ФЕДЕРАЛЬНОЕ АГЕНСТВО СВЯЗИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБЩЕОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИИ И ИНФОРМАТИКИ»

Кафедра
вычислительных систем

КУРСОВАЯ РАБОТА

по дисциплине «Архитектура ЭВМ»

Выполнил:
студент гр. ИП-213
Дмитриев Егор
Александрович

Проверил:
Преподаватель
Деревцова В. А.

Новосибирск

2024

Оглавление

ПОСТАНОВКА ЗАДАЧИ

Разработать транслятор с языка Simple Basic. Итог работы транслятора – бинарный файл с образом оперативной памяти Simple Computer, который можно загрузить в модель и выполнить;

Доработать модель Simple Computer – реализовать алгоритм работы блока «L1-кэш команд и данных» и модифицировать работу контроллера оперативной памяти и обработчика прерываний таким образом, чтобы учитывался простой процессора при прямом доступе к оперативной памяти;

Разработать транслятор с языка Simple Basic. Итог работы транслятора – текстовый файл с программой на языке Simple Basic.

Транслятор с языка Simple Assembler

Разработка программ для Simple Computer может осуществляться с использованием низкоуровневого языка Simple Assembler. Для того чтобы программа могла быть обработана Simple Computer необходимо реализовать транслятор, переводящий текст Simple Assembler в бинарный формат, которым может быть считан консолью управления.

Программа транслируется по строкам, задающим значение одной ячейки памяти. Каждая строка состоит как минимум из трех полей: адрес ячейки памяти, команда (символьное обозначение), операнд. Четвертым полем может быть указан комментарий, который обязательно должен начинаться с символа точка с запятой. Название команд представлено в таблице 1. Дополнительно используется команда =, которая явно задает значение ячейки памяти в формате вывода его на экран консоли (+XXXX).

Команда запуска транслятора должна иметь вид: sat файл.sa файл.o, где файл.sa – имя файла, в котором содержится программа на Simple Assembler, файл.o – результат трансляции.

Транслятор с языка Simple Basic

Для упрощения программирования пользователю модели Simple Computer должен быть предоставлен транслятор с высокоуровневого языка Simple Basic. Файл, содержащий программу на Simple Basic, преобразуется в файл с кодом Simple Assembler. Затем Simple Assembler-файл транслируется в бинарный формат.

В языке Simple Basic используются следующие операторы: rem, input, output, goto, if, let, end. Каждая строка программы состоит из номера строки, оператора Simple Basic и параметров. Номера строк должны следовать в возрастающем порядке. Все команды за исключением команды конца программы могут встречаться в программе многократно. Simple Basic должен оперировать с целыми выражениями, включающими операции +, -, *, и /. Приоритет операций аналогичен C. Для того чтобы изменить порядок вычисления, можно использовать скобки.

Транслятор должен распознавания только букв верхнего регистра, то есть все символы в программе на Simple Basic должны быть набраны в верхнем регистре (символ нижнего регистра приведет к ошибке). Имя переменной может состоять только из одной буквы. Simple Basic оперирует только с целыми значениями переменных, в нем отсутствует объявление переменных, а упоминание переменной автоматически вызывает её объявление и присваивает ей нулевое значение. Синтаксис языка не позволяет выполнять операций со строками.

БЛОК – СХЕМЫ ИСПОЛЬЗУЕМЫХ АЛГОРИТМОВ

Рисунок 1– Схема алгоритма Кэш-процессора

Рисунок 2 – Схема алгоритма SimpleAssembler

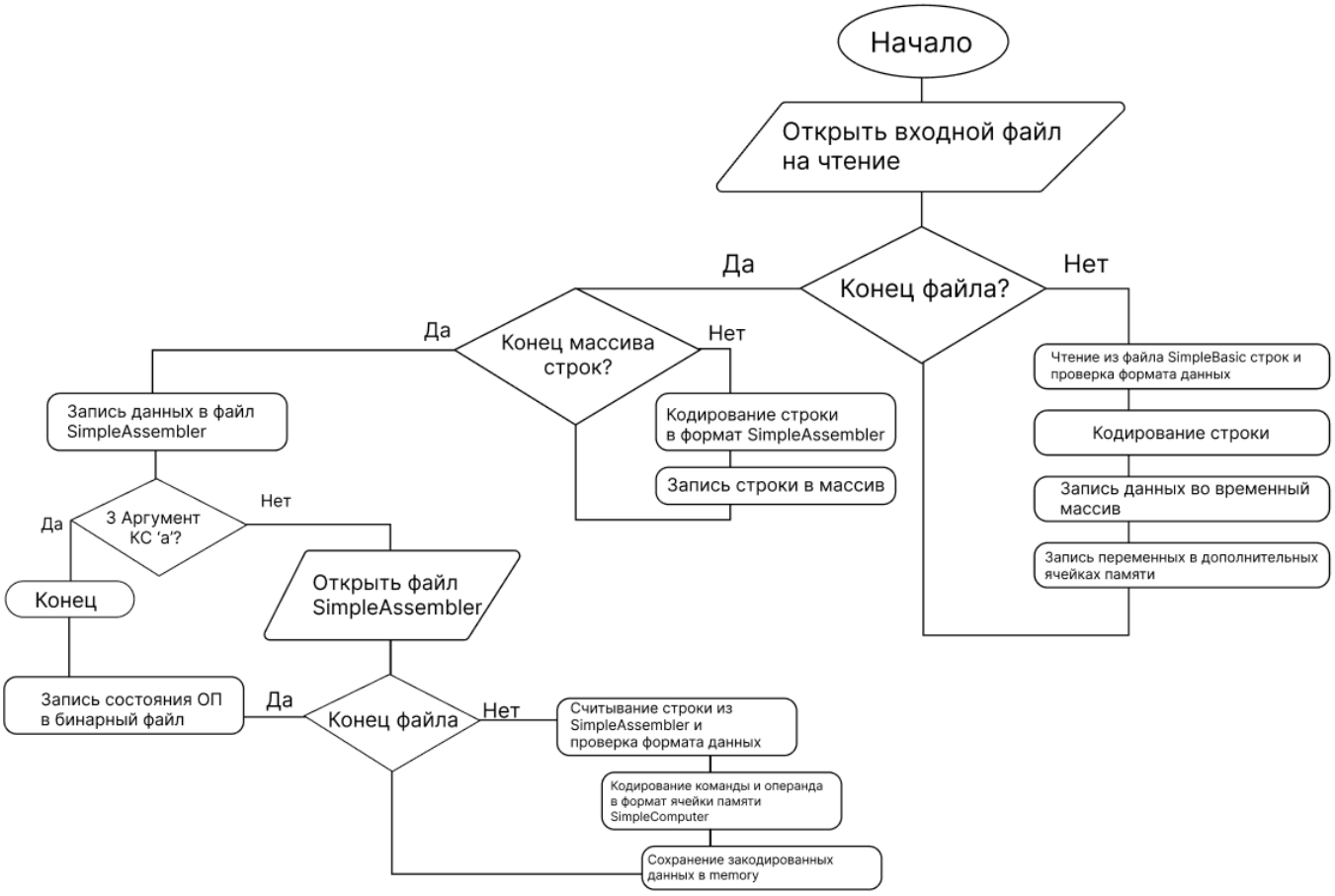

Рисунок 3 – Схема алгоритма SimpleBasic

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Кеш-процессора

sc_initCache.c

Подпрограмма содержит функцию, предназначенную для инициализации кэш – памяти. Для этого сначала с помощью встроенной функции языка Си malloc выделяется блок памяти под кэш, размер массива равен константному значению cache_size умноженному на размер одной записи cacheline. Если выделение памяти прошло успешно, осуществляется дальнейшая работа с блоком, иначе функция завершает свою работу. После успешного выделения памяти функция проходит по каждой линии кэша и инициализирует её значениями полей address, lastAccessTime и data (значение ячеек памяти).

sc_updateCache.c

Функция sc_updateCacheAfterSave предназначена для обновления состояния кэша, с ее помощью происходит поиск наименее используемой ячейки. Для этого сначала проверяется значение флага updateStatic. Если он имеет значение true, то переменные currentIndex и countIt обнуляются, так происходит сброс состояния кэша перед началом новой операции. Затем функция определяет адрес линии, в которую будет сохраняться новое значение, и проверяет, содержится ли в кэше копия данных для этого адреса, если копия данных не найдена, функция добавляет новую запись (если кэш еще не заполнен), или замещает запись, которую используют реже всего. Если копия данных уже хранится в кэше, она обновляется данными из основной памяти. В конце работы функции обновляется время последнего доступа к текущей линии.

Функция initStatic позволяет выполнить обнуление кэша при нажатии reset с помощью флага.

sc_findLeastRecentlyUsedCacheEntry.c

В функции объявляются переменные leastRecentlyUsedIndex (равна 0) и leastRecentTime (равна cache[0].lastAccessTime). В функция происходит прохождение по всем линиям кэша. Для каждой линии функция сравнивает время её последнего доступа (‘cache[i].lastAccessTime’) с временем последнего доступа к кэш – линии с наименьшим временем (‘leastRecentTime’). Если время последнего доступа к текущей кэш – линии меньше, чем leastRecentTime, то переменные leastRecentTime и leastRecentlyUsedIndex перезаписываются соответствующими значениями. После завершения цикла функция возвращает индекс линии с наименьшим временем последнего доступа, с помощью чего в программе определяется, какая линия кэша использовалась раньше других.

sc_printCache.c

Графическое представление содержимого кэш–памяти. В цикле for происходит проход по всем линиям кэша, для каждой из которых выводится её содержимое на экран: адрес кэш–линии и содержимое каждого ее элемента. Если линия оказалось пустой, её

адрес равен –1, выводится символ **“–”**. Все данные выводятся в определенном формате. Если старший бит (cache[i].data[j] >> 14) равен 1, это показатель отрицательного числа. В зависимости от этого выводится соответствующий формат числа со знаком плюс или минус.

Трансляторы

SimpleAssembler

main.c

Программа проверяет, что количество аргументов равно 3. Происходит попытка открыть входной файл .sa для чтения в бинарном режиме. Вызывается функция для инициализации памяти компьютера. Программа построчно считывает файл для извлечения всех данных. Если команда найдена и валидна, она кодируется вместе с операндом (если есть) в формат машины Simple Computer, а затем записывается в память. После обработки всех строк из файла, программа вызывает функцию для сохранения содержимого памяти Simple Computer в указанный выходной файл. Если все этапы прошли успешно, программа завершается с кодом 0.

SimpleBasic – SimpleAssembler – Машинный код

sb_clsBp.c

Удаление всех пробелов из строки str и сохранение результата в исходной строке.

sb_getGOTO.c

Поиск в массиве строк программы на Simple Basic адреса команды перехода (GOTO) по заданному номеру строки.

sb_getVariableAddress.c

Получение адреса переменной в памяти, представленной массивом variables.

sb_getVariableName.c

Получение имени переменной по её адресу в памяти.

sb_basicToAssembler.c

В начале программы открываются файлы для чтения исходного кода и его трансляции на язык Assembler. Происходит проверка на то, чтобы строка не была пустой и содержала необходимое количество операндов. Файл считывается построчно, каждая строка проверяется на соответствие тому или иному шаблону, например, это может быть строка, в которой выполняется присваивание, арифметическая операция, переход, чтение или обозначение конца программы. Для этого каждая строка разбирается на элементы, среди которых идет поиск обозначений ключевых команд или символов. После однозначного определения содержания строки происходит преобразование операторов кода в соответствующие команды на языке Assembler, или появляется сообщение о неверном синтаксисе.

Следующий шаг трансляция с ассемблера в машинный код. Сначала внутри цикла проверяется, не является ли текущая строка пустой. Далее выделяется команда, и если это чтение, запись, загрузка или другая основная команда строка дополнительно анализируется, чтобы получить адрес переменной, представленной символом “ch”. Если это операция загрузки, то также сохраняет число “d” в “numArray” по вычисленному адресу. Если была определена операция перехода, то она дополнительно анализируется, чтобы получить номер строки “goto_line” и затем сформатировать новую строку с операцией и адресом перехода в “asmStringOut”. Команда окончания программы транслируется без дополнительных действий. Если строка не содержит команду, то проверяется, является ли она объявлением переменной или числовым значением, и соответствующим образом обрабатывается.

После обработки все строки кодируются записываются в файл машинного кода.

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Результат трансляции с SimpleBasic на SimpleAssembler:

simplebasic > factorial.sb

```
10 INPUT A
20 LET B = 1
30 LET C = 1
40 LET B = A * B
50 LET A = A - C
60 IF A > 0 GOTO 40
70 PRINT B
80 END
```

Рисунок 1 – Код на языке SimpleBasic

simplebasic > factorial.sa

```
00 READ 15
01 LOAD 16
02 STORE 17
03 LOAD 16
04 STORE 18
05 LOAD 15
06 MUL 17
07 STORE 17
08 LOAD 15
09 SUB 18
10 STORE 15
11 LOAD 15
12 JNS 05
13 WRITE 17
14 HALT 00
15 = +0000
16 = +0001
17 = +0000
18 = +0000
```

Рисунок 2 – Код на языке SimpleAssembler

Отрисовка консоли Simple Computer:

Рисунок 3 – Консоль SimpleComputer

Результат выполнения программы подсчета факториала 6:

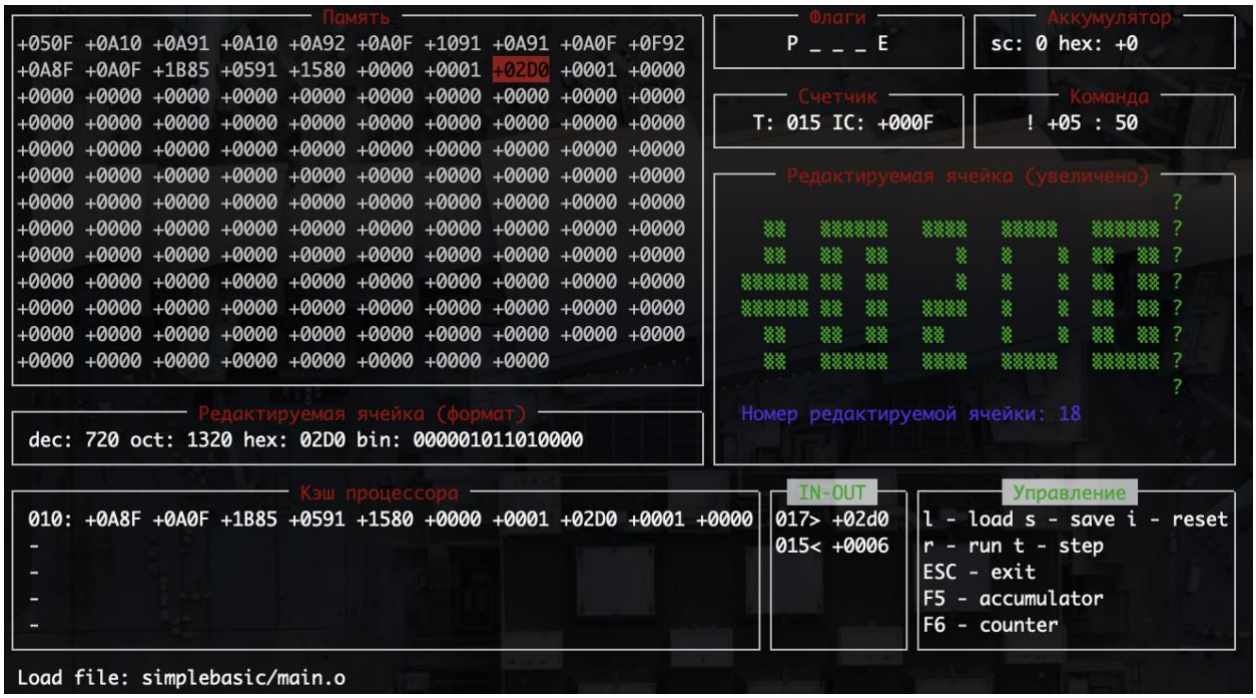

Рисунок 4 – Вычисление факториала от числа 6

ЗАКЛЮЧЕНИЕ

В ходе работы над курсовым проектом были реализованы следующие задачи:

1. Доработка модели Simple Computer путем интеграцией блока “Кеш процессора”.

2. Разработка транслятора с языка Simple Basic, позволяющего эффективно преобразовывать исходный код в Assembler, что упрощает процесс программирования.

3. Разработка транслятора с языка Simple Assembler, позволяющего осуществлять перевод кода с языка Assembler в машинные команды, что обеспечивает высокую производительность программы.

Готовая модель Simple Computer была протестирована вручную. Формат входных и выходных данных соответствует заявленному в задании. Отрисовка интерфейса проходит без сбоев. Вычисления, кодирование и декодирование проходят успешно, результаты работы тестовых программ совпадают с ожидаемыми.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. ЭВМ и периферийные устройства [Учебное пособие]. С.Н. Мамойленко О.В. Молдованова.
2. Язык программирования C // Metanit URL: https://metanit.com/c/tutorial// Дата обращения: 23.05.2024
3. Язык Си // C URL: https://prog-cpp.ru//
4. Signal handling // DevTut URL: https://devtut.github.io/c/