МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ЛИПЕЦКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Институт компьютерных наук
Кафедра автоматизированных систем управления

Лабораторная работа

по микропроцессорным системам №1

“Общая структура и CPU микроконтроллера”

Студент АС-21-1 _____ Станиславчук С. М.

(подпись, дата)

Руководитель

Ст. преподаватель _____ Болдырихин О. В.

(подпись, дата)

Липецк 2024

Содержание

1. Задание, конкретизированное вариантом

Вариант 7.

Задача: Преобразование числа в код с контролем по нечётности

I/O порт: C, A

Адрес верхушки стека: 022F

2. Программа

2.1 Блок схема алгоритма программы

Блок схема (flowchart) представлена на рисунке 1.

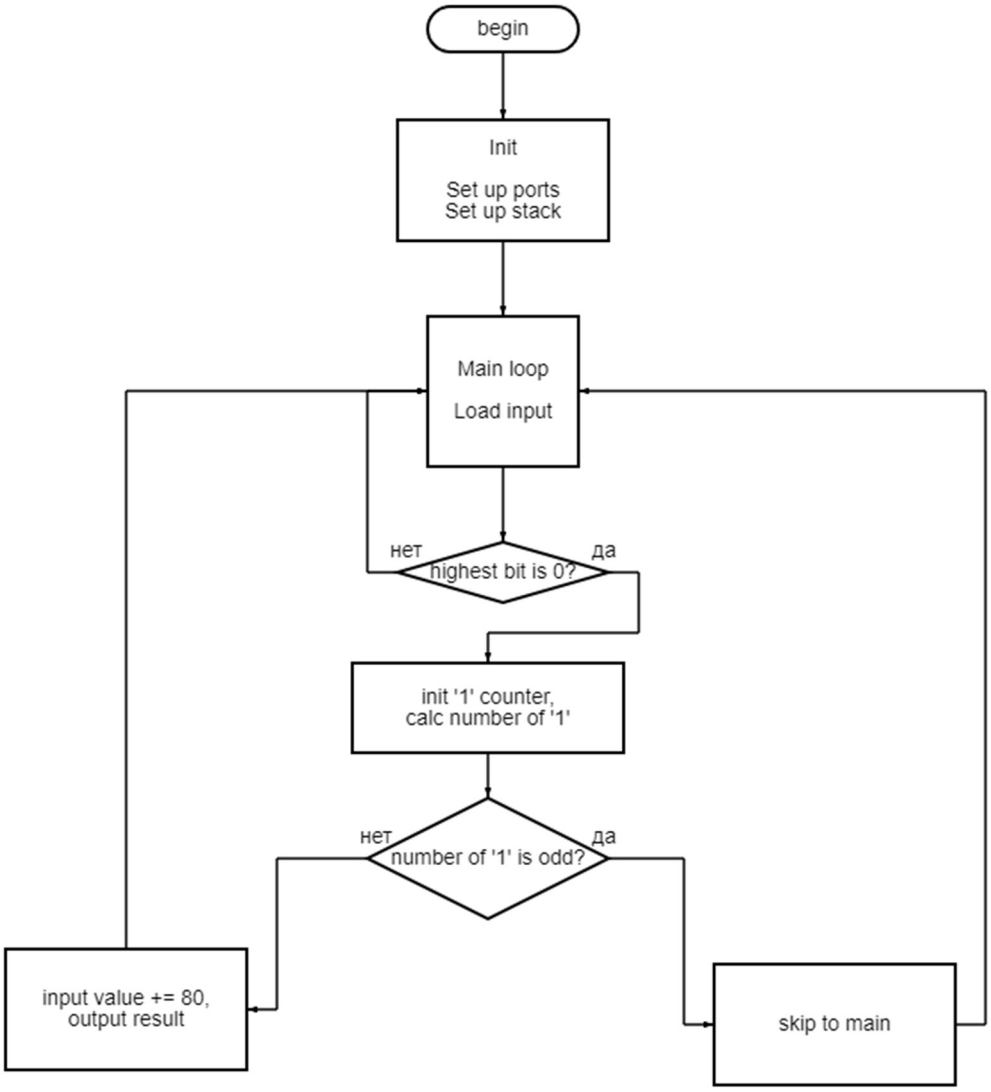

Рисунок 1. Блок-схема алгоритма программы

2.2 Ручной расчет по алгоритму

1) Предположим, у нас есть входное значение x, которое равно:

x = 0x2F

2) Проверка на четность наименьшего байта x:

Наименьший бит x равен 1 (в двоичной системе 0x101111). Это означает, что число нечетное.

3) Вывод результата:

Поскольку число нечетное, программа посчитает число единиц и на основе четности или нечетности примет решение о дальнейших инструкциях. В данном случае результат остается без изменений, но иначе, например, при x = 0x2E, мы бы выполнили преобразование и получили 128d. Результат будет выведен на порт A.

2.3 Текст программы

```
.include "m8535def.inc"
.cseg
.org 0
init:  ldi r16, 0xFF
       out DDRA, r16  ;port A to output
       ldi r16, 0
       out DDRC, r16  ;port C for input
;stack initialization
       ldi r16, low(0x022F)
       out spl, r16
       ldi r16, high(0x022F)
       out sph, r16
main:  in r16, PINC   ;entering the source number
       sbrc r16, 7    ;the highest bit of the source number must be 1
       rjmp main
       ldi r18, 0     ;initialization of the "1" counter
       rcall cnv
       out PORTA, r16 ;output of a number in a code with parity control
       rjmp main
cnv:   mov r17, r16
ccl:   sbrc r17, 0
       inc r18
       lsr r17
       brne ccl
       sbrs r18, 0
       ori r16, 0x80
       ret
```

2.4 Листинг программы

```
#endif  /* _M8535DEF_INC_ */


                .cseg
                .org 0
000000 ef0f     init:ldi r16, 0xFF
000001 bb0a          out DDRA, r16  ;port A to output
000002 e000          ldi r16, 0
000003 bb04          out DDRC, r16  ;port C for input
                ;stack initialization
000004 e20f          ldi r16, low(0x022F)
000005 bf0d          out spl, r16
000006 e002          ldi r16, high(0x022F)
000007 bf0e          out sph, r16
000008 b303     main:in r16, PINC   ;entering the source number
000009 fd07          sbrc r16, 7    ;the highest bit of the source number must be 1
00000a cffd          rjmp main
00000b e020          ldi r18, 0     ;initialization of the "1" counter
00000c d002          rcall cnv
00000d bb0b          out PORTA, r16 ;output of a number in a code with parity control
00000e cff9          rjmp main
00000f 2f10     cnv: mov r17, r16
000010 fd10     ccl: sbrc r17, 0
000011 9523          inc r18
000012 9516          lsr r17
000013 f7e1          brne ccl
000014 ff20          sbrs r18, 0
000015 6800          ori r16, 0x80
000016 9508          ret
```

3. Исследование процесса выполнения команд

3.1 Таблица с результатом исследования

№	Адрес команды (PC)	Машинный код	Ассемблерный код	Эффект	Характеристики команды		
					Флаги	Размер	Такты
1	0x000000	ef0f	ldi r16, 0xFF	PC = 0x000001 R16 = 0xFF	-	2	1
2	0x000001	bb0a	out DDRA, r16	PORTA = 0xFF	-	2	1
3	0x000002	e000	ldi r16, 0x00	R16 = 0x00	-	2	1
4	0x000003	bb04	out DDRC, r16		-	2	1
5	0x000004	e20f	ldi r16, low(0x022F)	R16=0x2F	-	2	1
6	0x000005	bf0d	out spl, r16	SP = 0x002F	-	2	1
7	0x000006	e002	ldi r16, high(0x022F)	R16 = 0x02	-	2	1
8	0x000007	bf0e	out sph, r16	SP = 0x022F	-	2	1
9	0x000008	b303	in r16, PINC		-	2	1
10	0x000009	fd07	sbrc r16, 7	R16 = 0x2F	-	2	2
11	0x00000B	e020	ldi r18, 0		-	2	1
12	0x00000C	d002	rcall cnv	SP = 0x22D	-	2	3
13	0x00000F	2f10	mov r17, r16	R17 = 0x2F	-	2	1
14	0x000010	fd10	sbrc r17, 0		-	2	1
15	0x000011	9523	inc r18	R18 = 0x01	Z, N, V	2	1
16	0x000012	9516	lsr r17	R17 = 0x17	C,S,V	2	1
17	0x000013	f7e1	brne ccl		-	2	2
18	0x000010	fd10	sbrc r17, 0			2	1
19	0x000011	9523	inc r18	R18 = 0x02	-	2	1
20	0x000012	9516	lsr r17	R17 = 0x0B	C,S,V	2	1
21	0x000010	fd10	sbrc r17, 0		-	2	1
22	0x000011	9523	inc r18	R18 = 0x03	-	2	1
23	0x000012	9516	lsr r17	R17 = 0x05	C,S,V	2	1
24	0x000010	fd10	sbrc r17, 0		-	2	1
25	0x000011	9523	inc r18	R18 = 0x04	C	2	1
26	0x000012	9516	lsr r17	R17 = 0x02	C,S,V	2	1
27	0x000010	fd10	sbrc r17, 0		-	2	1
28	0x000012	9516	lsr r17	R17 = 0x01	C,S,V	2	1
29	0x000010	fd10	sbrc r17, 0			2	1
30	0x000011	9523	inc r18	R18 = 0x05	C	2	1

31	0x000012	9516	lsr r17	R17 = 0x00	Z, C, S, V	2	2
32	0x000014	ff20	sbrs r18, 0		-	2	1
33	0x000016	9508	ret	SP = 0x022F	-	2	4
34	0x00000D	bb0b	out PORTA, r16	PORTA = 0x2F	-	2	1
35	0x00000E	cff9	rjmp main		-	2	2
ENDLESS MAIN LOOP							

3.2 Скриншот со значением результата в порту вывода

Значение результата программы в порте А представлен на рисунке 2

Рисунок 2. Значение результата программы (порт А)

4. Анализ результатов исследования (CISC x86 / RISC AVR)

4.1 Внешние сравнения

1. Система команд

X86 имеет полную систему команд, команды разной длины. Множества простых и сложных команд. Без конвейеров команды выполнялись бы за множество тактов, а не за один. Имеет большое число различных по формату и длине команд; большое число различных режимов адресации; обладает сложной кодировкой инструкции.

В AVR система команд имеет упрощенный вид. Все команды одинакового формата с простой кодировкой. Обращение к памяти происходит посредством команд загрузки и записи, остальные команды типа регистр-регистр.

2. Размер и формат команд

Все инструкции микропроцессоров CISC-архитектуры больше, чем размер одного слова. Процессору с такой архитектурой приходится иметь дело с более сложными инструкциями неодинаковой длины. Выполнение одиночной CISC-инструкции может происходить быстрее, однако обрабатывать несколько таких инструкций параллельно сложнее.

В AVR иначе: команды помещаются в одном слове. Команда, поступающая в CPU, уже разделена по полям и не требует дополнительной дешифрации.

3. Время выполнения команд

В x86 есть команды (в основном простые), которые выполняются за один такт, благодаря конвейеризации. Но все же далеко не все

Большинство команд в AVR выполняются за один такт, но не все. Так, например, выход из подпрограммы (ret) для выполнения требует 4 такта; вызов подпрограммы (rcall) требует для выполнения 3 такта.

4. Порядок изменения счетчика команд

В x86 счетчик команд считает байты, т.к. команды разной длины. И изначально процессор не знает длину команды, только после прочтения первого байта кода операции.

В AVR читаются слова (2 байта), за исключением пары 4х байтных команд, которые будут считаться 2 раза.

5. Порядок изменения указателя стека

В x86 сначала декрементируется, и по новому адресу помещается в стек очередной элемент, а при извлечении из стека происходит извлечение, а затем инкремент. Указатель стека указывает на последнюю свободную ячейку стека. Минимальный элемент, помещаемый в стек за один раз равен 2-м байтам.

В AVR по значению указателю стека помещается элемент, а затем происходит декремент. При извлечении сначала происходит инкремент, а потом извлечение. Указатель стека указывает на первую свободную ячейка стека. Минимальный элемент, помещаемый в стек за один раз равен байту.

6. Порядок хранения слов в памяти

x86 - Little endian order – младший байт данных будет расположен в начале адреса памяти, а старший байт - в его конце.

AVR - Big endian order - старший байт данных будет расположен в начале адреса памяти, а младший - в его конце

7. Регистр флагов

Регистры флагов для процессоров CISC и RISC архитектуры приведены в таблице 1.

Флаг	RISC	CISC	Описание
Zero (Z)	Да	Да	Устанавливается, если результат операции равен нулю.
Carry (C)	Да	Да	Устанавливается при переносе или заеме при арифметических операциях.
Overflow (V)	Да	Да	Устанавливается при переполнении числового диапазона.
Negative (N)	Да	Да	Устанавливается в единицу, если результат операции является отрицательным числом в двоичном представлении.
Sign (S)	Да	Да	Устанавливается в зависимости от знака результата операции.
Parity (P)	Нет	Да	Устанавливается в зависимости от четности бит в результате.
Interrupt (I)	Да	Да	Разрешение/запрещение прерываний.
Direction (D)	Нет	Да	Управление направлением обработки строк в строковых операциях.
Auxiliary (A)	Нет	Да	Помогает в операциях с дополнительным разрядом.
Trap (T)	Да	Нет	Генерирует ловушку при отладке.

4.2 Внутренние сравнения

1. Доступность и использование регистров общего назначения и ввода-вывода.

RISC:

Регистры общего назначения:

RISC-процессоры обычно имеют относительно большое количество регистров общего назначения. Эти регистры используются для хранения временных данных, адресов, результатов вычислений и других промежуточных значений (R0-R31)

Регистры общего назначения в RISC-архитектуре обычно имеют прямой доступ из большинства инструкций, что упрощает программирование и повышает производительность за счет сокращения обращений к памяти.

Регистры ввода-вывода:

В RISC-архитектуре регистры ввода-вывода могут быть частью общего пространства регистров общего назначения или быть выделенными специально для ввода-вывода.

Обычно для ввода-вывода в RISC-архитектуре используются специальные инструкции, которые могут обращаться к определенным регистрам ввода-вывода для выполнения операций ввода-вывода.

2. Размер и время выполнения команд.

Инструкции RISC обычно имеют фиксированный размер, что упрощает декодирование и выполнение, однако время выполнения команд различное.

Так, например, инструкция ldi выполняется за 1 такт, а размер равен двум байтам. Команда ret имеет точно такой же размер, но выполняется уже за 4 такта; rcall имеет размер равный 2-м байтам, а вот время выполнения команды равно 3-м тактам.

5. Выводы

В ходе выполненной работы написал ассемблерную программу для микропроцессора ATMEGA8535 (RISC), нашел основные отличия RISC и CISC архитектур.