AVR204: BCD Arithmetics

8-Bit AVR MCU with Downloadable Flash

Application Note

Features

- **Conversion 16 Bits ↔ 5 Digits, 8 Bits ↔ 2 Digits**
- **2-Digit Addition and Subtraction**
- **Superb Speed and Code Density**
- **Runable Example Program**

Introduction

This application note lists routines for BCD arithmetics. A listing of all implementations with key performance specifications is given in Table 1.

Table 1. Performance Figures Summary

Application	Code Size (Words)	Execution Time (Cycles)
16-bit binary to 5-digit BCD conversion	25	760
8-bit binary to 2-digit BCD conversion	6	28
5-digit BCD to 16-bit binary conversion	30	108
2-digit BCD to 8-bit binary conversion	4	26
2-digit packed BCD addition	19	19
2-digit packed BCD subtraction	13	15

16-Bit Binary to 5-digit BCD Conversion - “bin2BCD16”

This subroutine converts a 16-Bit binary value to a 5-digit packed BCD number. The implementation is Code Size optimized. This implementation uses the Z-pointer's auto-decrement facility, and can not be used as is for the AT90Sxx0x series. To use this routine on an AT90Sxx0x, add an “INC ZL” instruction where indicated in the file listing.

Algorithm Description

“bin2BCD16” implements the following algorithm:

1. Load loop counter with 16.
2. Clear all three bytes of result.
3. Shift left input value low byte.
4. Shift left carry into input value high byte.
5. Shift left carry into result byte 0 (two least significant digits).
6. Shift left carry into result byte 1.
7. Shift left carry into result byte 2 (most significant digit).
8. Decrement loop counter
9. If loop counter is zero, return from subroutine.
10. Add $03 to result byte 2.
11. If bit 3 is zero after addition, restore old value of byte 2.
12. Add $30 to result byte 2.
13. If bit 7 is zero after addition, restore old value of byte 2.
14. Add $03 to result byte 1.
15. If bit 3 is zero after addition, restore old value of byte 1.
16. Add $30 to result byte 1.
17. If bit 7 is zero after addition, restore old value of byte 1.
18. Add $03 to result byte 0.
19. If bit 3 is zero after addition, restore old value of byte 0.
20. Add $30 to result byte 0.
21. If bit 7 is zero after addition, restore old value of byte 0.

0938A-A–8/97

22. Goto Step 3.

In the implementation. Steps 10-21 are carried out inside a loop, where the Z-pointer is used for successive access of all three bytes of the result. This is shown in the flow chart below.

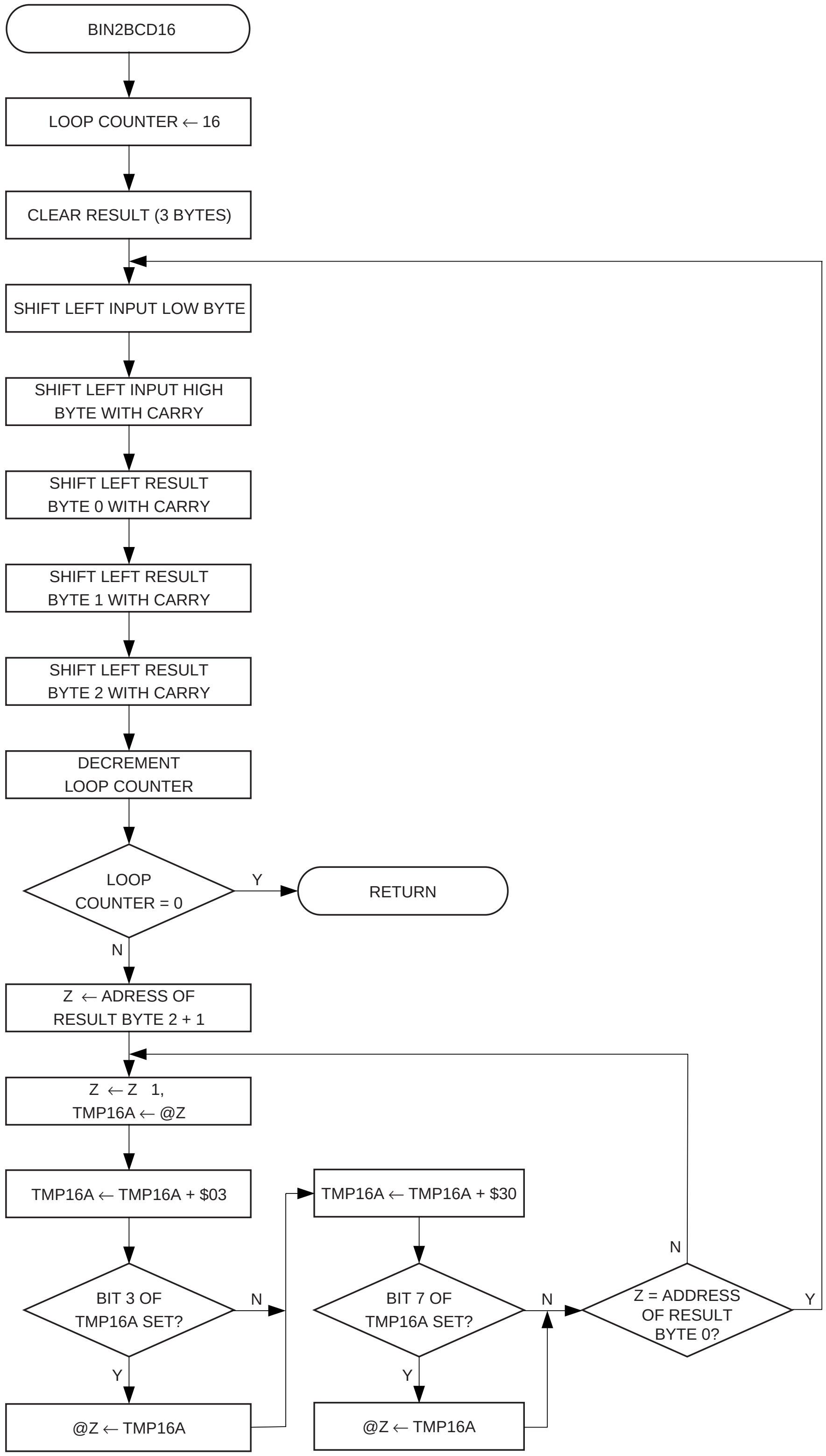

Figure 1. "bin2BCD16" Flow Chart

Usage

1. Load the 16-bit register variable “fbinH:fbinL” with the 16-bit number to be converted. (High byte in “fbinH”)
2. Call “bin2BCD16”.
3. The result is found in the 3-byte register variable “fBCD2:fBCD1:fBCD0” with MSD in the lower nibble of “fBCD2”.

Performance

Table 2. “bin2BCD16” Register Usage

Register	Input	Internal	Output
R13			“fBCD0” - BCD digits 1 and 0
R14			“fBCD1” - BCD digits 2 and 3
R15			“fBCD2” - BCD digit 4
R16	“fbinL” - binary value low byte		
R17	“fbinH” - binary value high byte		
R18		“cnt16a” - loop counter	
R19		“tmp16a” - temporary storage	
R30		ZL	
R31		ZH	

Table 3. “bin2BCD16” Performance Figures

Parameter	Value
Code Size (Words)	25
Average Execution Time (Cycles)	760
Register Usage	• Low registers :3 • High registers :4 • Pointers :Z
Interrupts Usage	None
Peripherals Usage	None

8-Bit Binary to 2-digit BCD Conversion - “bin2BCD8”

This subroutine converts an 8-Bit binary value to a 2-digit BCD number. The implementation does not generate a packed result, i.e. the two digits are represented in two separate bytes. To accomplish this, some smaller modifications must be done to the algorithm as shown in the following section.

Algorithm Description

“bin2BCD8” implements the following algorithm:

1. Clear result MSD.
2. Subtract 10 from the 8-bit input number.
3. If result negative, add back 10 to 8-bit input number and return
4. Increment result MSD and goto step 2.

LSD of the result is found in the same register as the input number. If a packed result is needed, make the following changes to the algorithm:

- Instead of incrementing MSD in Step 4, add $10 to MSD.
- After adding back 10 to the input number in step 3, add LSD and MSD together.

Where to make these changes is indicated in the program listing.

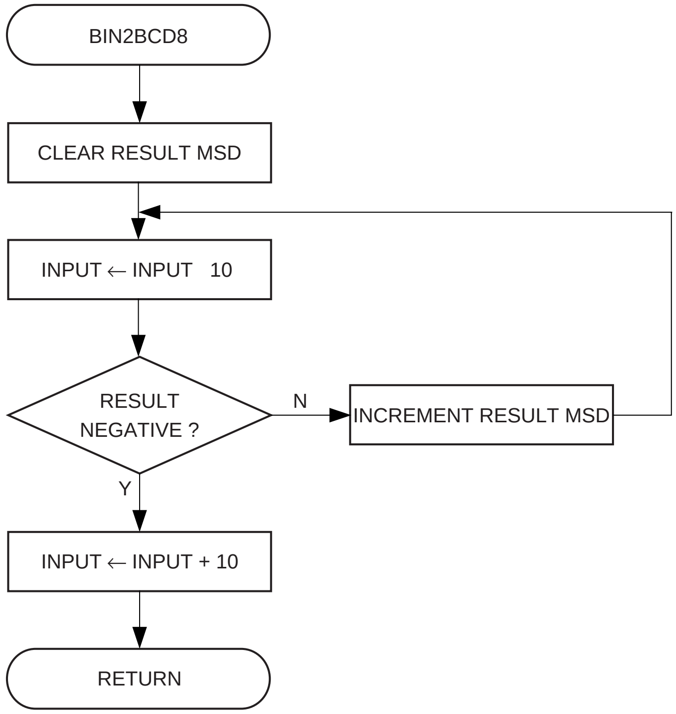

Figure 2. "bin2BCD8" Flow Chart

Usage

1. Load the register variable "fbin" with the value to be converted.
2. Call "bin2BCD8".
3. The result MSD and LSD is found in "fBCDH" and "fBCDL", respectively.

Performance

Table 4. "bin2BCD8" Register Usage

Register	Input	Internal	Output
R16	"fbin" - binary value		"tBCDL" - LSD of result
R17			"tBCDH" - MSD of result

Table 5. "bin2BCD8" Performance Figures

Parameter	Value
Code Size (Words)	6 + return
Average Execution Time (Cycles)	28
Register Usage	• Low registers :None • High registers :2 • Pointers :None
Interrupts Usage	None
Peripherals Usage	None

5-Digit BCD to 16-Bit BCD Conversion - "BCD2bin16"

This subroutine converts a 5-digit packed BCD number to a 16-Bit binary value.

Algorithm Description

Let *a, b, c, d, e* denote the 5 digits in the BCD number (*a* = MSD, *e* = LSD). The result is generated by computing the following equation:

$$10(10(10(10a + b) + c) + d) + e$$

The four times repeated operation "multiply-by-10-and-add" is implemented as a subroutine. This subroutine takes the 16-bit register variable "mp10H:mp10L" and the register variable "adder" as input parameters. The subroutine can be called by two separate addresses, "mul10a" and "mul10b". The difference is summarized as follows:

"mul10a" - multiplies "mp10H:mp10L" and adds the *higher* nibble of "adder".

"mul10b" - multiplies "mp10H:mp10L" and adds the *lower* nibble of "adder".

The subroutine implements the following algorithm.

1. Swap high/low nibble of "adder"
2. Make a copy of "mp10H:mp10L".
3. Multiply original by 2 (Shift left).
4. Multiply copy by 8 (Shift left x 3).
5. Add copy and original.
6. Clear upper nibble of "adder"
7. Add lower nibble of "adder"
8. If carry set, increment "mp10H".

When calling "mul10b", Step 1 is omitted.

The main routine follows this algorithm:

1. Clear upper nibble of BCD byte 2 (MSD)
2. Clear "mp10H"
3. "mp10H" ← BCD byte 2
4. "adder" ← BCD byte 1
5. Call "mul10a"
6. "adder" ← BCD byte 1
7. Call "mul10b"
8. "adder" ← BCD byte 0
9. Call "mul10a"
10. "adder" ← BCD byte 0
11. Call "mul10b"

Usage

1. Load the 3-byte register variable "fBCD2:fBCD1:fBCD0" with the value to be converted (MSD in the lower nibble of "fBCD2").
2. Call "BCD2bin16".
3. The 16-bit result is found in "tbinH:tbinL".

Performance

Table 6. "BCD2bin16" Register Usage

Register	Input	Internal	Output
R12		"copyL" - temporary value used by "mul10a/mul10b"	
R13		"copyH" - temporary value used by "mul10a/mul10B"	
R14		"mp10L" - low byte of input to be multiplied by "mul10a/mul10b"	"tbinL" - low byte of 16-bit result
R15		"mp10H" - high byte of input to be multiplied by "mul10a/mul10b"	"tbinH" - high byte of 16-bit result
R16	"fBCD0" - BCD digits 1 and 0		
R17	"fBCD1" - BCD digits 2 and 3		
R18	"fBCD2" - BCD digit 4		

Table 7. "BCD2bin16" Performance Figures

Parameter	Value
Code Size (Words)	30
Execution Time (Cycles)	108
Register Usage	• Low registers :4 • High registers :4 • Pointers :None
Interrupts Usage	None
Peripherals Usage	None

2-digit BCD to 8-Bit Binary Conversion - "BCD2bin8"

This subroutine converts a 2-digit BCD number to an 8-Bit binary value. The implementation does not accept a packed BCD input, i.e. the two digits must be represented in two separate bytes. To accomplish this, some modifications will have to be made to the algorithm as shown in the following section.

Algorithm Description

"BCD2bin8" implements the following algorithm:

1. Subtract 1 from input MSD.
2. If result negative, return.
3. Add 10 to 8-bit result/input LSD.
4. Goto Step 1.

The result is found in the same register as the input number LSD. To make the algorithm accept a packed BCD input, use this algorithm:

1. Copy BCD input to result.
2. Clear higher nibble of result
3. Subtract $10 from input MSD.
4. If half carry set, return.
5. Add decimal 10 to result.
6. Goto Step 3.

The program listing shows how and where to make the changes. Figure 3 shows the flowchart which applies to the non-packed input implementation.

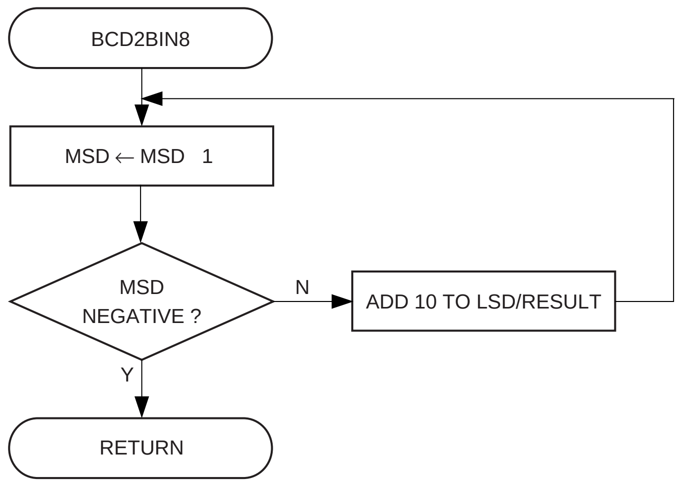

Figure 3. "BCD2bin8" Flow Chart

Usage

1. Load the register variables "fBCDH" and "fBCDL" with the input MSD and LSD, respectively.
2. Call "BCD2bin8".
3. The 8-bit result is found in "tbin".

Performance

Table 8. "BCD2bin8" Register Usage

Register	Input	Internal	Output
R16	"fBCDL" - LSD of BCD input		"tbin" - 8-bit of result
R17	"fBCDH" - MSD of BCD input		

Table 9. "BCD2bin8" Performance Figures

Parameter	Value
Code Size (Words)	4 + return
Average Execution Time (Cycles)	26
Register Usage	• Low registers :None • High registers :2 • Pointers :None
Interrupts Usage	None
Peripherals Usage	None

2-Digit Packed BCD Addition - "BCDadd"

This subroutine adds two 2-digit packed BCD numbers. The output is the sum of the two input numbers, also as 2-digit packed BCD, and any overflow carry.

Algorithm Description

"BCDadd" implements the following algorithm:

1. Add the values binary.
2. If half carry set, set BCD carry, add 6 to LSD and Goto Step 5.
3. Clear BCD carry and add 6 to LSD.
4. If half carry clear after adding 6, LSD ≤ 9, so restore LSD by subtracting 6.
5. Add 6 to MSD.
6. If carry set, MSD > 9, so set BCD carry and return.
7. If carry was set during Step 1, restore MSD by subtracting 6.

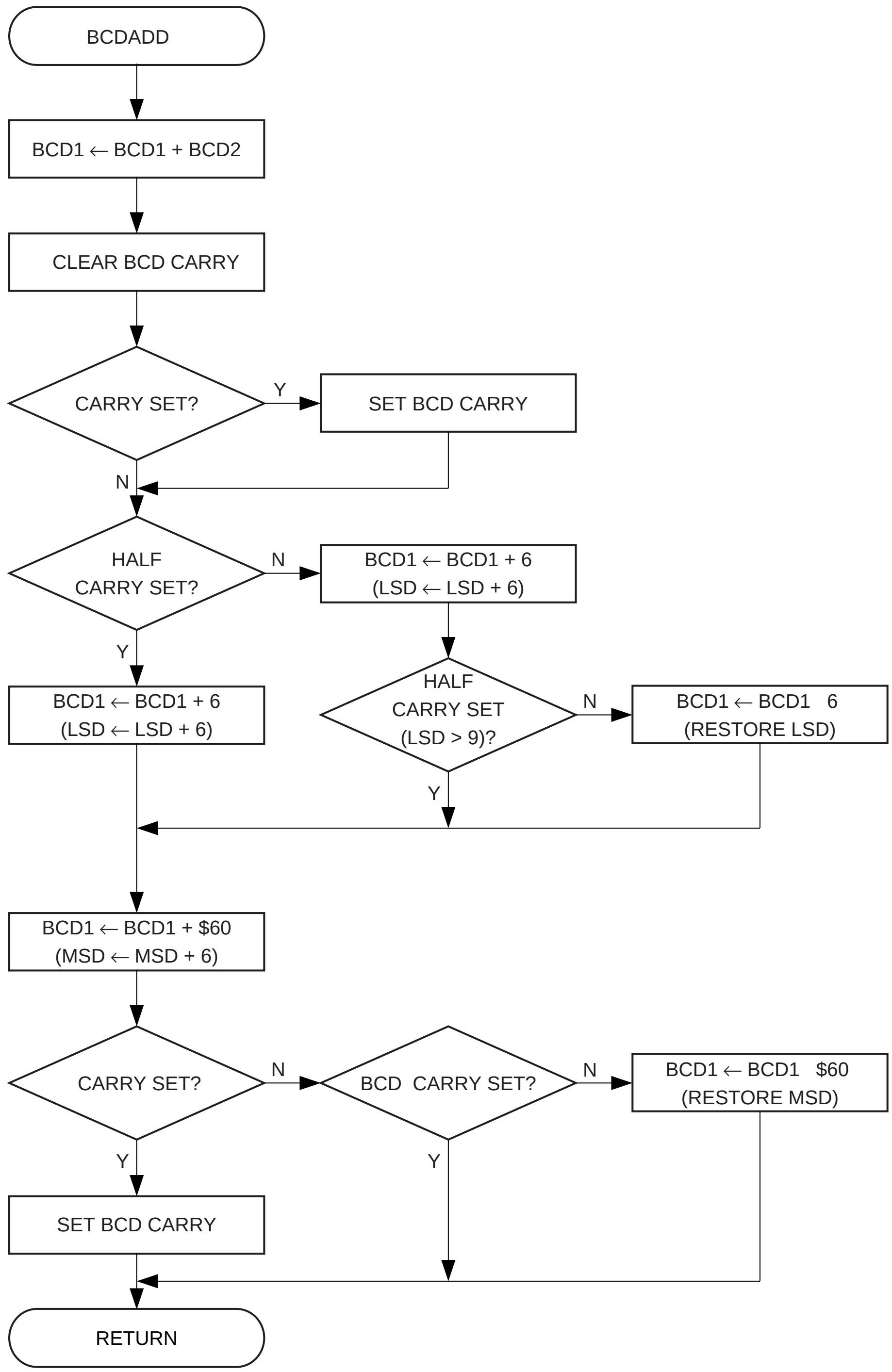

Figure 4. “BCDadd” Flow Chart

Usage

1. Load the register variables “BCD1” and “BCD2” with the numbers to be added.
2. Call “BCDadd”.
3. The BCD sum is found in “BCD1” and the overflow carry in “BCD2”.

Performance

Table 10. "BCDadd" Register Usage

Register	Input	Internal	Output
R16	"BCD1" - BCD number 1		"BCD1" - BCD result
R17	"BCD2" - BCD number 2		"BCD2" - overflow carry
R18		"tmpadd" - holds values $06 and $60 to be added	

Table 11. "BCDadd" Performance Figures

Parameter	Value	
Code Size (Words)	19	
Average Execution Time (Cycles)	19	
Register Usage	• Low registers • High registers • Pointers	:None :3 :None
Interrupts Usage	None	
Peripherals Usage	None	

2-Digit Packed BCD Subtraction - "BCDsub"

This subroutine subtract two 2-digit packed BCD numbers. The output is the difference of the two input numbers, also as 2-digit packed BCD, and any underflow carry.

Algorithm Description

"BCDadd" implements the following algorithm:

1. Add the values binary.
2. If carry set, set BCD carry
3. If half carry set, subtract 6 from LSD.
4. If BCD carry clear, return.
5. Subtract 6 from MSD and set BCD carry.
6. If carry set, set BCD carry.

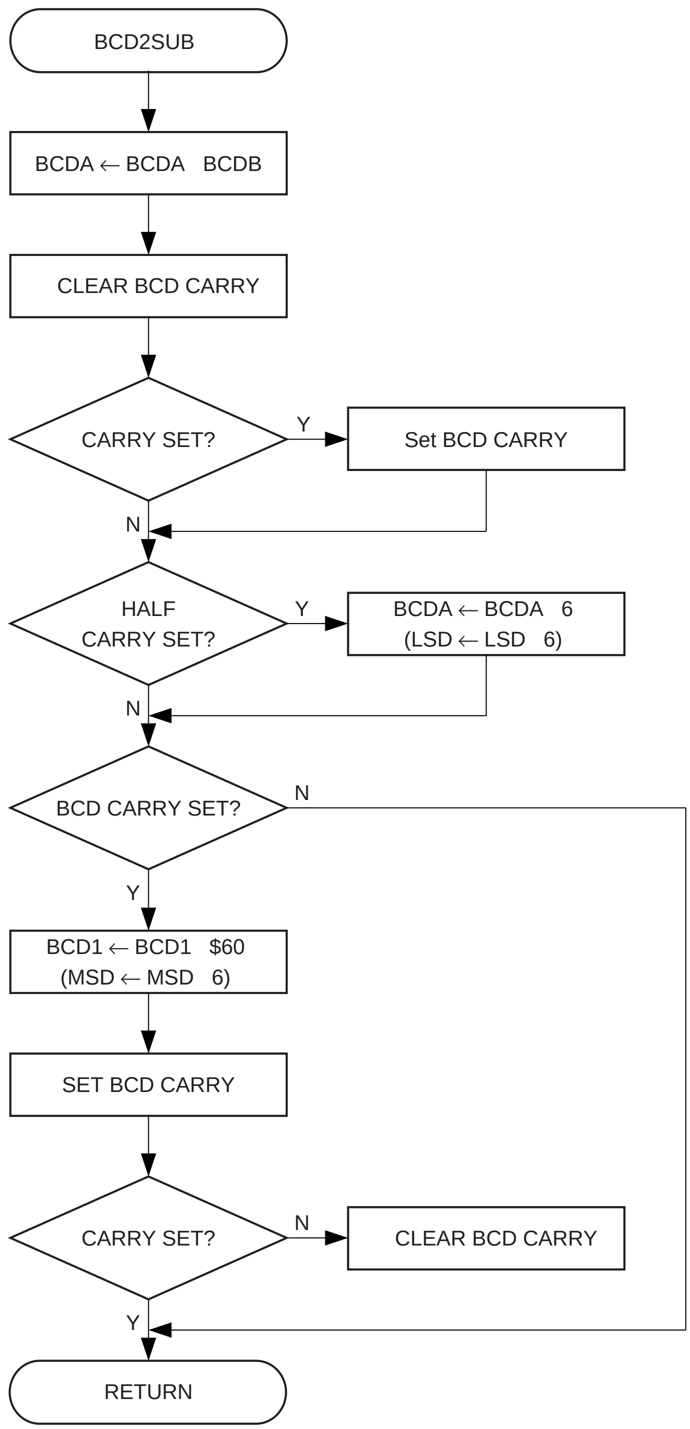

Figure 5. "BCDsub" Flow Chart

Usage

1. Load the register variable "BCDa" with the number to be subtracted and "BCDb" with the number subtract.
2. Call "BCDsub".
3. The BCD sum is found in "BCDa" and the underflow carry in "BCDb".

Performance

Table 12. "BCDadd" Register Usage

Register	Input	Internal	Output
R16	"BCDa" - BCD number to subtract from		"BCD1" - BCD result
R17	"BCDb" - BCD number to subtract		"BCD2" - underflow carry

Table 13. "BCDadd" Performance Figures

Parameter	Value
Code Size (Words)	13
Average Execution Time (Cycles)	15
Register Usage	• Low registers :None • High registers :2 • Pointers :None
Interrupts Usage	None
Peripherals Usage	None