит их по ячейкам хеш-таблицы без коллизий. Хеш-таблицы, которые используют подобные функции, не требуют механизма разрешения коллизий, и называются хеш-таблицами с прямой адресацией.

В данной работе для исследования свойств алгоритмов реализации АЗУ программными методами выбрана задача поиска ключевого слова в тексте. Так как уникальность входных данных в этом случае достаточно низкая, для борьбы с коллизиями выбран метод цепочек.

Исследование алгоритмов

Описанные алгоритмы были разработаны на языке C++ в среде разработки Visual Studio Community 2015. Перед тестированием определим параметры входного текста с помощью онлайн калькулятора [10]. В результате получены следующие характеристики:

- общее количество слов – 35705;
- число уникальных слов – 9942.

Скорость инициализации

Кроме скорости поиска очень важно знать скорость подготовки алгоритма, поскольку в условиях реального использования задержки данного этапа могут привести к невозможности использования алгоритма, если общее время выполнения перевисит максимально допустимый предел. Особенно в программировании системных функций этот недостаток может приводить к нарушениям работы всего программного обеспечения.

Время инициализации для бинарного дерева при проведении эксперимента составило 57 мс, для таблицы хеширования – 28 мс. По данному показателю лидирует алгоритм на основе хеш-таблицы.

Скорость поиска

Этот тест выполним дифференциальным образом, а именно на первом этапе выполним поиск слова в начале текста, на втором этапе выполним поиск слова, которое находится в конце текста. Это позволит оценить, имеет ли зависимость время доступа от положения в буфере, и если имеет то насколько большую.

Этап 1. Выполняем поиск самого первого слова в книге – «Артур». Следует заметить, что алгоритмы повторно вызываются 100000 раз, следовательно, реальное время одной операции T_{OP} определяется по формуле.

$$T_{OP} = \frac{T_t}{100000} \text{ мс},$$

где T_t – время поиска, определенное программно.

Среднее время работы одной операции:

- алгоритм бинарное дерево - 3 ·10-8 с;
- алгоритм хеш-таблица - 3 ·10-8 с.

На первом этапе виден паритет результатов.

Этап 2. Проверим, как изменятся результаты, если слово будет находиться в конце текста. В качестве такого слова использован «Примечания» так как оно встречается всего один раз и находится практически в конце тестового фрагмента текста. Среднее время работы одной операции:

- алгоритм бинарное дерево - 0,0019 мс;
- алгоритм хеш-таблица - 0,0001 мс.

На втором этапе прослеживается зависимость времени поиска алгоритма бинарного дерева от позиции в тексте. Поэтому лидером в данном тесте является алгоритм хеш-таблицы.

Использование системных ресурсов

Реализация программного обеспечения была выполнена в среде Visual Studio Community 2015. Данный инструментарий предоставляет также возможность проводить профилирование разработанного ПО, создавать отчеты, и сравнивать их между собой.

На рис. 6 отражены графики нагрузки на центральный процессор при выполнении теста, где видна пиковая нагрузка на центральный процессор (ЦП). Наибольшую нагрузку на ЦП, как ожидалось, дает алгоритм хеш-таблица.

Результаты экспериментов позволяют утверждать, что по нагрузке на ЦП лучшие показатели имеет ассоциативный алгоритм бинарного дерева.

По критерию пикового количества задействованной оперативной памяти алгоритм на основе бинарного дерева показал значение 4,1 Мб, а алгоритм на основе хеш-таблицы 3,7 Мб.

Заключение

В результате проведенного исследования получены следующие характеристики методов программной организации АЗП: по нагрузке на ЦП лидером является алгоритм бинарного дерева; по нагрузке на системную память лидером алгоритм хеш-таблицы; по общей скорости выполнения лидирует алгоритм хеш-таблица.

Авторы предполагают, что аппаратная реализация АЗУ с поиском по алгоритму бинарного дерева и по алгоритму хеш-таблицы будут иметь схожие относительные характеристики, а абсолютные будут зависеть от характеристик микросхемы АЗУ или базиса, в котором будет такое запоминающее устройство реализовано. Дальнейшие исследования направлены на исследования различных аппаратных реализаций ассоциативных запоминающих устройств.

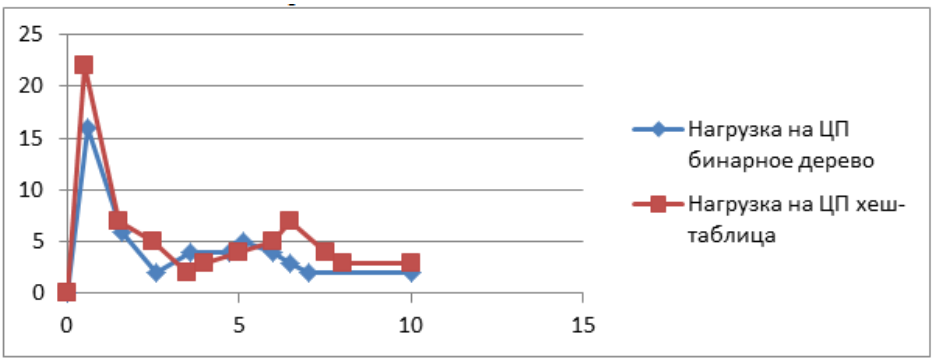

Рисунок 6 - Результат тестирования ассоциативных алгоритмов по критерию нагрузки на ЦП

Список использованной литературы

1. Кохонен Т. Ассоциативные запоминающие устройства: пер. с англ. / Т. Кохонен. – М.: Мир, 1982. – 384 с.
2. Кохонен Т. Ассоциативная память пер. с англ. / Т. Кохонен. – М.: Мир, 1980. – 239 с.
3. Огнев И.В. Интеллектуальные системы ассоциативной памяти / И.В. Огнев, В.В. Борисов. – М.: Радио и связь, 1996. – 176 с.
4. Радченко А.Н. Ассоциативная память. Нейронные сети. Оптимизация нейропроцессоров / А.Н. Радченко. – СПБ.: Наука, 1998. – 261 с.
5. Огнев И.В. Проектирование систем ассоциативной памяти современных ЭВМ / И.В. Огнев, В.В. Борисов. – М.: МЭИ, 1997. – 70 с.
6. Соломатин В.Ф. Теория ассоциативных запоминающих устройств с распределенной записью информации / В.Ф. Соломатин // Автометрия. – 1982. – № 1. – 21-34.
7. Вирт Н. Алгоритмы и структуры данных пер. с англ. / Н. Вирт. – М.: Мир, 1989. – 360 с.
8. Керниган Б. Язык программирования Си: пер. с англ. / Б. Керниган, Д. Ритчи; под ред. и с предисл. Вс. Штаркмана. - 2-е изд., перераб. и доп. – М.: Финансы и статистика, 1992. – 272 с.
9. Парадокс днів народження [Електронний ресурс]. – Режим доступу до електронного джерела: https://uk.wikipedia.org/wiki/Парадокс_днів_народження.
10. Он-лайн калькулятор: Підрахунок однакових слів [Електронний ресурс]. – Режим доступу до електронного джерела: http://planetcalc.ru/3205/

Надійшла до редакції 20.01.2016

Г.Е. МАРГІЄВ, Г.В. ТИЩЕНКО, О.М. МІРОШКІН
Донецький національний технічний університет

ДОСЛІДЖННЯ РЕАЛІЗАЦІЇ АСОЦІАТИВНИХ ЗАПАМ'ЯТОВУЮЧИХ ПРИСТРОЇВ

У статті розглянуті питання внутрішньої організації асоціативних запам'ятовуючих пристроїв (АЗП), пошук в яких реалізується за алгоритмом бінарного дерева і хеш-таблиці. Розглянуто аспекти програмної реалізації АЗП, тривалості ініціалізації таких пристроїв, тривалості процесу пошуку і ймовірності колізій. Дослідження проводилися при вирішенні завдання пошуку ключових слів у тексті. На основі досліджень визначені характеристики основних типів АЗП за критеріями швидкості роботи, навантаження на ЦП і пікового обсягу використаної оперативної пам'яті.

Ключові слова: асоціативний запам'ятовуючий пристрій, хеш-таблиця, бінарне дерево, колізія.

H.E. MARHIIEV, H.V. TYSHCHENKO, O.M. MIROSHKIN
Donetsk National Technical University

RESEARCH OF ASSOCIATIVE MEMORY DEVICES IMPLEMENTATION

Solving of present-day computation problems requires permanent improvement in speed and performance of data search and operation processes. Classical approaches to storage devices development do not allow to operate data fast enough. New approaches to the organization of such storage devices are necessary, one of which is by addressing information using content instead of address. Given article is devoted to actual scientific and technical task of associative memory devices implementation. The features of the software and hardware realization of parallel content memory devices and those, that use shift registers, are described. The advantages and disadvantages of each type of given types of content memory devices are identified. Two basic types of software implementation of associative memory devices were investigated: a search algorithm for a binary tree and using a hash table. It is shown that each approach has its own characteristics that must be considered when choosing the type of device. Special attention was paid to the start initialization of data structures, which requires a certain time and memory resources. Also the duration of the search process and the probability of collisions in two types of software implementations of associative memory devices were investigated. The experimental task for performing the research was to find a keyword in the test piece of text. It is shown that the duration of the search using a binary tree algorithm affects the depth of the desired position relative to the root word. Based on these studies identified characteristics of the main types of associative memory devices criteria speed, the load on the CPU and RAM peak. Further studies are aimed at determining the compliance characteristics of storage devices implemented software and hardware.

Keywords: associative memory device, hash table, binary tree, collision.