УДК 004.31(075.8) DOI: 10.34759/trd-2020-115-14

Устройство планирования загрузки процессоров в мультипроцессорных системах критического назначения

Борзов Д.Б.[1*], Басов Р.Г.[1], Титов В.С.[1***], Соколова Ю.В.[2****]**

[1]Юго-Западный государственный университет, ЮЗГУ,

ул. 50 лет Октября, 94, Курск, 305040, Россия

[2]Научно-производственное объединение им. С.А. Лавочкина,

ул. Ленинградская, 24, Химки, Московская область, 141400, Россия

**e-mail: borzovdb@kursknet.ru*

***e-mail: r-basov@eureca.ru*

****e-mail: titov-kstu@rambler.ru*

*****e-mail: jv.sokolova@mail.ru*

Статья поступила 10.11.2020

Аннотация

Статья посвящена мультипроцессорным системам критического назначения, к которым можно отнести объекты высокой готовности, работоспособность которых критична для любого рода деятельности. К ней можно отнести человека, страну, предприятие, организацию и т.д. Отказ в этом случае приводит к уменьшению времени реакции системы и дальнейшему снижению ее производительности, а значит коэффициента готовности. В этом случае использование программных средств решения данного вопроса неприемлемо, а значит необходимо применение специализированных аппаратных средств планирования загрузки.

Ключевые слова: мультипроцессорная критическая система, планирование загрузки процессоров, высокая готовность, векторная мультипроцессорная система, алгоритм планирования.

Постановка задачи

В современных мультипроцессорных критических системах возникает необходимость оперативной реакции со стороны вычислительной системы. Под критическими системами понимают системы, отказы которых приводят к потерям (экономическим, физическим, человеческим и т.п.). В случае отказа к таким системам предъявляются высокие требования к работоспособности, безотказности, безопасности, защищенности и т.д. При этом, расходы, связанные с внесением изменений в систему или с ее заменой (прямые, косвенные и т.д.) важнее потерь в случае прямого или косвенного отказа самой мультипроцессорной системы. Очевидно, что наиболее важным является минимизация времени и аппаратных затрат, необходимых для реакции мультипроцессорной системы на внештатную ситуацию.

К критическим ситуациям, возникающим в мультипроцессорных системах можно отнести отказы внутренних процессорных модулей, таких как кабины пилотов самолетов, систем наблюдения, слежения, прицеливания, атомные системы и т.п. При этом в случае отказа мультипроцессорной системы и уменьшается ее производительности и снижается быстродействие, чего в критических системах допускать нельзя. Одним из вариантов решения данного вопроса может быть планирование загрузки процессоров в мультипроцессорных системах. В этом случае можно избежать одновременной загрузки нескольких процессоров одной задачей (программой, подпрограммой, алгоритмом, файлом и т.п.) и, вместе с тем, запланировать очередь поступающих задач таким образом, чтобы они подавались одновременно. Это позволяет снизить незапланированные простои в работе и

одновременно повысить коэффициент ее готовности вместе с повышением быстродействия.

В связи с этим, в статье предлагается метод, алгоритм и устройство планирования загрузки процессоров в мультипроцессорных системах критического назначения, обеспечивающего повышение производительности мультипроцессорных систем и увеличения их коэффициента готовности.

Введение

Для мультипроцессорной системы [1-3] из P процессоров определены вычисления из $X=\{x_1,x_2,\ldots,x_j,\ldots,x_n\}$ заданий, заданные некоторым алгоритмом, сведенного в вектор сложности $W=\|w_i\|_n$, где $N=n=|x|$, $i\in X$. При этом задача i приходят в случайный момент времени и их количество на входе в очереди $\le Q$, где Q – суммарный объем заданий (биты, байты, килобайты и т.д.), находящихся в очереди, а сами задачи могут приходить в случайный момент времени. Задания поступают на процессор в момент времени $r_i\ge 0$ и задаются заранее значениями w_i.

Для мультипроцессорной системы заранее определено, что процессоры одинаковы, т.е. обладают одинаковыми тактовыми частотами; производительностью и архитектурой, не прерываются на выполнение других задач, то есть не имеют приоритетов и равнозначны (равноправны) [5]. Мультипроцессорная система задается множеством Pr, которое представлено как: $\mathrm{Pr}=\{P_1,P_2,\ldots,P_\beta,\ldots,P_\lambda\}$, где P_β – процессоры мультипроцессорной системы, причем ($P_\beta=\overline{1,\lambda}$) [6-7].

Таким образом, необходимо построить расписание выполнения для заданий с учетом всех приведенных ограничений. То есть необходимо для каждой задачи

$x_i \in X$ определить момент начала выполнения S_j такого, что $S_j \geq r_j$ так что $S_j + \mathrm{w_i} \leq S_j$, где S_i, S_j – моменты начала выполнения других заданий, причем $i,j \in X$, $i \neq j$. Аналитически поиск расписания загрузки процессоров может быть описан отображением [8-9]:

$$\alpha_q = \left\{ x_{q_1}, x_{q_2}, \ldots, x_{q_j}, \ldots, x_{q_n} \right\} \rightarrow \left\{ p_{q_1}, p_{q_2}, \ldots, p_{q_\lambda} \right\}, \quad (1)$$

где $q = 1, \ldots, \mathrm{n}$, символ «$\rightarrow$» показывает операцию планирования заданий множества X на соответствующее множество процессоров Pr.

Здесь q – это номер очередного варианта назначения, соответствующий q-у варианту назначения. Мощность множества $\Psi = \alpha_q$ всевозможных отображений (1) равна числу всевозможных назначений заданий $\left\{ x_{q_1} \right\}$ в матрице $X : |\Psi| - N!$.

На основе формализованной постановки задачи ее соответствующая математическая постановка выглядит следующим образом [2,3]:

$$\sum_{j=1}^{n} r_j \rightarrow \min$$
$$\alpha_q = \left\{ x_{q_1}, x_{q_2}, \ldots, x_{q_j}, \ldots, x_{q_n} \right\} \rightarrow \left\{ p_{q_1}, p_{q_2}, \ldots, p_{q_\lambda} \right\}. \quad (2)$$

С учетом (1–2) и исходя из представленных выше теоретических предположений в работе предложен метод конвейерного планирования загрузки процессоров в мультипроцессорных критических системах, основанный на (1-2). Фрагменты программ предлагается назначать независимо на процессоры мультипроцессорной системы без учета других операторов. Предлагаемый метод состоит из следующих этапов [8-10]:

1. Получение множества заданий $X=\{x_1, x_2, ..., x_j, \ldots x_n\}$, требующих составления плана загрузки процессоров, множества процессоров $Pr=\{P_1,P_2,\ldots,P_\beta,\ldots,P_\lambda\}$ мультипроцессорной системы и запись в виде матрицы $PrX=\|p_\beta, x_j\|$.

2. Получение в строке и столбце $\Pr X_{\beta\alpha}$ минимального элемента и вычитание его из всех остальных элементов.

3.Поиск в строке $\Pr X_{\beta\alpha}$ нуля вычеркивание остальных нулей в строке.

4. Поиск в столбце $\Pr X_{\beta\alpha}$ нуля вычеркивание остальных нулей в столбце.

5. Повтор п. 3-4 для всех элементов $\Pr X_{\beta\alpha}$.

6. Проведение минимального количества горизонтальных и вертикальных пересечений через отмеченные нули.

7. Поиск минимума среди не зачеркнутых прямыми чисел и вычитание его из этих чисел.

8. Прибавление минимального числа к числам, стоящим на пересечении прямых.

9. Повторение п. 2-8 пока в $\Pr X_{\beta\alpha}$ не найден ноль.

10. Если $\Pr X_{\beta\alpha} = 0$, то подпрограмма *i* назначается на процессор *j*.

11. Повторение п. 1–11 до нахождения остальных назначений подпрограмм на процессоры мультипроцессорной системы.

12. Получение тэга готовности *Exec*.

13. Анализ битов доступности поля *Work* тэга *Exec*. Если $Work = 1$, то прочитать номер свободного процессора и п.1, иначе п. 4.

Общая длительность всех операций вычисляется по формуле:

$$T = \sum_{j=1}^{n} r_j \to \min. \qquad (3)$$

Для решения задачи составления плана загрузки процессоров предлагается алгоритм, позволяющий снизить время планирования параллельного выполнения подпрограмм на процессорах мультипроцессорной системы, исключающий связи по управлению и данным между ними по критериям (1-3).

На основе предложенного метода был разработан алгоритм планирования загрузки процессоров в мультипроцессорных системах, состоящий из следующих шагов [11].

1. Задать $\Pr X_{\beta\alpha}$ - матрицу заданий, где $\alpha = \overline{1, P}$ – количество заданий мультипроцессорной системы, а $\beta = \overline{1, P}$ – количество процессоров.

2. Поиск в строке $\min\limits_{j}^{P} \Pr X_{\beta\alpha}$ и вычитание его из всех ее элементов.

3. Определение в столбце $\min\limits_{i}^{P} \Pr X_{\beta,\alpha}$ и вычисление из всех ее элементов.

4. Поиск в строке минимального элемента и его фиксация. Если есть еще нули в строке, то вычеркивание их.

5. П. 4 выполнить для всех строк $\Pr X_{\beta\alpha}$.

6. Поиск в столбце ноля и его фиксация. Если есть еще нули в столбце, то вычеркнуть их.

7. П. 6 выполнить для всех $\Pr X_{\beta\alpha}$.

8. Если в $\Pr X_{\beta\alpha}$ нет нулей, то выделение минимальное количество горизонтальных и вертикальных пересечений через отмеченные нули и п.9, иначе п. 2-7.

9. Среди не зачеркнутых прямыми чисел поиск минимального значения и вычитание его из этих чисел. Прибавление минимального числа к числам, стоящим на пересечении прямых.

10. Повторять п. 2-13 пока в $\Pr X_{\beta\alpha}$ не появится один ноль.

11. Если $\Pr X_{\beta\alpha} = 0$, то подпрограмма *i* назначается на процессор *j*.

12. Повторять п. 1–11 до нахождения остальных назначений подпрограмм на процессоры мультипроцессорной системы.

В соответствии с предложенным методом, в алгоритме на первом шаге задается исходная матрица $\Pr X_{\beta\alpha}$, где α – строки, задающие множество заданий, для которых необходимо найти расписание загрузки процессоров, а столбцы – множество процессоров соответственно.

На втором шаге выполняется поиск минимального элемента в каждой строке исходной матрицы, после чего на третьем шаге происходит вычитание найденного элемента из всех элементов соответствующей строки. Третий шаг выполняет аналогичные действия со столбцами матрицы. На четвертом шаге находим в строке минимальное значение и фиксируем его. В случае наличия в строке аналогичных нулей, необходимо все их вычеркнуть.

Если на данном шаге в матрице присутствует количество нулей, равное начальному количеству подпрограмм, то поиск завершен. В этом случае каждый ноль в матрице означает: строка показывает номер подпрограммы, а столбец – процессор, на который назначается эта подпрограмма.

Если количество нулей не равно числу подпрограмм, то перейти к шагу 9. На данном этапе необходимо провести минимальное число прямых, проходящих через все нули таблицы. На девятом шаге в полученной таблице выполняется поиск минимального числа, не проходящего ни через одну прямую. Далее вычитаем найденный минимум из всех чисел, через которые не проходит ни одна прямая, и добавляем его ко всем элементам, лежащим на пересечении двух прямых. Если после выполнения п. 9 не появилось числа нулей, равного числу подпрограмм, то повторять п. 2–10, пока не выполнится данное требование.

Результаты и их обсуждение

В соответствии с представленной математической постановкой задачи, задания для мультипроцессорной системы из P процессоров сводятся в вектор сложности $W = \|w_i\|_n$. Как следует из предложенного метода и алгоритма планирования загрузки процессоров, фактически для работы процессора используется две операции сложения и вычитания, исполняемых по классической векторной схеме. В свою очередь это приводит к возможности применения векторного процессора, структурная схема которого представлена на рисунке 1 [1,6-7,12-15].

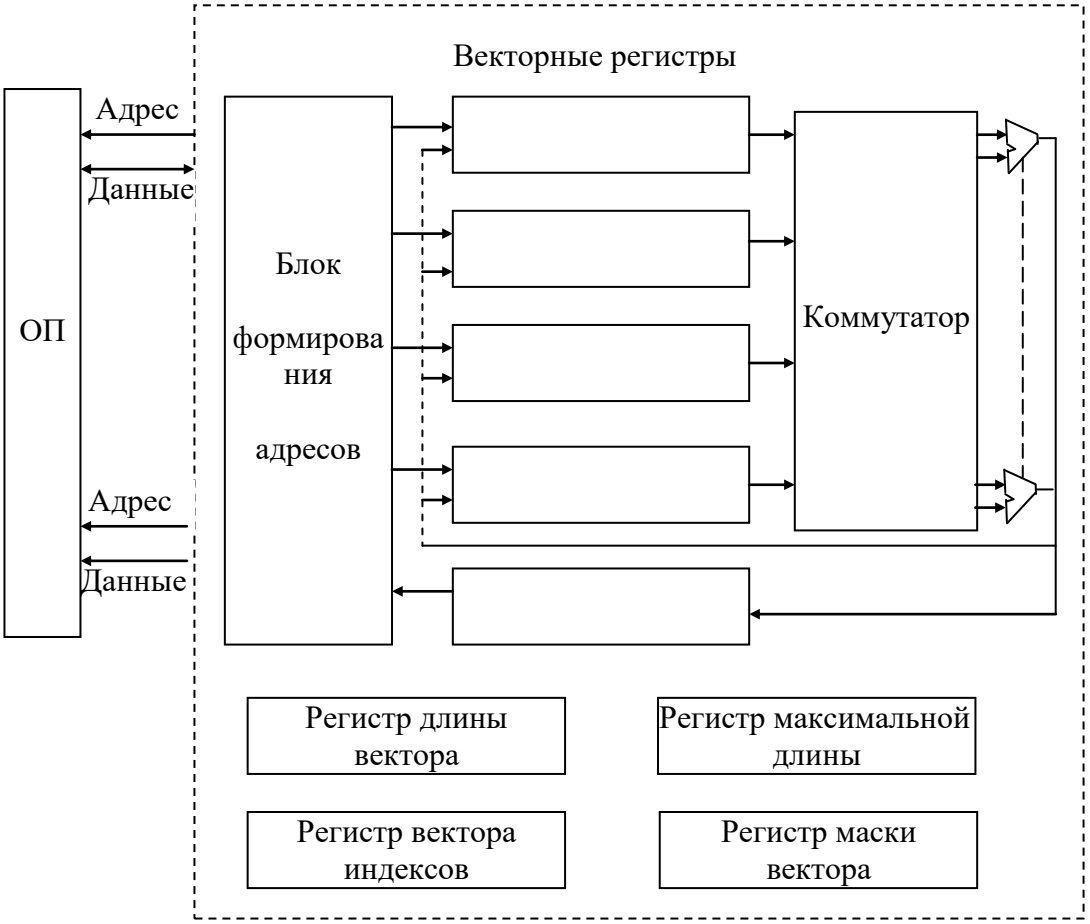


Рис. 1. Структурная схема векторного процессора

Обработка всех n компонентов векторов-операндов задается одной векторной командой (рис. 1). Распространена структура, когда АЛУ состоит из отдельных блоков сложения и умножения. Например, иногда блок для вычисления обратной величины операции деления $\frac{X}{Y}$ реализуется в виде $X\left(\frac{1}{Y}\right)$. Каждый из таких блоков также конвейеризирован. Кроме того, в состав векторной вычислительной системы обычно включают и скалярный процессор, что позволяет параллельно выполнять векторные и скалярные команды.

В случае конвейерной обработки очередной элемент вектора подается на вход конвейера, как только освобождается первая ступень так, как это показано на рисунке 2.

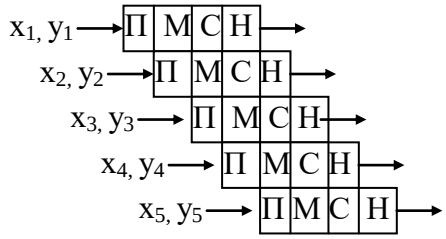


Рис. 2. Обработка векторов в конвейерном АЛУ

Как видно из рисунка 2, данный вариант пригоден для обработки векторов.

В [16] считается, что типовое вычислительное устройство выглядит так, как показано на рисунке 3.

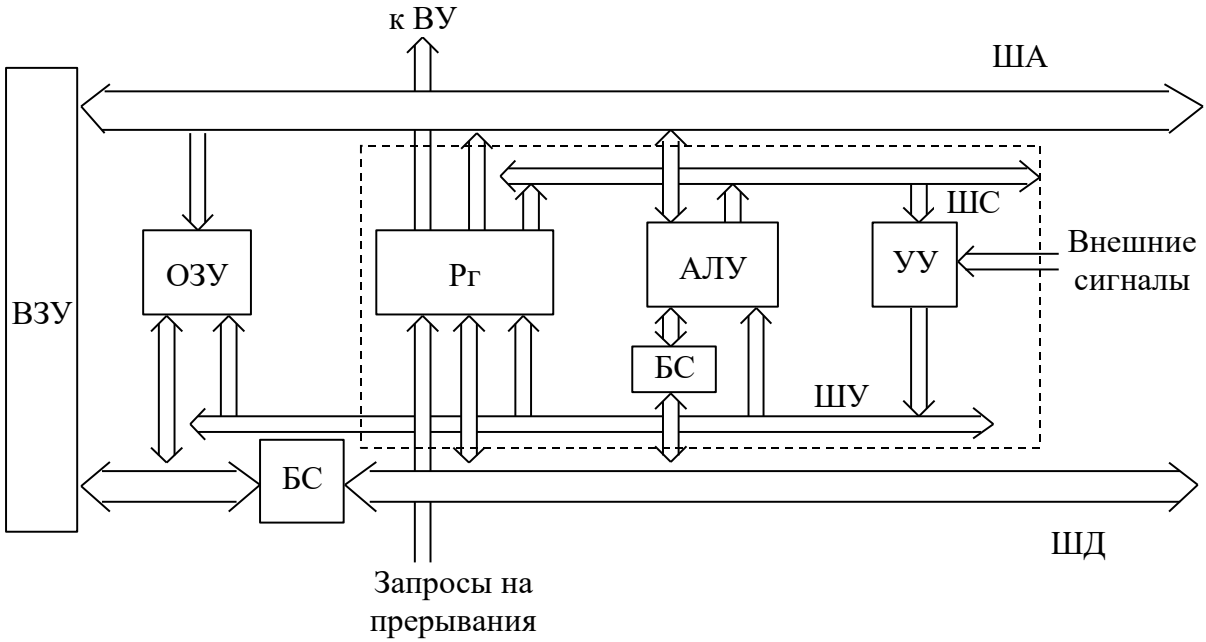


Рис. 3. Структурная схема вычислительно устройства: ОЗУ – оперативное запоминающее устройство, Рг - блок регистров и логических схем, АЛУ - арифметико-логическое устройство (устройство управления АЛУ не показано), УУ - устройство управления, БС - блок согласования разрядности шин, ША - шина адреса, ШД - шина данных, ШС - шина состояния, ШУ - шина управления, ВУ - внешние устройства

Процесс управления исполнением команд представляется в виде двухступенчатой иерархической структуры управляющих автоматов, состоящей из

ведущего устройства управления процессором и ведомого (подчиненного) автомата устройства управления АЛУ (УУ АЛУ) [16] (рис. 4).

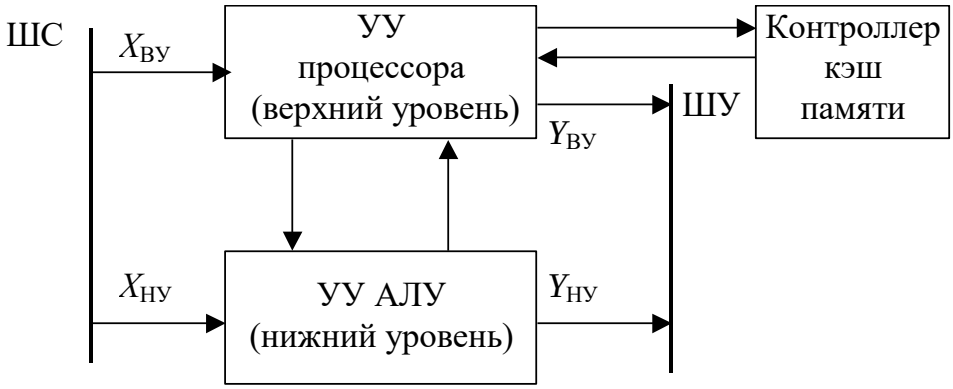


Рис. 4. Двухступенчатая структура процесса управления

Как следует из рисунков 3 и 4 для реализации функций векторного процессора следует определить разрядности основных элементов верхнего уровня процессора, к которым относятся шины адреса (ША), шины данных (ШД), ОЗУ, шины управления (ШУ) и элементов нижнего уровня процессора: операционный автомат АЛУ и устройства управления (УУ) АЛУ.

Введем предварительные ограничения:

– задания представляются числовым эквивалентом разрядностью не больше 8 бит;

– количество заданий в очереди, ожидающих обработки не более 100;

– количество процессоров равно четырем:

– количество очередей, ожидающих выполнения в векторном процессоре равно шести.

Необходимо реализовать следующие операции:

– сложение;

– вычитание;

– условный переход по значению >0;

– условный переход по значению <0.

Используемый формат команды «регистр–регистр».

Формирование исполнительного адреса операнда должно быть организовано по прямому способу адресации.

Для определения разрядностей ША, ЩД и ШУ воспользуемся исходными данными для проектирования векторного процессора и получаем:

– так как разрядность передаваемых данных не более восьми, то выбираем **ШД(7-0)**;

– так как очередей, ожидающих выполнения равно шести и предполагается использование векторного процессора, то получаем **ША(7-0)**.

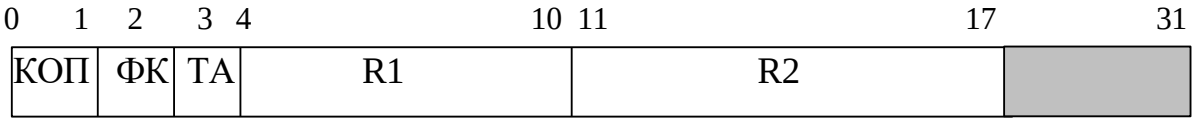
Та как в векторном процессоре предполагается использовать четыре команды, то их кодирование необходимо два двоичных разряда, а значит получаем разрядность поля кода операции **КОП(1-0)**. Получаем таблицу кодов команд (табл. 1).

Таблица 1

Кодирование команд

Тип операции	Код
Сложение	00
Вычитание	01
условный переход по значению <0	10
условный переход по значению >0	11

Получаем формат команды:



На основании разрядности полей формата команды получаем разрядность **ШУ(31-0)**.

На основе полученных разрядностей форматов команды получаем содержательную таблицу кодирования вариантов форматов команд (табл. 2).

Таблица 2

Кодирование форматов команд

Код операции	Формат команды	Тип адресации (ТА)	Код ТА	Содержание операции
Сложение 00	RR	П	00	AH.AL := POH[R_1] + POH[R_2]
Вычитание 00	RR	П	01	AH.AL := POH[R_1] + POH[R_2]
УП по > 0 10	RR	П	10	if (SF = 0) and (ZF = 0) then PC := R_1
УП по < 0 10	RR	П	10	if SF = 1 then PC := R_2

На основе содержательной таблицы команд 2 проектируем обобщенный алгоритм командного цикла (рис. 5).

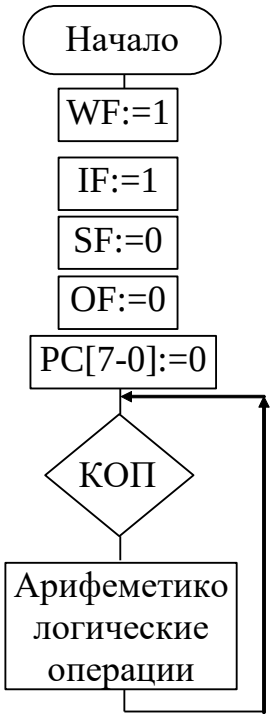


Рис. 5. Блок -схема обобщенного алгоритма командного цикла

В алгоритме, представленном на рисунке 3первоначально устанавливаются флаги процессора: флаг работы устройства WF устанавливается в единицу (работа разрешена), флаг прерывания IF=1 (прерывания разрешены), флаг знака равен нулю (SF=0), флаг переполнения равен нулю (OF=0).

Работа алгоритмов векторного процессора представлена на рисунке 6-7.

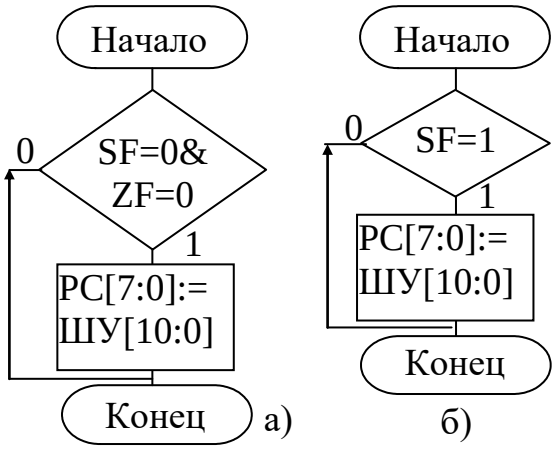


Рис. 6. Алгоритм перехода по значению больше нуля (а) и меньше нуля (б)

Алгоритмы операций перехода выполняются в зависимости от состояния соответствующих флагов. Управление на новый адрес передаётся путём загрузки программного счётчика значением с шины управления (из кода команды).

Алгоритмы команды сложения и вычитания приведены на рисунке 7.

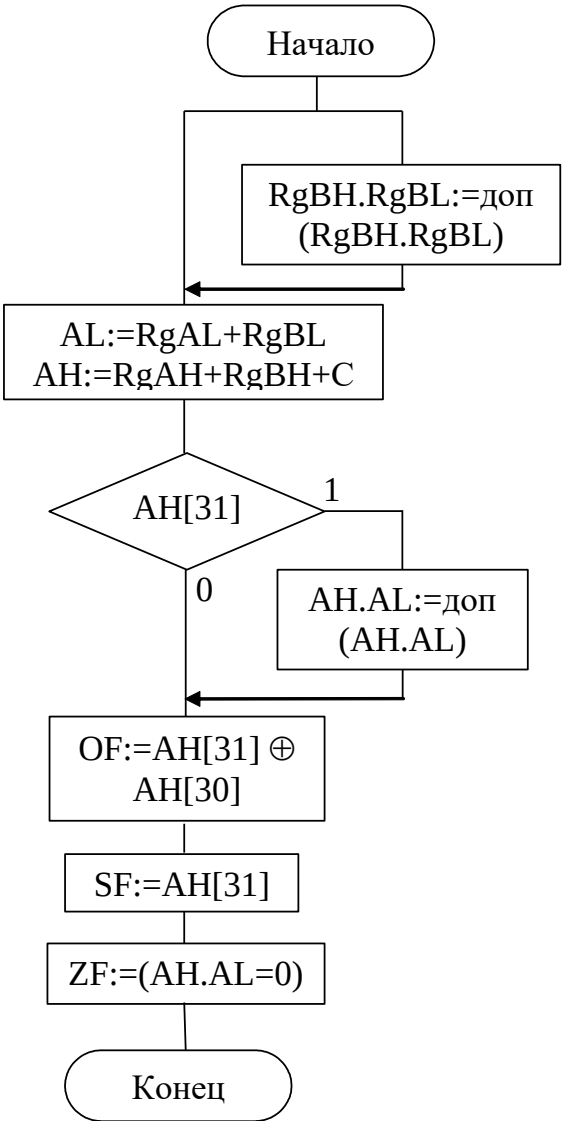


Рис. 7. Алгоритмы команд сложения и вычитания

Операции сложения и вычитания в алгоритме (рис. 7) выполняются одинаково. Отличие состоит в том, что при вычитании операндов, хранящийся в регистровой паре RgBH.RgBL, преобразуется в дополнительный код. Затем происходит сложение содержимого регистров RgAL и RgBL с помещением результата в AL и установкой

при переполнении флага переноса C. Сложение содержимых RgAH и RgBH происходит с учетом этого флага.

Если после выполнения операции получился отрицательный результат, нужно преобразовать его, проинвертировав и прибавив единицу. Флаг переполнения равен сумме по модулю 2 знакового и предшествующего ему разрядов результата. Флаг знака – знаковый разряд результата. Флаг нуля – признак равенства результата нулю.

При построении операционного автомата АЛУ необходимо вначале на основании полученных алгоритмов составить список микроопераций МПА ведомого УУ [3]. Обозначим множество его входных сигналов (логических условий) через $X = \{x_1, x_2, ..., x_l\}$, а множество его выходных (управляющих) сигналов – через $Y = \{y_1, y_2, ..., y_m\}$. Множество состояний управляющего автомата обозначим через $S = \{s_1, s_2, ..., s_n\}$. Список соответствующих микроопераций представлен в таблице 3.

Таблица 3

Список микроопераций

Выходные сигналы УУ	Микрооперация
y0	RgBH.RgBL:=доп(RgBH.RgBL)
y1	AL:=RgAL+RgBL
y2	AH:=RgAH+RgBH+c
y3	AH.AL:=доп(AH.AL)
y4	SF:=AH[31]
y5	ZF:=(AH.AL=0)
y6	OF:=AH[31] ⊕ AH[30]
y7	PC[13:0]:=ШУ[35:22]
x1	КОп = ”УП>0”
x2	КОп = ”УП<0”

x3	КОп = ”УП=0”
x4	SF=1
x5	SF=0 &ZF=0
x6	АН[31]=1

После разработки обобщенного алгоритма АЛУ, кодирования микроопераций и логических условий строится функциональная схема операционного автомата АЛУ. Управляющие сигналы $\{y_i\}$ расставляются в соответствии с разметкой присваиваний в алгоритме АЛУ (табл. 3). Выходные сигналы $\{x_i\}$ формируются с учетом разметки и используются в качестве входных сигналов (логических условий) МПА ведомого УУ АЛУ, который разрабатывается на основании этого же обобщенного размеченного алгоритма АЛУ с учетом обозначенных на рис. 8 его состояний $\{S_i\}$.

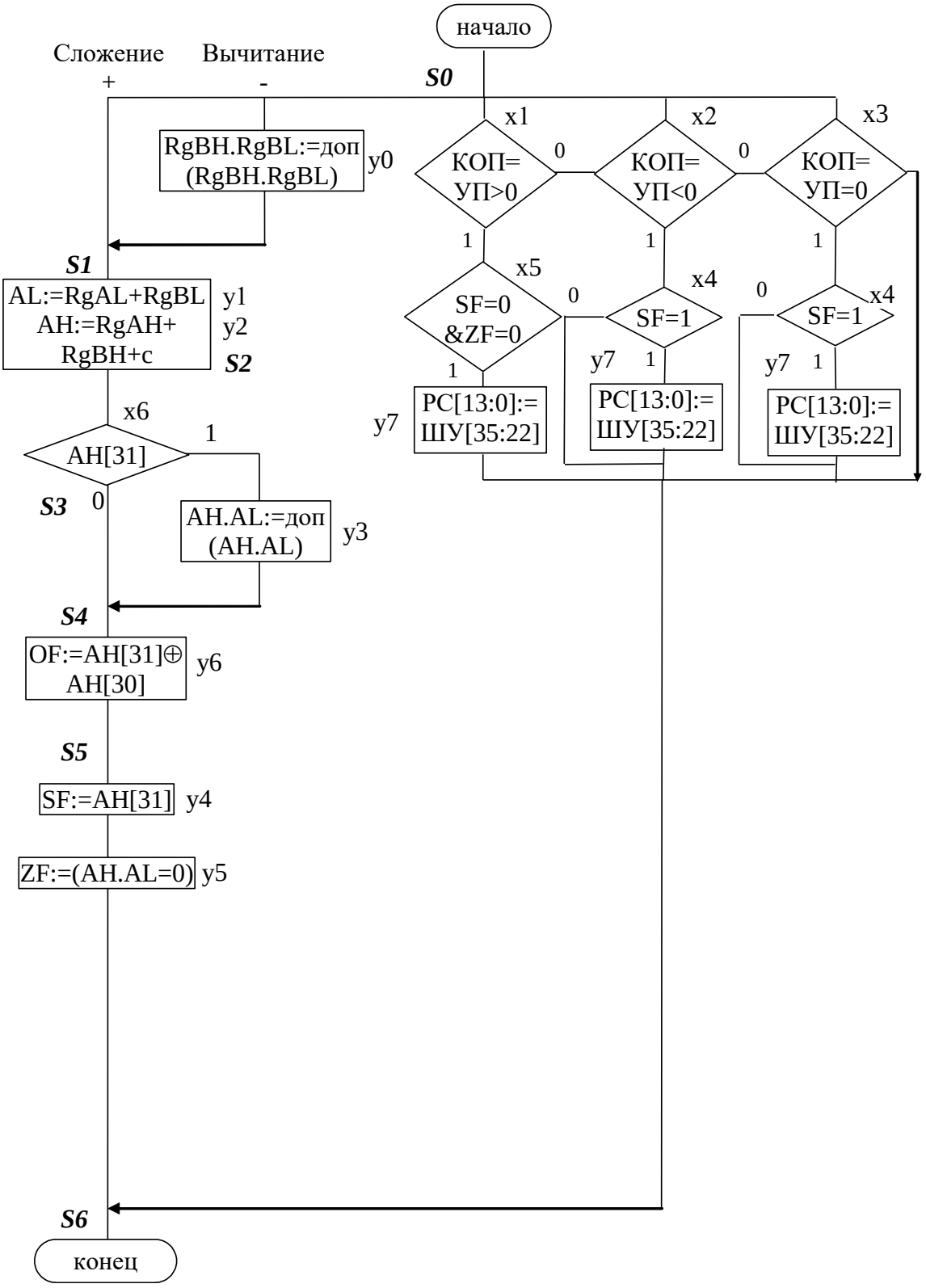


Рис. 8. Обобщенный размеченный алгоритм АЛУ

В соответствии обобщенным размеченным алгоритмом командного цикла процессора (рис. 8, МПА УУ АЛУ процессора должен фиксировать 6 состояний (состояния s0…s6). Для хранения в памяти МПА кодов этих состояний

воспользуемся D-триггерами, которые функционируют на основе следующей таблицы истинности (табл. 4).

Таблица 4

Таблица истинности для D-триггера

Входной сигнал	Исходное состояние	Конечное состояние
0	0	0
0	1	0
1	0	1
1	1	1

Выбираем количество триггеров, соответствующее числу, равному ближайшей наибольшей степени двойки. Этот показатель степени равен 3; следовательно, необходимо 3 D-триггера.

Комбинационная схема КСх состоит из набора программируемых логических интегральных схем ПЛМ или ПЛИС, которые, в зависимости от значений логических сигналов X = $\{x_0, x_1, \ldots, x_l\}$ и элементов множества S = $\{S_0, S_1, \ldots, S_m\}$, определяющего текущее состояние устройства, вырабатывают множество выходных значений, состоящее из подмножества управляющих сигналов Y = $\{y_0, y_1, \ldots, y_k\}$ и подмножества элементов $S' = \{S'_0, S'_1, ..., S'_m\}$, определяющего следующее состояние устройства. Триггеры T_1, T_2, T_3 хранят код состояния МПА, соответствующий номеру бита, который будет истинным на выходе дешифратора DC в унитарном коде текущего состояния S (рис. 9).

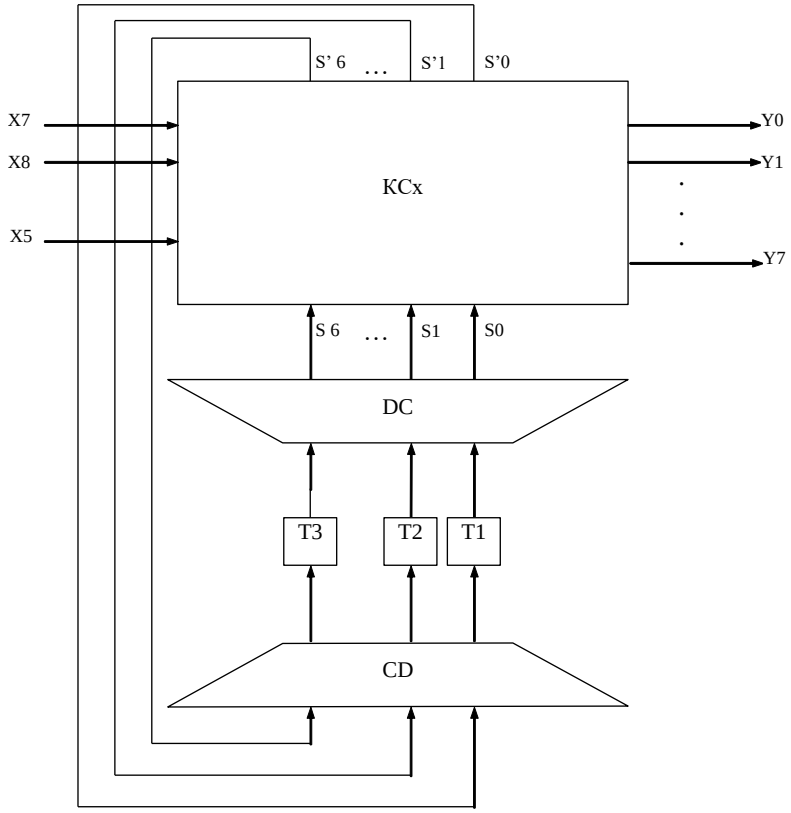


Рис. 9. Структура управляющего автомата процессора на жесткой логике

На выходе КСх образуется унитарный код следующего состояния S' автомата, который шифратором CD сворачивается в номер, используемый в следующем такте как код будущего текущего состояния.

Для ведомого УУ процессора список кодов и таблица переходов автомата будут следующими (табл. 5).

Таблица 5

Список кодов состояний и таблица переходов ведущего управляющего автомата процессора

Исходное состояние S	Код исходного состояния S	Состояние перехода S'	Код состояния перехода S'	Входной набор	Выходной набор	Функции возбуждения триггеров
s0	000	s'1	001	-	y1	D3
s1	001	s'2	010	-	Y2	D2
s2	010	s'3	011	$\overline{x6}$	-	D2,D3
s2	010	s'4	100	x6	Y3	D1

s3	011	s'4	100	-	-	
s4	100	s'5	101	-	Y6	D1.D3
s5	101	s'6	110	-	Y4,Y5	D1,D2
s0	000	s'6	110	x1,x5	Y7	D1,D2
s0	000	s'6	110	x2,x4	Y7	D1,D2
s0	000	s'6	110	x3,x4	Y7	D1,D2

На основе данных таблицы 5 запишем функции возбуждения триггеров, при этом соответствующие коды исходных состояний будем обозначать буквой а для значения единица и $\bar{a}$ для значения ноль.

$D1 = a_2\bar{a}_1\bar{a}_0 x6Y3 \vee a_2\bar{a}_1\bar{a}_0 Y6 \vee a_2\bar{a}_1\bar{a}_0 Y4Y5 \vee a_2a_1\bar{a}_0 x1x5Y7 \vee a_2a_1\bar{a}_0 x2x4Y7 \vee a_2a_1\bar{a}_0 x3x4Y7$

$D2= \bar{a}_2a_1\bar{a}_0Y2 \vee \bar{a}_2a_1a_0Y2\bar{x6} \vee a_2a_1\bar{a}_0Y4Y5 \vee a_2a_1\bar{a}_0 x1x5Y7 \vee a_2a_1\bar{a}_0 x2x4Y7 \vee a_2a_1\bar{a}_0 x3x4Y7$

$D3= \bar{a}_2\bar{a}_1a_0Y1 \vee \bar{a}_2a_1a_0\bar{x6} \vee a_2\bar{a}_1a_0Y6.$

На основе спроектированного операционного автомата и арифметико-логического устройства разработана функциональная схема векторного процессора (рис. 10). Функциональная схема процессора является объединением названного операционного автомата нижнего уровня процессора и АЛУ, его устройства управления, блоков согласования разрядности шин.

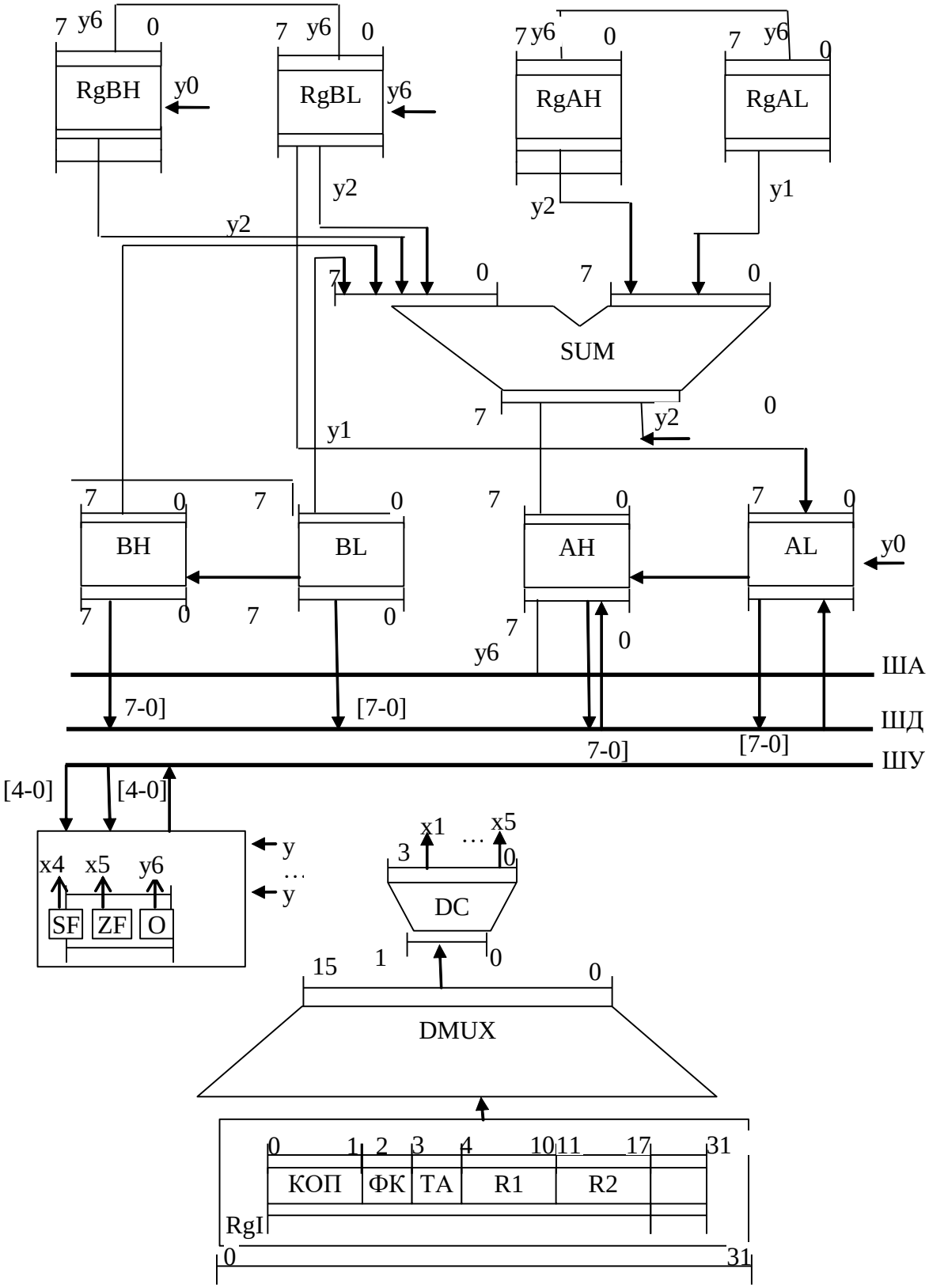


Рис. 10. Функциональная схема векторного процессора

Работа векторного процессора, представленного на рисунке 10 происходит следующим образом.

Первоначально предполагается, на вход векторного процессора поступили два вектора сложности $W = \|w_i\|_n$, соответствующие двум операндам x_{q_i} длиной не более ста, разрядностью ≤ 8 с количеством таких векторов не более шести.

Предложенный процессор считывает разрядности полей КОП[1-0] для определения того, какая именно операция должна быть выполнена в данный момент времени. При этом в данный момент на входе АЛУ процессора присутствует два вектора, которые подаются на вход векторного сумматора. После окончания операции суммирования происходит сохранение результатов в регистрах AH.AL и установка соответствующих флагов выполнения операции SF, ZF. O.

Для определения производительности предложенного процессора и его сравнения со стандартным скалярным процессором, проведено моделирование сложения двух наборов по десять чисел.

Скалярный процессор в этом случае последовательно выбирает пары чисел после чего складывает их. То есть необходимо повторить цикл 10 раз, прочитать следующую инструкцию, декодировать ее, получить первое слагаемое, получить второе слагаемое сложить сохранить результат и т.д. [1].

При использовании векторного процессора необходимо выполнить следующие инструкции: прочитать инструкцию, декодировать ее, получить 10 чисел, получить 10 чисел, получить 10 чисел, сложить, умножить их и сохранить результат [17,18].

Из приведенного анализа очевиден выигрыш в скорости выполнения около двух раз, что подтверждает выигрышность использования предложенного

векторного процессора планирования загрузки процессоров в мультипроцессорных системах критического назначения.

Заключение

В работе была предложено устройство планирования загрузки процессоров в мультипроцессорных системах критического назначения, ориентированный на аппаратную реализацию. Выполнено программное моделирование, подтверждающее справедливость применения разработанного векторного устройства планирования загрузки планирования процессоров в мультипроцессорных системах критического назначения. Проведено программное моделирование, в результате которого установлена зависимость влияния загрузки заданий от количества процессоров мультипроцессорной системы и зависимость выбора необходимого и достаточного их количества для обработки определенного количества информации.

Предложено векторное устройство планирования загрузки процессоров в мультипроцессорных системах критического назначения.

Библиографический список

1. Цилькер Б.Я. Организация ЭВМ и систем. - СПб.: Питер, 2007. – 668 с.

2. Оре О. Теория графов. - М.: Наука, 1968. - 352 с.

3. Кормен Т.М. и др. Алгоритмы: построение и анализ. Алгоритмы для работы с графами. - СПБ.: Издательский дом "Вильямс", 2006. - 1296 с.

4. Stallings W. Computer organization and architecture. Designing for Perfomance. Tenth Edition, Prentice-Hall, 1999. URL:

http://home.ustc.edu.cn/~leedsong/reference_books_tools/Computer%20Organization%20and%20Architecture%2010th%20-%20William%20Stallings.pdf

5. Левин И.И., Штейнберг Б.Я. Сравнительный анализ эффективности параллельных программ для различных архитектур параллельных ЭВМ // Искусственный интеллект. 2001. № 3. С. 234 - 242.

6. Слюсар В. Многоядерная архитектура: проблемные аспекты // Электроника НТБ. 2007. № 1. URL: https://www.electronics.ru/journal/article/514

7. Майоров С.А., Кириллов В.В., Приблуда А.А. Введение в микроЭВМ. - Ленинград: Машиностроение, 1988. - 304 с.

8. Микушин А.В., Сажнев А.М., Сединин В.И. Цифровые устройства и микропроцессоры. - СПб.: БХВ-Петербург, 2010. – 832 с.

9. Бусурин В.И., Медведев В.М., Карабицкий А.С., Гроппа Д.В. Алгоритмы анализа цифровой информации для оптимизации контроля систем управления // Труды МАИ. 2017. № 97. URL: http://trudymai.ru/published.php?ID=87277

10. Басов Р.Г., Борзов Д.Б., Локтионова О.Г. Программа моделирования загрузки мультипроцессорной системы. Свидетельство о государственной регистрации программ на ЭВМ № 2019616927 РФ, 30.05.2019.

11. Басов Р.Г., Борзов Д.Б., Титов В.С. Метод и алгоритм планирования загрузки процессоров в мультипроцессорных системах критического назначения // Телекоммуникации. 2020. № 1. С. 41 – 48.

12. Трахтенгерц Э.А. Введение в теорию анализа и распараллеливания программ ЭВМ в процессе трансляции - М.: Наука, 1981. - 254 с.

13. Tyutlyaeva E.O., Konyukhov S.S., Odintsov I.O., Moskovsky A.A. Seismic Processing Performance Analysis on Different Hardware Environment // Seismic Processing Performance Analysis on Different Hardware Environment, 2017, vol. 4, no. 3, pp. 80 - 90. DOI: 10.14529/jsfi170305

14. Хокни Р., Джесссхоуп К. Параллельные ЭВМ. Архитектура, программирование и алгоритмы. - М.: Радио и связь, 1986. - 392 с.

15. Каляев А.В., Левин И.И. Модульно-наращиваемые многопроцессорные системы со структурно-процедурной организацией вычислений. - М.: Янус-К, 2003. – 379 с.

16. Борзов Д.Б., Чернецкая И.Е. Проектирование процессора ЭВМ. – Курск: Изд-во Юго-Западный государственный университет, 2020. - 199 с.

17. Momose S. SX-Aurora TSUBASA. Brand-new Vector Supercomputer // SC’17 Supercomputer Forum, 2017. URL: https://www.osp.ru/os/2018/01/13053934

18. Кузьминский М. Из ускорителей в процессоры // Открытые системы СУБД. 2016. № 3. С. 4 – 6. URL: https://www.osp.ru/os/2016/03/13050252

19. Маркарян А.О., Чурков И.С. Задачи управления в системе принятия решений при отказах автоматизированных рабочих мест // Труды МАИ. 2020. № 113. URL: http://trudymai.ru/published.php?ID=118150. DOI: 10.34759/trd-2020-113-10

20. Набатов А.Н., Веденяпин И.Э., Мухтаров А.Р. Применение онтологического подхода к процессу проектирования информационной системы // Труды МАИ. 2018. № 102. URL: http://trudymai.ru/published.php?ID=99177

Processor load scheduling device in multiprocessor systems of critical purpose

Borzov D.B.[1*], Basov R.G.[1], Titov V.S.[1***], Sokolova Yu.V.[2****]**

[1]Southwestern State University, SWSU, 94, str. 50 Let Oktyabrya, Kursk, 305040, Russia

[2]Lavochkin Research and Production Association, NPO Lavochkin, 24, Leningradskay str., Khimki, Moscow region, 141400, Russia

**e-mail: borzovdb@kursknet.ru*

***e-mail: r_basov@eureca.ru*

****e-mail: titov-kstu@rambler.ru*

*****e-mail: jv.sokolova@mail.ru*

Abstract

The need to prompt response from the computing system side emerges in the state-of-the-art multiprocessor critical systems. The systems, which failures lead to the losses (economic, physical, human etc.) are defined as critical systems. In the event of failure, such systems are the subject to high requirements for operability, reliability, safety, security, etc. All costs herewith associated with introduction of changes to such system or its replacing (direct, indirect, etc.) are more important that the losses occurred in the case of the multiprocessor system failure itself. It is obvious that minimization of both time and hardware costs, necessary for the multiprocessor system response to the hazardous situation is of most importance.

The failures occurring in the internal processor modules, such as aircraft cockpits, surveillance systems, targeting, atomic systems etc., can be assigned to the critical situations. In case of the multiprocessor system failure herewith, its performance and response decrease, which is unacceptable in critical systems. One of the options for this

issue solving may be processor loading planning in multiprocessor systems. In this case, the multiple loading of several processors by the same task (routine, subroutine, algorithm, file, etc.) can be avoided, and, with this, the queue of the incoming tasks can be planned in such a fashion that they are being fed simultaneously. This allows reduce unplanned operation downtime and, at the same, time increase its availability factor along with increased performance.

The presented work is devoted to critical multiprocessor systems, to which high availability objects, which operability is critical for any kind of activity, can be assigned. In such cases, a person, country, enterprise, organization, etc. is distinguished. A failure in this case leads to in the system response time decrease and a further decrease in its performance, and, thus, the availability. In this case, the application of the software for solving this issue is unacceptable, which means that it is necessary to use specialized hardware load scheduling tools.

Keywords: multiprocessor system, scheduling, loading, high availability, assignment, method, algorithm, schedule.

References

1. Tsil'ker B.Ya. *Organizatsiya EVM i system* (Organization of computers and systems), Saint Petersburg, Piter, 2007, 668 p.

2. Ore O. *Teoriya grafov* (Theory of Graphs), Moscow, Nauka, 1968, 352 p.

3. Kormen T.M. et al. *Algoritmy: postroenie i analiz. Algoritmy dlya raboty s grafami* (Algorithms: construction and analysis. Algorithms for working with graphs), Saint Petersburg, Izdatel'skii dom "Vil'yams", 2006, 1296 p.

4. Stallings W. *Computer organization and architecture. Designing for Perfomance.* Tenth Edition, Prentice-Hall, 1999. URL: http://home.ustc.edu.cn/~leedsong/reference_books_tools/Computer%20Organization%20and%20Architecture%2010th%20-%20William%20Stallings.pdf

5. Levin I.I., Shteinberg B.Ya. *Iskusstvennyi intellect*, 2001, no. 3, pp. 234 - 242.

6. Slyusar V. *Elektronika NTB*, 2007, no. 1. URL: https://www.electronics.ru/journal/article/514

7. Maiorov S.A., Kirillov V.V., Pribluda A.A. *Vvedenie v mikroEVM* (Introduction to microcomputers), Leningrad, Mashinostroenie, 1988, 304 p.

8. Mikushin A.V., Sazhnev A.M., Sedinin V.I. *Tsifrovye ustroistva i mikroprotsessory* (Digital devices and microprocessors), Saint Petersburg, BKhV-Peterburg, 2010, 832 p.

9. Busurin V.I., Medvedev V.M., Karabitskii A.S., Groppa D.V. *Trudy MAI*, 2017, no. 97. URL: http://trudymai.ru/eng/published.php?ID=87277

10. Basov R.G., Borzov D.B., Loktionova O.G. *Svidetel'stvo o gosudarstvennoi registratsii programm na EVM № 2019616927 RF*, 30.05.2019.

11. Basov R.G., Borzov D.B., Titov V.S. *Telekommunikatsii*, 2020, no. 1, pp. 41 – 48.

12. Trakhtengerts E.A. *Vvedenie v teoriyu analiza i rasparallelivaniya programm EVM v protsesse translyatsii* (Introduction to the theory of analysis and parallelization of computer programs in the translation process), Moscow, Nauka, 1981, 254 p.

13. Tyutlyaeva E.O., Konyukhov S.S., Odintsov I.O., Moskovsky A.A. Seismic Processing Performance Analysis on Different Hardware Environment, *Seismic Processing Performance Analysis on Different Hardware Environment*, 2017, vol. 4, no. 3, pp. 80 - 90. DOI: 10.14529/jsfi170305

14. Khokni R., Dzhesskhoup K. Parallel'nye EVM. *Arkhitektura, programmirovanie i algoritmy* (Parallel computers. Architecture, programming and algorithms), Moscow, Radio i svyaz', 1986, 392 p.

15. Kalyaev A.B., Levin I.I. *Modul'no-narashchivaemye mnogoprotsessornye sistemy so strukturno-protsedurnoi organizatsiei vychislenii* (Modularly expandable multiprocessor systems with structural and procedural organization of computations), Moscow, Yanus-K, 2003, 379 p.

16. Borzov D.B., Chernetskaya I.E. *Proektirovanie protsessora EVM* (Computer processor design), Kursk, Izd-vo Yugo-Zapadnyi gosudarstvennyi universitet, 2020, 199 p.

17. Momose S. SX-Aurora TSUBASA. Brand-new Vector Supercomputer, *SC'17 Supercomputer Forum*, 2017. URL: https://www.osp.ru/os/2018/01/13053934

18. Kuz'minskii M. *Otkrytye sistemy SUBD*, 2016, no. 3, pp. 4 – 6. URL: https://www.osp.ru/os/2016/03/13050252

19. Markaryan A.O., Churkov I.S. *Trudy MAI*, 2020, no. 113. URL: http://trudymai.ru/eng/published.php?ID=118150. DOI: 10.34759/trd-2020-113-10

20. Nabatov A.N., Vedenyapin I.E., Mukhtarov A.R. *Trudy MAI*, 2018, no. 102. URL: http://trudymai.ru/eng/published.php?ID=99177