Санкт-Петербургский политехнический университет Петра Великого
Кафедра компьютерных систем и программных технологий

Отчёт по курсовой работе

Дисциплина: Низкоуровневое программирование

Тема: Симулятор 8-битного процессора intel 8051

Выполнил студент гр. 23531/2
_____ Н.С. Макаревич

Преподаватель
_____ М.Х. Ахин
«___» _____ 2018 г.

Санкт-Петербург, 2018 г.

Содержание

1 Задание

Реализовать симулятор работы процессора intel 8051. Содержимое памяти задается и выводится в текстовый файл. Формат файла фиксируется в ТЗ.
Необходимо обеспечить возможность имитации работы процессора до останова (HLT) или в течение заданного числа шагов (инструкций). Необходимо обеспечить возможность отладки машины (вывод содержимого памяти и регистров, возобновление работы машины).
Задание не предполагает имитацию ввода-вывода и контроллера прерываний.

2 Описание функциональности симулятора

Данный симулятор процессора intel 8051 способен запускать программы либо из бинарного файла прошивки микроконтроллера (образ памяти программ), либо из специального текстового формата.
Пользователь составляет файл, описывающий состояние памяти устройства в начальный момент времени. Бинарный файл прошивки содержит только образ памяти программ и воспринимается симулятором как состояние процессора в начальный момент времени.
Текстовый формат содержит информацию о состоянии памяти программ, памяти данных, регистров и счётчика инструкций, так что может быть использован в качестве снимка состояния процессора в определённый момент времени.
Симулятор позволяет просматривать последовательность выполняемых инструкций, менять скорость выполнения программы, а также исполнять программу пошагово – по одной инструкции.
Пользователь может остановить программу в нужном месте, нажав клавишу Enter, указав необходимый адрес в текстовом файле, а также при помощи аргументов командной строки. Пока программа остановлена, можно просматривать состояние памяти программ и памяти данных, а также делать снимки состояния.
По умолчанию память программ имеет размер 4 Кбайта, а память данных – 256 байт, но при необходимости их можно расширить до 64 Кбайт.
Большинство прошивок микроконтроллера находится в формате intel hex, но их можно легко перевести в бинарный файл (например, с помощью программы avr-objcopy).

3 Система команд процессора

Набор инструкций, которые может обрабатывать симулятор, должен соответсововать тому набору инструкций, которыми оперирует реальный процессор intel 8051.
Описание всех инструкций, регистров и поведения процессора приведено в книге

Горюнов А.Г., Ливенцов С.Н. Архитектура микроконтроллера Intel 8051
Томск: Изд-во ТПУ, 2005. - 86 с.

4 Работа с симулятором

4.1 Аргументы командной строки

-h --help
Показать краткую справку по командам симулятора.
Выводит в консоль содержимое файла resources/help.txt

-d --debug
Включает отладочные средства (реакцию на брэйкпойнты, вывод в консоль). Если флаг не установлен, то симулятор исполняет программу, ничего не выводя в консоль.

-i --infile
Имя входного файла.
Пример: -i myfile

-o --outfle
Имя выходного файла.
Так будет называться файл образа памяти, полученный после завершения программы. Если пользователь делает промежуточные снимки состояния процессора, то в начало имени файла добавляются дата, время и номер дампа.
Имя файла по-умолчанию – "memory".
Пример: -o myfile

-c --clk
Время задержки между исполнением инструкций. Количество милисекунд, целое беззнаковое число. По умолчанию задержки нет.
Пример: -c 1000

-v --verbose
Verbose режим. Отображает в реальном времени последовательность машинных команд, которые исполняет процессор. По умолчанию выключен.

-m --mode
Тип принимаемого на вход файла (bin или text)
Пример: -m bin

-b --break
Добавление брэйкпойнтов
В этом параметре передаётся адрес в шестнадцатеричном виде. На этом адресе будет добавлен брэйкпойнт.
^ означает, что брэйкпойнт сработает перед исполнением соответствующей инструкции. _ означает, что он сработает после её исполнения.
Пример: -b ^2C

--nobreak Игнорирование брэйкпойнтов
С этим флагом программа не будет останавливаться на брэйкпойнтах, обозначенных пользователем в файле или в командной строке.

-s --save
Добавление сэйвпойнтов

В этом параметре передаётся адрес в шестнадцатеричном виде. На этом адресе будет сделан снимок состояния процессора.
^ означает, что снимок будет сделан перед исполнением соответствующей инструкции. _ означает, что он будет сделан после её исполнения.
Пример: -s _2C

-z --convert
С этим флагом симулятор не исполняет программу, а просто преобразует входной бинарный файл в текстовый файл - образ памяти.

-e --end
Задание конечного адреса программы.
Адрес задаётся как шестнадцатеричное неотрицательное число.
Если счётчик PC превысит это значение, то программа будет остановлена, а конечное состояние сохранено в файл.
Пример: -e E6

--epm Флаг, включающий поддержку внешней памяти программ.

--edm Флаг, включающий поддержку внешней памяти данных.

--step Step-by-step режим. С этим флагом программа будет исполняться по одной инструкции по нажатию клавиши Enter.
Брэйкпойнты генерируются после каждой исполненной инструкции.

4.2 Интерфейс

Симулятор имеет консольный интерфейс.
Если флаги -d и -v не установлены, то в консоль ничего не выводится. Программа исполняется без временных задержек и конечное состояние процессора записывается в выходной файл.
Если установлен флаг -d (debug), то в консоль будет выводиться информация о брэйкпойнтах и сэйвпойнтах.
Если же установлены флаги -d и -v (verbose), то в консоль будут выводиться инструкции, которые исполняет процессор.

Пример вывода программы:

Рис. 1

В квадратных скобках выведено значение счётчика инструкций PC. Затем в строке представлены коды инструкций, они могут состоять из одного, двух или трёх байт. Последней в строке указывается мнемоника в круглых скобках, соответствующая данной инструкции (Для случая, когда программа запущена из текстового файла).
Если пользователю требуется остановить исполнение инструкций, достаточно нажать клавишу Enter – это вызовет Breakpoint в текущем месте программы.
Когда программа остановлена на брэйкпойнте, то пользователь может использовать команды для просмотра памяти и сохранения состояния. Они описаны ниже в разделе о брэйкпойнтах.

5 Текстовый формат снимка состояния симулятора

Снимок состояния микроконтроллера intel 8051
представляет собой JSON-структуру вида:

```
{
  "PC": "#.."
  "rga": "#..",
  "rgb": "#..",
  "rgc": "#..",
  ...

  "program": "...",
  "data": "..."
}
```

Состояние счётчика инструкций (PC), памяти программы (program) и памяти данных (data) задаётся в специальном текстовом формате.
Значения регистров указываются в виде шестнадцатеричного числа в строке.
Память программы и память данных заданы в виде строк, содержащих мнемокоды инструкций и числовые константы. Пробелы, табуляция и переносы строки используются в качестве разделителя.

5.1 Числа и инструкции в памяти

Обычное число (8 двоичных разрядов) может быть записано в виде:

Шестнадцатеричного числа:	#FF
Двоичного числа:	11111111
Десятичного числа:	*255

Числа должны быть целыми и неотрицательными. Если симулятор прочтёт число больше 255, то программа завершится с ошибкой.

Также пользователь может писать названия инструкций и регистров. Инструкции разделяются пробелами, символами табуляции или переносом строки. Регистр букв не учитывается.
Соответствие мнемоник инструкциям задаётся в специальном конфигурационном файле resources/mnemonics.json
Значения регистров представлены в файле отдельно для удобства, но при этом продублированы в памяти данных.
Если значение регистра указано отдельно, то оно обладает более высоким приоритетом, чем значения в памяти данных.

5.2 Определение мнемокодов

Мнемокоды, соответствующие определённым инструкциям, задаются пользователем в конфигурационном файле resources/mnemonics.json.
Пользователь может на своё усмотрение назначить соответствие различных символьных строк кодам от 0 до 255.
Когда симулятор стартует из текстового файла, то по очереди парсит инструкции, заданные в текстовом формате. Если код инструкции или константы указан в виде числа, например 1101 или #FF, то он сразу декодируется и попадает в память. Если же программа встречает строку, то ищет соответствующий ей код в файле mnemonics.json.
Все мнемокоды должны быть указаны в нижнем регистре.

Формат файла mnemonics.json

```
{
  "nop": 0,
  "ajmp_ad11_0": 1,
  "ljmp_ad16":   2,
  "rr_a":    3,
  "inc_a":   4,
  "inc_ad":  5,
  "acc":     224
}
```

Здесь мы указываем мнемокод как ключ и соответствующий ему код как значение (целое неотрицательное число).

5.3 Брэйкпойнты

В текстовое представление можно добавлять точки останова программы (breakpoint). Когда программа доходит до брэйкпойнта, то приостанавливается и пользователь может выполнять некоторые действия, прежде чем запустит ход выполнения программы дальше.
Можно просматривать значения, хранящиеся в определённых ячейках памяти программ и памяти данных. Для этого необходимо написать букву «p» или «d», которые указывают на место, откуда мы читаем значение. «p» - из памяти программы, «d» - из памяти данных.
Далее после буквы пользователь должен указать адрес ячейки в шестнадцатеричном виде и нажать клавишу Enter.
Например, чтобы прочитать из памяти данных значение, находящееся по адресу 88, достаточно написать d88 .
Также после остановки программы на брэйкпойнте пользователь может сохранить состояние процессора в файл, написав команду "save".
Команда "step"позволяет включать/выключать режим пошагового исполнения инструкций.
Брэйкпойнт в текстовом файле записывается в формате ^BREAK или _BREAK
Символ ^ ставится перед BREAK, если программа должна быть приостановлена после предыдущей инструкции, а символ _ ставится, если программа должна быть приостановлена перед следующей инструкцией.

5.4 Точки сохранения снимка памяти

Можно добавлять в код точки сохранения (savepoint). Когда программа доходит до сейвпойнта, то дампит в текстовый файл состояние памяти и регистров в данный момент.
Сейвпойнт записывается в формате ^SAVE или _SAVE
Символы ^ и _ выполняют ту же функцию, что и в случае с брэйкпойнтами.

Брэйкпойнты и сейвпойнты не влияют на последовательность инструкций, которую в итоге исполняет процессор. Симулятор держит их в памяти отдельно и отслеживает, когда то или иное правило должно сработать.

5.5 Комментарии

Пользователь может включать в текст программы комментарии, которые будут проигнорированы симулятором, но сделают код более понятным. Они записываются в одинарных кавычках (' '). Комментарий может содержать любые символы кроме одинарных кавычек. Комментарии будут проигнорированы на этапе перевода текста в бинарный код и никак не повлияют на программу.

Пример снимка состояния симулятора

```
{
"PC":  "#0",
"ACC": "#0",
"B":   "#0",
"PSW": "#0",
"SP":  "#7",
"P0": "#0",
"P1": "#0",
"P2": "#0",
"P3": "#0",

"program": "

'Здесь комментарий'

LJMP_ad16   0   #06
LJMP_ad16   #01 #5b
MOV_ad_d    #81 #2e
LCALL_ad16  #01 #5e

MOV_a_ad  #82
JZ_rel    #03

'addr 0010' #02 0 #03

MOV_r1_d  0
MOV_a_r1
ORL_a_d   0
'addr 0018'
JZ_rel #1b

#7a 0 #90 #01 #62 #78
'addr 0020' #01 #75 #a0 0 #e4 #93 #f2 #a3
'addr 0028' #08 #b8 0 #02 #05 #a0 #d9 #f4
'addr 0030' #da #f2 #75 #a0 #ff
",

"data": "
'addr 0000' 0 0 0 0 0 0 0 0
'addr 0008' 0 0 0 0 0 0 0 0
'addr 0010' 0 0 0 0 0 0 0 0
'addr 0018' 0 0 0 0 0 0 0 0
'addr 0020' 0 0 0 0 0 0 0 0
'addr 0028' 0 0 0 0 0 0 0 0
'addr 0030' 0 0 0 0 0 0 0 0
'addr 0038' 0 0 0 0 0 0 0 0
'addr 0040' 0 0 0 0 0 0 0 0
'addr 0048' 0 0 0 0 0 0 0 0
'addr 0050' 0 0 0 0 0 0 0 0
'addr 0058' 0 0 0 0 0 0 0 0
'addr 0060' 0 0 0 0 0 0 0 0
'addr 0068' 0 0 0 0 0 0 0 0
'addr 0070' 0 0 0 0 0 0 0 0
'addr 0078' 0 0 0 0 0 0 0 0
'addr 0080' 0 #07 "
}
```

6 Сборка проекта

Сборка программы, в соответствии с заданием, осуществляется при помощи «make». Для сборки проекта необходимы библиотеки jansson и pthread.
Дополнительно требуется указать, под какой тип архитектуры процессора мы компилируем программу, little-endian или big-endian.

«make ENDIANNESS=0» - для little-endian архитектуры.
«make ENDIANNESS=1» - для big-endian архитектуры.

Опции компилятора:

```
-lm -Os -std=c11 -pedantic -Wextra -Wall -l pthread -l jansson
```

7 Проверка работоспособности симулятора

Для проверки работоспособности симулятора было написано три программы под микроконтроллер на ядре intel8051 на языке Си. Программу, выполняющую операции над числом в цикле, программу, производящую арифметические операции, и программу сортировки массива пузырьком.
Для всех трёх программ известен результат их работы и можно делать выводы, правильный ли результат выдаёт программа, запущенная на симуляторе.
Для компиляции программы был использован компилятор sdcc-3.6.0 под архитектуру intel 8051. Затем полученный файл в формате intel hex (.ihx) был переведён в бинарный файл при помощи программы avr-objcopy и запущен на симуляторе.

Программы выдают правильный результат. Это значит, что программа работает корректно в данной ситуации и ошибок не происходит.
Программы включены в проект и находятся в папке test.

8 Выводы

В данной курсовой работе был реализован симулятор работы процессора intel 8051. С его помощью можно запускать программы для микроконтроллеров на ядре 8051 и отлаживать их. Симулятор позволяет отслеживать ход исполнения инструкций в реальном времени, просматривать содержимое памяти и сохранять снимки состояния процессора в файл.
Такие возможности недоступны при работе с реальным микроконтроллером.
В данном симуляторе не реализована поддержка работы с внешними интерфейсами ввода/вывода и прерываниями, но я планирую продолжать работу над ним.

Список литературы

[1] Горюнов А.Г., Ливенцов С.Н. Архитектура микроконтроллера Intel 8051 Томск: Изд-во ТПУ, 2005. - 86 с.

[2] http://www.gaw.ru/html.cgi/txt/doc/micros/mcs51/asm/start.htm