AVR220: Bubble Sort

Features

- **14-Word Subroutine Sorts up to 255 Bytes of Data**
- **Runable Demo Program**

Introduction

This application note implements the Bubble Sort algorithm on the AVR controllers. The subroutine “bubble” sorts up to 224 bytes of SRAM data which is the SRAM area that can be reached by the lower 8 bits of a pointer.

The Bubble Sort Algorithm - Theory

The Bubble Sort algorithm is known as a quite slow and trivial algorithm for data sorting. However, for small amounts of data, the algorithm provides compact code and relatively fast sorting.

Given an array of data 1, 2, …, *n*-1, *n*, the algorithm is described as follows:

1. Compare elements *n*-1and *n*.
2. If *n*-1 is lower than *n*, swap the contents of the two array locations.
3. Repeat Steps 1 and 2 for elements *n*-2 and *n*-1. Move up one location at a time, repeat until elements 1 and 2 have been compared and possibly swapped.
4. Repeat the whole run from element *n* to 2.
5. Repeat the run from element *n* to 3.
6. …
7. Compare and, if needed, swap elements (*n*-1) and *n*.
8. When completed, the array is sorted with the highest value in location 0 and the lowest one in location *n*.

While the algorithm is executed, the higher elements move (“bubble”) through the array until they reach their final position. After the first run, the highest value finds its final position. After the second run, the second highest value finds its final position, and so on…

The total number of compare operations needed to sort an array of *n* elements is:

$$\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$$

As the total compare time grows exponentially, with the number of elements to sort, calculate the execution time first if before sorting a large number of bytes.

In Pseudo-code, the Bubble Sort algorithm would be as follows:

```
for i=n downto 1 do
begin
for j=n downto i
  begin
      if A(n-1)<A(n)
      swap (A(n-1),A(n))
  end
end
```

To reverse the sort order, replace the “<” sign with “>”.

8-Bit AVR Microcontroller

Application Note

0939A-A–8/97

Implementation

Usage

The subroutine “bubble” is used according to the following procedure:

1. Load “endH:endL” with the address of last element in the array.
2. Load the loop counter “cnt1” with the size of the data array - 1.
3. Call “bubble”.

If preferred, the routine will work fine using the Y-pointer instead.

Algorithm Description

The following procedure describes how the sorter is implemented on the AVR:

1. Copy “cnt1” to “cnt2”.
2. Copy “endH:endL” to “Z”
3. Load register variable “A” with the byte at Z.
4. Decrement Z and load register variable “B” with the byte at Z.
5. If A<B, store “A” at Z and “b” at Z+1 (swap bytes).
6. Decrement “cnt2”
7. If not zero, goto Step 2.
8. Decrement “cnt1”
9. If not zero, goto Step 1.

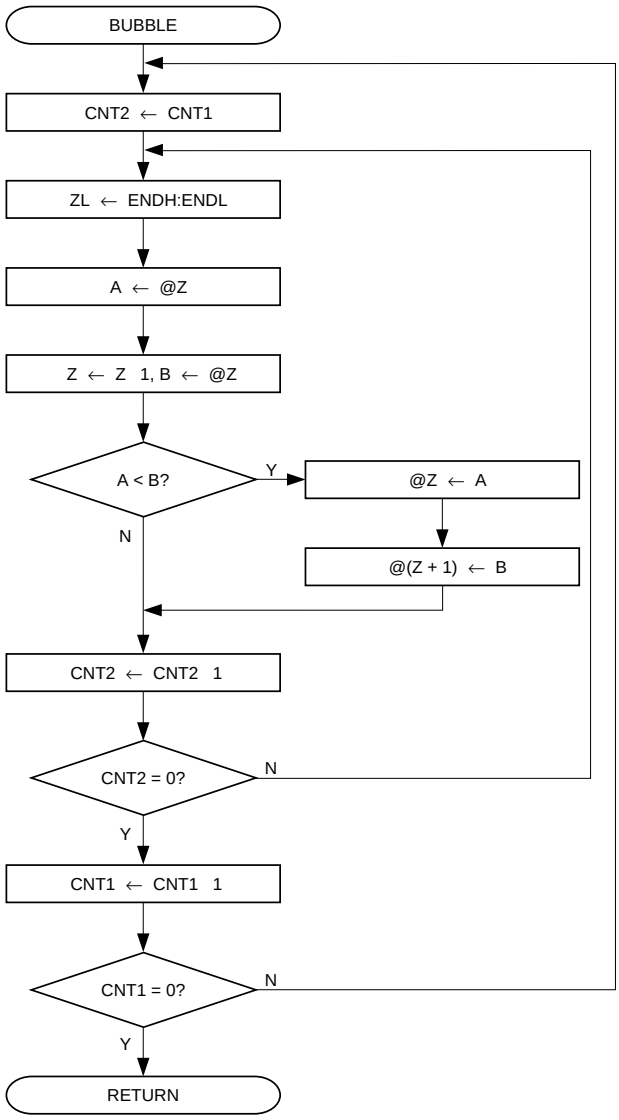

Figure 1. “bubble” Flow Chart

Performance

Table 1. "bubble" Register Usage

Register	Input	Internal	Output
R13		"A" - first value to compare	
R14		"B" - second value to compare	
R15		"cnt2" - inner loop counter	
R16	"cnt1" - # of bytes to sort - 1	"cnt1"- outer loop counter	
R17	"endL" - low address of last element		
R18	"endH" - high address of last element		
R30		ZL	
R31		ZH	

Table 2. "bubble" Performance Figures

Parameter	Value
Code Size (Words)	12 + return
Average Execution Time (Cycles)	5 x (SIZE-1) +11.5 x (SIZE(SIZE-1)) + return
Register Usage	• Low registers :None • High registers :2 • Pointers :Z
Interrupts Usage	None
Peripherals Usage	None

Note: SIZE = Number of bytes to sort

Test/Example Program

"avr220.asm" contains a test program which copies 60 bytes of random data from the program memory to SRAM and calls "bubble" to sort the data. The test program is well suited for running under the AVR Studio. To get a feeling for how the data "bubbles" through the array, place data a breakpoint somewhere in the inner loop and run single loop cycles while watching the SRAM memory window.