RCA MICROPROCESSOR 1800

Binary Arithmetic Subroutines for RCA COSMAC Microprocessors

MPM-206

Suggested Price $5.00

RCA 1800
MICROPROCESSOR

Binary Arithmetic Subroutines for RCA COSMAC Microprocessors

MPM-206

Suggested Price $5.00

Binary Arithmetic Subroutines for RCA COSMAC Microprocessors

RCA | Solid State Division | Somerville, N. J. 08876

Printed in USA/6-76

Foreword

The RCA CDP1801 and CDP1802 COSMAC Microprocessors are CMOS 8-bit register-oriented central processing units. They are suitable for use in a wide range of stored-program computer systems and products. Often, their applications require extended precision arithmetic calculations with possible interfacing of the system to BCD-oriented peripheral hardware. For such applications, this Manual provides a set of 16-bit 2's complement arithmetic subroutines designed to be operated on the RCA COSMAC Microprocessors. Versions are supplied for use with either the RCA CDP1801 or CDP1802. A suitable selection from this set of subroutines may be made for the calculations required in a specific application.

The subroutines described in this Manual are also available in source language on a floppy diskette CDP18S826 for use with the RCA Floppy Disk System CDP18S800.

In this Manual, the 29 Binary Arithmetic Subroutines are described, first in general and then in detail. A Standard Call and Return Technique for facilitating their use is provided along with complete listings. Additionally, BCD-to-binary and binary-to-BCD conversions are described. As a tutorial aid, a sample program is also provided.

Users requiring additional information on the available hardware, firmware, and software support systems should refer to the COSMAC technical literature. Reference should be made to the device technical data sheets and to the following publications:

MPM-201	User Manual for the RCA CDP1802 COSMAC Microprocessor
MPM-202	Timesharing Manual for the RCA CDP1802 COSMAC Microprocessor
MPM-203	Evaluation Kit Manual for the RCA CDP1802 COSMAC Microprocessor
MPM-208	Operator Manual for the RCA COSMAC Development System
MPM-101	User Manual for the CDP1801 COSMAC Microprocessor
MPM-102	Program Development Guide for the CDP1801 COSMAC Microprocessor (Timesharing)

Table of Contents

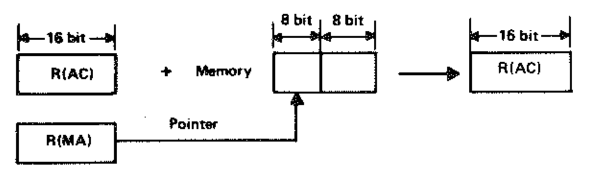
Binary Arithmetic Subroutines

Introduction

The Binary Arithmetic Subroutine Package given in this Manual is a set of 16-bit 2's complement arithmetic subroutines designed to be operated on a COSMAC microprocessor. The subroutines are coded in Level-I assembly language and require 1K-byte of memory space. These subroutines do not alter themselves and can be stored in Read-Only Memory. Different versions are available for the CDP1801 and CDP1802. The subroutines may be used individually or in any combination.

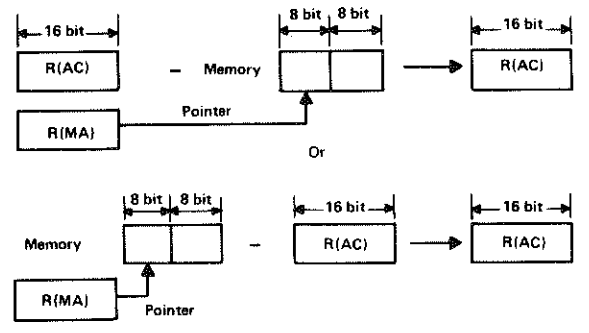
Four arithmetic functions are included:

1. **16-bit 2's complement Addition:**



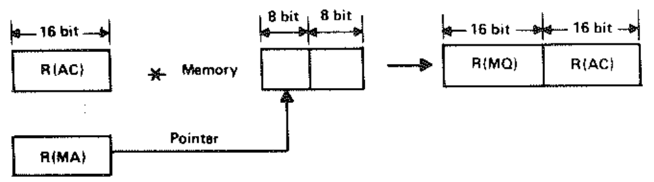
92CS-28092

2. **16-bit 2's complement Subtraction:**



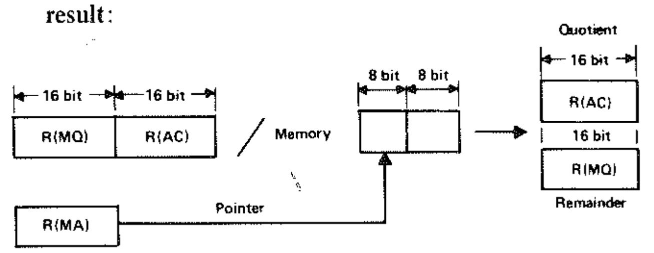
92CS-28089

3. **16-bit 2's complement Multiplication which yields 32-bit result:**



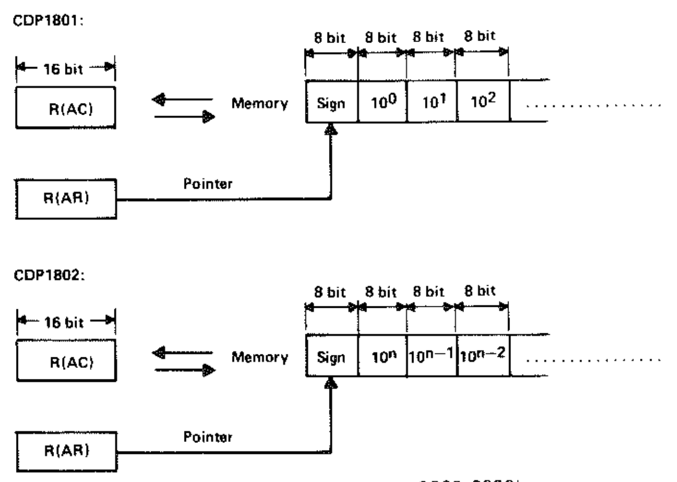
92CS-28090

4. **32-bit 2's complement Division which yields 16-bit result:**



92CS-28088

BCD-to-binary and binary-to-BCD conversions together with utility routines (which can be used to save or restore the contents of registers) are provided.



92CS-28091

Applications

Various applications of these subroutines are possible. For example, a user may enter BCD numbers from external devices, manipulate these numbers in 2's complement arithmetic functions, and finally output the result in BCD form back to external devices. A diagram of such an application is given in Fig. 1.

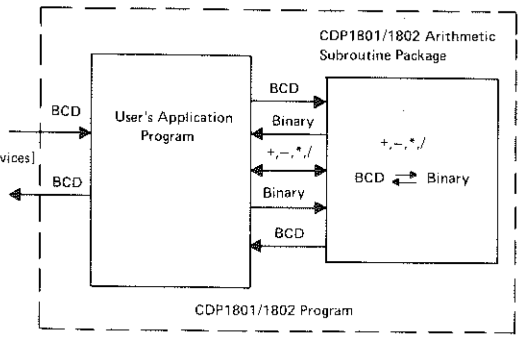


Fig. 1 – Example of microprocessor system using the arithmetic subroutine package.

It is possible to use the arithmetic function(s) only, as diagrammed in Fig. 2, or a combination of these subroutines to suit the user's needs.

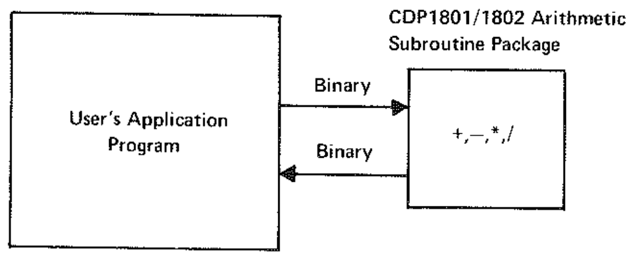


Fig. 2 – Application using arithmetic functions only.

Basic Conventions

All of the subroutines in this package follow the call and return conventions described in the **User Manual for the RCA CDP1802 COSMAC Microprocessor** (MPM-201). A stack area in RAM is also implied. Symbolic register names are used to permit custom modifications and register allocation. The standard registers' definitions and their functions are:

R(SP) = R2	.. The stack pointer.
R(PC) = R3	.. The program counter used by all the subroutines.
R(CALL) = R4	.. Pointer to CALL routine.
R(RETN) = R5	.. Pointer to RETURN routine.
R(LINK) = R6	.. Pointer to arguments passed from main program.
R(AR) = RA	.. Pointer to BCD digits in memory. Used in BCD conversion routines.
R(NR) = RB	.. Counter for the number of BCD digits. Used in BCD conversion routines.
R(CR) = RC	.. Scratchpad and counter. Used in multiply, divide and others.
R(MA) = RD	.. Pointer to the address of one of the two operands in add, subtract, multiply and divide operations.
R(MQ) = RE	.. 16-bit accumulator or sign extension. Used in multiplication and division to hold the most significant half of the product and dividend. It is also used to store the remainder after a divide operation.
R(AC) = RF	.. 16-bit accumulator. Used as one of the two operands for all computations.

Subroutine Categories

A list of the binary arithmetic subroutines by category together with a description of their functions follows.

The secondary subroutines shown are used internally by the other subroutines only. Their functions, therefore, will not be described in further detail. For the interested user, commented coding is provided on the actual listing of the binary arithmetic subroutine package. These secondary routines are relatively short and self-explanatory.

I. ADDITION

NAME	ARGUMENT	FUNCTION	
ADDOP	aaaa	R(AC)+M(OPERAND POINTER)	→ R(AC)
ADD	- - -	R(AC)+OPERAND	→ R(AC)
ADDCON	nnnn	R(AC)+CONSTANT	→ R(AC)
ADDST	- - -	R(AC)+two bytes on stack	→ R(AC)

II. SUBTRACTION

SDOP	aaaa	M(OPERAND POINTER)–R(AC)	→ R(AC)
SD	- - -	OPERAND–R(AC)	→ R(AC)
SDCON	nnnn	CONSTANT–R(AC)	→ R(AC)
SMOP	aaaa	R(AC)–M(OPERAND POINTER)	→ R(AC)
SM	- - -	R(AC)–OPERAND	→ R(AC)

III. MULTIPLICATION

MPYOP	aaaa	R(AC)*M(OPERAND POINTER)	→ [R(MQ);R(AC)]
MPY	- - -	R(AC)*OPERAND	→ [R(MQ);R(AC)]

IV. DIVISION

DIVOP	aaaa	[R(MQ);R(AC)]/M(OPERAND POINTER) → R(AC)	
DIV0	- - -	zero divide check and sign extension	
DIV	- - -	quotient overflow check	
DIVQ	- - -	[R(MQ);R(AC)]/OPERAND	→ R(AC)

V. BINARY-CODED-DECIMAL NUMBER CONVERSION

CBD	aaaa,nn	Binary in R(AC) → BCD
CDB	aaaa,nn	BCD → Binary in R(AC)

VI. UTILITY ROUTINES

PUSHAC	- - -	R(AC) → stack
PUSHCQ	- - -	R(MQ),R(MA),R(CR) → stack
POPAC	- - -	stack → R(AC)
POP	- - -	stack → discard
POPCQ	- - -	stack → R(MQ),R(MA),R(CR)
LOADOP	aaaa	M(OPERAND POINTER) → R(AC)
LOAD	- - -	OPERAND → R(AC)
LODCON	nnnn	CONSTANT → R(AC)
STOROP	aaaa	R(AC) → M(OPERAND POINTER)
STORE	- - -	R(AC) → OPERAND
COMPOP	aaaa	If R(AC) ⩾M(OPERAND POINTER) then 1 → DF
COMP	- - -	If R(AC) ⩾OPERAND then 1 → DF
TEST	- - -	compare R(AC) with 0
SWAPAQ	- - -	R(AC) ⇄ R(MQ)

VII. SECONDARY ROUTINES

DABS*	ABS[R(MQ)] ABS(OPERAND) → R(MQ)
DSM*	entry point of DABS
DVA*	entry point of DABS
DSHL*	MQ*2 → MQ
DZS	clear memory locations for BCD

NOTES:
aaaa is two bytes address of operand.
nnnn is two bytes constant literal.
* found in CDP1801 version only.

Calling Sequence

The calling sequences of the subroutines are similar; the only differences are the nature of their arguments.

Some routines deal implicitly with an operand on the stack, or with an operand pointed to by the address register. Some fetch the address of the operand from the call sequence argument list, and others fetch the actual argument from the call sequence. Typically, a routine which fetches the address of an operand is used if the operand is a variable. On the other hand, a routine that fetches the actual argument is used if the operand is a constant. For instance, if the function A=B/C+5 is to be performed, the calling sequence of the divide and add routines could be:

```
SEP CALL
,A(DIVOP)       . . assume [R(MQ);R(AC)] con-
                    tain the variable B.
,A(C)           . . address of the variable C in
                    memory.
                . . now R(AC) contains the quo-
                    tient of the
                . . operation B/C.
SEP CALL
,A(ADDCON)
,X'0005'        . . the constant 5 is to be added
                    to the
                . . quotient of B/C stored in
                    R(AC) previously.
                . . now R(AC) contains B/C+5
```

Algorithms

The add and subtract routines use ordinary 2's complement add and subtract methods. For multiplication, the add-shift algorithm is implemented. The divide routine uses the nonrestoring method. These algorithms can be found in computer arithmetic books. For example, see **Digital Computer Design Fundamentals** by Yaohan Chu, McGraw-Hill Book Company.

The BCD-to-binary conversion is accomplished by taking the digits in order, starting with the most significant, and adding them to ten times the previous partial result—successively.

The binary-to-BCD conversion for the CDP1801 is implemented in the reverse order. The binary number in register AC is divided by ten successively until R(AC) becomes zero. On each iteration, the remainder is the corresponding BCD digit starting from the least significant. DZS is an internal subroutine used to initialize all the digits to zero.

The implementation of the binary-to-BCD conversion for the CDP1802 is faster than that for the CDP1801. It is described in the book referenced above.

Meaning of DF

The add, subtract, divide, BCD-to-binary conversion, and binary-to-BCD conversion routines affect DF. The DF is set if the result has overflowed. Since no overflow condition is possible for multiplication, the multiply routine sets DF if the product is greater than 16 bits. Some of the subroutines also use DF to indicate the terminal condition of an operation. For example, the compare subroutine sets DF if the content of R(AC) is greater than or equal to the operand pointed to by register MA. Thus, it is assumed and is essential that the subroutine linkage routines do not alter the state of DF. The "Standard Subroutine Call and Return Technique" is assumed. Appendix A contains a listing typical of these programs.

Utility Routines

Four of the general registers R(AC), R(MQ), R(MA), and R(CR), are used by various routines. For programs that need these registers, special routines have been included which save them on the stack, and later restore them. Accordingly, the register save and restore routines are completely independent. In particular, these routines are partitioned into two sets: routines to push and pop the accumulator, and routines to push and pop the other three registers as a block. The separate routines for the accumulator permit intermediate results to be saved on the stack for later use in computation. In addition, a routine is provided to swap or exchange the accumulator with the accumulator extension, which is used in multiplication and division.

It is assumed that the stack pointer normally points to one byte above the first of two bytes stored by the subroutine call operation (i.e., the last pushed, having the lower memory address), and that these two bytes are normally popped off by the return operation. Note that this assumption is consistent with the standard subroutine linkage conventions. Thus, to save data on the stack, the data must be inserted beneath those first two bytes, so that when they are popped off, the saved data will be on top. Similarly, when the data is to be recovered, it must be fetched from beneath the top two bytes, then the gap closed up. One of the entries to the binary add routine also permits using the virtual top of the stack as one operand (that is, the top of the stack as seen by the calling program).

Defined Options

Some of the lines in the listing have one or more flags on the right margin. They consist of a symbol enclosed within parentheses. These symbols are used to flag "Defined Options" which are specified in the notes to follow. Usually to invoke a particular option, all of the lines flagged by the same character are altered in some specified way.

(@) Separate Stack

While it is expected that the user will utilize the standard subroutine linkage, there is no requirement that the stack pointer used for the binary arithmetic be the same as that for the subroutine linkage. Since some of these routines do not themselves make use of the call linkages, it is not always necessary to use a stack for the calls. If a stack is not needed, subroutines PUSHCQ, PUSHAC, POPCQ and POPAC are inappropriate. If their functions are desired, the routines must be rewritten omitting the operation of slicing off the top of the stack and working under it (actually this results in the routines being trivially simple, so the user should have no difficulty recoding them). If the arithmetic from the top of the stack is still a desirable feature, the lines flagged by (@) must be deleted, so that R(MA) will point to the actual top of the stack.

(0) 16-Bit Dividend

By retaining the lines so flagged, and deleting the lines flagged (/), the divide routine will assume that all dividends fit entirely into the R(AC), and that error conditions can occur only if the divisor is zero.

(/) 32-Bit Dividend

By retaining the lines so flagged, and deleting the lines flagged (0), the divide routine will assume that all dividends occupy the full 32 bits of the R(AC) and R(MQ), and that a divide fault is possible only if the quotient result of the dividend divided by the divisor would exceed the 16 bits of the R(AC), or if the divisor is zero (which is equivalent to an infinite quotient).

(0) (/) No Divide Check

By deleting all the lines flagged by either (0) or (/), the divide routine will perform no testing on the possibility of a divide fault. This deletion poses the simple hazard that the numbers resulting from the operation will be meaningless if a divide fault occurs, and no indication of the error will be returned to the calling routine. Note that in this option, the dividend is assumed to be 32 bits.

If the user needs to make alterations to the routines, or if some are omitted, their placement in memory may need changing to eliminate possible branches out of page.

Timing

Timing measurements, given in Fig. 3, were taken on the arithmetic and code conversion subroutines. For the CDP1801 version, the speed is limited by the size of the program since the objective is to fit the total package into 1-K of memory. However, with the CDP1802 instruction set, it is possible to meet this memory space requirement while optimizing for speed. **All** the **measurements** were based on a clock rate of 2 MHz. The actual timing can be rescaled depending on the user's application system clock rate. For instance, if the CDP1802 is operated at a 6-MHz clock rate, the time shown is **reduced** by a factor of three.

I. Arithmetic Functions

	ADD	SUBTRACT	MULTIPLY	DIVIDE
Best:	0.152 ms	0.168 ms	3.112 ms	13.80 ms
Worst:	0.264 ms	0.32 ms	9.782 ms	16.15 ms

II. BCD ⇄ BINARY

	BINARY → BCD	BCD → BINARY
Best:	15.36 ms	5.84 ms
Worst:	73.48 ms	29.76 ms

Fig. 3(a)—Timing measurements for the RCA CDP1801 COSMAC Microprocessor (based on 2-MHz clock rate).

I. Arithmetic Functions

	ADD	SUBTRACT	MULTIPLY	DIVIDE
Best:	0.136 ms	0.128 ms	2.792 ms	4.496 ms
Worst:	0.224 ms	0.256 ms	4.223 ms	5.856 ms

II. BCD ⇄ BINARY

	BINARY → BCD	BCD → BINARY
Best:	4.37 ms	0.31 ms
Worst:	9.27 ms	2.67 ms

Fig. 3(b)—Timing measurements for the RCA CDP1802 COSMAC Microprocessor (based on 2-MHz clock rate).

Detailed Description of Subroutines

The following material provides a detailed description of each subroutine and how it is used. For ease of use this information is given in a standard format which includes the subroutine identification, its function, the calling procedure, the registers used, other subroutines used, the meaning of DF on return, the number magnitude, the time range for the subroutine, its length in bytes, and, where helpful, an example.

Index by Category

Index - Alphabetically

1. **Subroutine:** ADDOP
   ADD
2. **Function:**
   ADDOP and ADD are two entries of a subroutine that add the content of R(AC) and a two-byte operand in memory pointed to by R(MA). The sum is stored in R(AC).

   ADDOP fetches the address of operand from the calling program and stores it in R(MA).

   ADD assumes the address of operand is already in R(MA).
3. **Calling Procedure:**

```
              . . enter from ADDOP
SEP CALL
,A(ADDOP)
,A(OPR)     . . where OPR is
              . . the address of the
              . . operand.

              . . enter from ADD
SEP CALL
,A(ADD) . . R(MA) contains the address of
              . . operand.
```

4. **Registers Used:**
   R(LINK) for ADDOP only, R(MA), R(AC), R(SP).
5. **Other Subroutines Used:**
   ADDOP : should be followed in memory by ADD.
   ADD : none
6. **On Return:**
   DF = 0 means addition was successful.
   DF = 1 means the sum has exceeded the maximum or minimum range of representable numbers.
7. **Number:**

Representation:	Signed 2's complement
Width:	16 bits
Range:	$8000 \leqslant \text{number}_{16} \leqslant 7FFF$
	$-32768 \leqslant \text{number}_{10} \leqslant 32767$

8. **Time:**

CLOCK RATE:		2 MHz	6.4 MHz	
CDP1801	Best	0.152	–	ms
	Worst	0.264	–	ms
CDP1802	Best	0.136	0.042	ms
	Worst	0.192	0.060	ms

9. **Length:**

	CDP1801 version	CDP1802 version	
ADDOP	41	26	bytes
ADD	37	22	bytes

10. **Example:**

```
              . . assume AC = FFFE(= –2);
              . . assume M(OPR) = #0001
              . . enter from ADDOP
SEP CALL
,A(ADDOP)
,A(OPR)     . . address of operand
[code]      . . R(AC) is now FFFF (= –1)
              . . with DF = 0
```

1. **Subroutine:** ADDCON
2. **Function:**
   This subroutine adds the content of R(AC) to the two-byte constant passed from the calling program. The sum is placed in R(AC).
3. **Calling Procedure:**

```
SEP CALL
,A(ADDCON)
,X'nnnn'      . . where nnnn is the
              . . constant to be added to R(AC).
```

4. **Registers Used:**
   R(LINK), R(MA), R(AC), R(SP).
5. **Other Subroutine Used:**
   ADD
6. **On Return:**
   DF = 0 means addition was successful.
   DF = 1 means the sum exceeded the maximum or minimum range of representable numbers.
7. **Number:**

Representation:	Signed 2's complement
Width:	16 bits
Range:	$8000 \leqslant \text{number}_{16} \leqslant 7FFF$
	$-32768 \leqslant \text{number}_{10} \leqslant 32767$

8. **Time:**

CLOCK RATE:		2 MHz	6.4 MHz	
CDP1801	Best	0.208	–	ms
	Worst	0.288	–	ms
CDP1802	Best	0.192	0.060	ms
	Worst	0.216	0.067	ms

9. **Length:**

CDP1801 version	CDP1802 version	
45	30	bytes

1. **Subroutine:** ADDST
2. **Function:**
   ADDST adds the content of R(AC) to the top two bytes of stack. These two bytes should be stored in a sequence such that the more significant part is on the top and the less significant part is at the bottom.
3. **Calling Procedure:**

```
SEP CALL
,A(ADDST)   . . 2-byte constant is stored
              . . on top of stack.
```

4. **Registers Used:**
   R(LINK), R(MA), R(AC), R(SP).
5. **Other Subroutine Used:**
   ADD
6. **On Return:**
   DF = 0 means addition was successful.
   DF = 1 means the sum exceeded the maximum or minimum range of representable numbers.
7. **Number:**

Representation:	Signed 2's complement
Width:	16 bits
Range:	$8000 \leqslant \text{number}_{16} \leqslant 7FFF$ or
	$-32768 \leqslant \text{number}_{10} \leqslant 32767$

8. **Time:**

CLOCK RATE:		2 MHz	6.4 MHz	
CDP1801	Best	0.218	–	ms
	Worst	0.298	–	ms
CDP1802	Best	0.204	0.064	ms
	Worst	0.224	0.070	ms

9. **Length:**

CDP1801 version	CDP1802 version	
46	31	bytes

1. **Subroutine:** SDOP
   SD
2. **Function:**
   SDOP and SD are two different entries of a subroutine that subtract R(AC) from the contents of two bytes in memory that is pointed to by R(MA).

   SDOP fetches the address of minuend from the calling program and stores it in R(MA).

   SD assumes the address of minuend is already in R(MA).
3. **Calling Procedure:**

```
              . . to enter from SDOP
SEP CALL
,A(SDOP)
,A(MINU)    . . where MINU is the address
              . . of the minuend.

              . . to enter from SD
SEP CALL
,A(SD)      . . the address of minuend should be
              . . in R(MA).
```

4. **Registers Used:**
   R(LINK) for SDOP only, R(MA), R(AC), R(SP).
5. **Other Subroutines Used:**
   SDOP: should be followed in memory by SD.
   SD : none
6. **On Return:**
   DF = 0 means subtraction was successful.
   DF = 1 means subtraction was unsuccessful because the difference has exceeded the maximum or minimum range of representable numbers.
7. **Number:**

Representation:	Signed 2's complement
Width:	16 bits
Range:	$8000 \leqslant \text{number}_{16} \leqslant 7FFF$ or
	$-32768 \leqslant \text{number}_{10} \leqslant 32767.$

8. **Time:**

CLOCK RATE:		2 MHz	6.4 MHz	
CDP1801	Best	0.168	–	ms
	Worst	0.256	–	ms
CDP1802	Best	0.128	0.040	ms
	Worst	0.184	0.057	ms

9. **Length:**

	CDP1801 version	CDP1802 version	
SDOP	35	24	bytes
SD	31	20	bytes

10. **Example:**

```
[code]      . . assume R(AC) = X'000A'
              . . assume M(MINU) = X'0005'
SEP CALL
,A(SDOP)    . . enter from SDOP
,A(MINU)    . . address of minuend will be
              . . fetched by SDOP
[code]      . . R(AC) is now FFFB (= –0005)
              . . with DF = 0.
```

1. **Subroutine:** SDCON
2. **Function:**
   This subroutine subtracts AC from a 16-bit constant and stores the difference in AC.
3. **Calling Procedure:**

```
SEP CALL
,A(SDCON)
,X'nnnn'  . . where nnnn is the constant.
```

4. **Registers Used:**
   R(LINK), R(MA), R(AC), R(SP).
5. **Other Subroutine Used:**
   SD
6. **On Return:**
   DF = 0 means subtraction was successful.
   DF = 1 means subtraction was unsuccessful because the difference exceeded the maximum or minimum range of representable numbers.
7. **Number:**

Representation:	Signed 2's complement
Width:	16 bits
Range:	$8000 \leqslant \text{number}_{16} \leqslant 7FFF$ or $-32768 \leqslant \text{number}_{10} \leqslant 32767$

8. **Time:**

CLOCK RATE:		2 MHz	6.4 MHz	
CDP1801	Best	0.216	—	ms
	Worst	0.272	—	ms
CDP1802	Best	0.184	0.057	ms
	Worst	0.208	0.065	ms

9. **Length:**

CDP1801 version	CDP1802 version	
39	28	bytes

10. **Example:**

```
[code]       . . assume R(AC) = X'0002'
SEP CALL
,A(SDCON)
,X'0002'     . . constant is X'0005'
,[code]      . . R(AC) is now X'0003' with DF = 0
```

1. **Subroutine:** SMOP SM
2. **Function:**
   SMOP and SM are two different entries of a subroutine that subtract the contents of two bytes in memory, which is pointed to by R(MA), from R(AC). SMOP fetches the address of subtrahend from the calling program and stores it in R(MA). SM assumes the address of subtrahend is already in R(MA).
3. **Calling Procedure:**

```
             . . to enter from SMOP
SEP CALL
,A(SMOP)
,A(SUBT)     . . where SUBT is the
             . . address of the subtrahend.

             . . to enter from SM
SEP CALL
,A(SM)       . . R(MA) should contain
             . . the address of subtrahend.
```

4. **Registers Used:**
   R(LINK) for SMOP only, R(MA), R(AC), R(SP).
5. **Other Subroutines Used:**
   SMOP: should be followed in memory by SM.
   SM : none
6. **On Return:**
   DF = 0 means subtraction was successful.
   DF = 1 means subtraction was unsuccessful because the difference has exceeded the maximum or minimum range of representable numbers.
7. **Number:**

Representation:	Signed 2's complement
Width:	16 bits
Range:	$8000 \leqslant \text{number}_{16} \leqslant 7FFF$ or $-32768 \leqslant \text{number}_{10} \leqslant 32767$

8. **Time:**

CLOCK RATE:		2 MHz	6.4 MHz	
CDP1801	Best	0.216	—	ms
	Worst	0.320	—	ms
CDP1802	Best	0.192	0.060	ms
	Worst	0.256	0.080	ms

9. **Length:**

	CDP1801 version	CDP1802 version	
SMOP	49	34	bytes
SM	45	30	bytes

10. **Example:**

```
[code]       . . assume M(100; 101) = 0001 and
             . . R(MA) = 0100 and R(AC) = 8000
SEP CALL
,A(SM)       . . take SM entry
[code]       . . DF = 1
             . . difference exceeded the
             . . maximum range of negative
             . . number
```

1. **Subroutine:** MPYOP MPY
2. **Function:**
   MPYOP and MPY are two different entries of a subroutine that multiply R(AC) by the contents of two bytes in memory which are pointed to by R(MA). The product is placed in R(MQ) and R(AC) with the highest bit of R(MQ) as the sign bit and the lowest bit of R(AC) as the least significant bit.

   MPYOP fetches the address of multiplier from the calling program and stores it in R(MA).

   MPY assumes the address of multiplier is already in R(MA).
3. **Calling Procedure:**

```
             . . to enter from MPYOP
SEP CALL
,A(MPYOP)
,A(MPLR)     . . where MPLR is the address
             . . of the multiplier.

             . . to enter from MPY
SEP CALL
,A(MPY)      . . the address of multiplier
             . . should be in R(MA).
```

4. **Registers Used:**
   R(LINK) for MPYOP only, R(MA), R(MQ), R(AC), R(CR).
5. **Other Subroutines Used:**
   MPYOP: should be followed in memory by MPY.
   MPY : DSM, DVA (for CDP1801 version only).
6. **On Return:**
   DF = 0 means product $\leqslant$ 16 bits
   DF = 1 means product > 16 bits
7. **Number:**

Representation:	Signed 2's complement
Width:	Multiplicand and Multiplier: 16 bits
	Product: 32 bits
Range:	Multiplicand and Multiplier: $8000 \leqslant \text{number}_{16} \leqslant 7FFF$ or $-32768 \leqslant \text{number}_{10} \leqslant 32767$
	Product: $C0008000 \leqslant \text{number}_{16} \leqslant 40000000$ $-1073709056 \leqslant \text{number}_{10} \leqslant 1073741824$

8. **Time:**

CLOCK RATE		2 MHz	6.4 MHz	
CDP1801	Best	3.112	–	ms
	Worst	9.782	–	ms
CDP1802	Best	2.792	0.872	ms
	Worst	4.223	1.320	ms

9. **Length:**

	CDP1801 version*	CDP1802 version	
MPYOP	131	81	bytes
MPY	127	77	bytes

*Including secondary subroutines.

10. **Example:**

```
[code]       . . assume M(MPLR) = X'0001'
             . . assume R(AC) = FFFF(= -0001)
SEP CALL
,A(MPYOP). . enter from MPYOP
,A(MPLR)
[code]       . . [R(MQ); R(AC)] = FFFFFFFF(= -1)
             . . with DF = 0 since the magnitude
             . . of product is less than 15 bits
```

1. **Subroutine:** **DIVOP DIV0 DIV DIVQ**

2. **Function:**
DIVOP, DIV0, DIV, DIVQ are four different entries of a subroutine which divide the contents of [R(MQ); R(AC)] by the contents of two bytes in memory pointed to by R(MA).

Upon returning from the subroutine, R(AC) contains the quotient and R(MQ) contains the remainder.

DIVOP fetches the address of divisor from the calling program and stores it in R(MA).

DIV0 assumes the address of divisor is already in R(MA) and the magnitude of dividend does not exceed 15 bits. It extends the sign bit of R(AC) into R(MQ), and does a divide check (divisor = 0).

DIV also assumes the address of divisor is already in R(MA). It checks dividend against divisor to make sure that the magnitude of quotient will not exceed 15 bits. The divisible numbers are shown on the following page.

DIVQ assumes R(MA) contains the address of divisor. It divides [R(MQ); R(AC)] by the operand in memory which is pointed to by R(MA). When returning to the calling program, R(AC) contains the quotient with remainder stored in R(MQ). This entry does not check condition; hence, it is the user's responsibility to determine if the quotient is meaningful.

3. **Calling Procedure:**

```
                   . . enter from DIVOP
SEP CALL
,A(DIVOP)
,A(DIVR)           . . where DIVR is the
                   . . address of divisor.

                   . . enter from the other entry points
SEP CALL
,A(DIV0)           . . or DIV, or DIVQ
                   . . the address of divisor
                   . . should be in MA.
```

4. **Registers Used:**

DIVOP:	R(LINK), R(MA).
DIV0:	R(MA), R(AC), R(MQ).
DIV:	R(CR), R(MQ), R(AC), R(MA).
DIVQ:	R(CR), R(MQ), R(AC), R(MA), R(SP).

5. **Other Subroutines Used:**
DIVOP: The code should fall through DIV0 and DIVQ if zero divide check and sign extension option (which is performed by DIV0) is chosen.
Alternatively, the code could fall through DIV and DIVQ if the option of quotient overflow check is selected.

DIV0: The code should fall through DIV, DIVQ.

DIV: (1) The code should fall through DIVQ.
(2) DSHL, DABS (for CDP1801 only).

DIVQ: DSHL, DABS (for CDP1801 only).

6. **On Return:**
DF = 0 means division was successful.
DF = 1 means:
(1) If subroutines are in sequence of DIVOP, DIV0, DIVQ, or DIV0, DIVQ: Division was unsuccessful because [R(MQ); R(AC)] division by 0 was attempted.
(2) If subroutines are in sequence of DIVOP, DIV, DIVQ or DIV, DIVQ: Division was unsuccessful because the magnitude of dividend is too large and, consequently, R(AC) will overflow.

The contents of R(MQ) and R(AC) are not changed if DF is set to 1.

7. **Number:**
Representation: Signed 2's complement
Width: Dividend: 32 bits
Divisor, Quotient, Remainder: 16 bits
Range: Dividend (absolute):
C0000001 $\leqslant \text{number}_{16} \leqslant$ 400007FFF
$-1073741823 \leqslant \text{number}_{10} \leqslant 1073774591$
Divisor, Remainder, Quotient:
8000 $\leqslant \text{number}_{16} \leqslant$ 7FFF
$-32768 \leqslant \text{number}_{10} \leqslant 32767$

8. **Time:**

CLOCK RATE:		2 MHz	6.4 MHz	
CDP1801	Best	13.38	–	ms
	Worst	15.76	–	ms
CDP1802	Best	4.496	1.405	ms
	Worst	5.856	1.830	ms

9. **Length:**
The numbers indicated here are the lengths of each entry of the divide subroutine. Thus, the actual length of the subroutine should be the sum of the entries that are used.

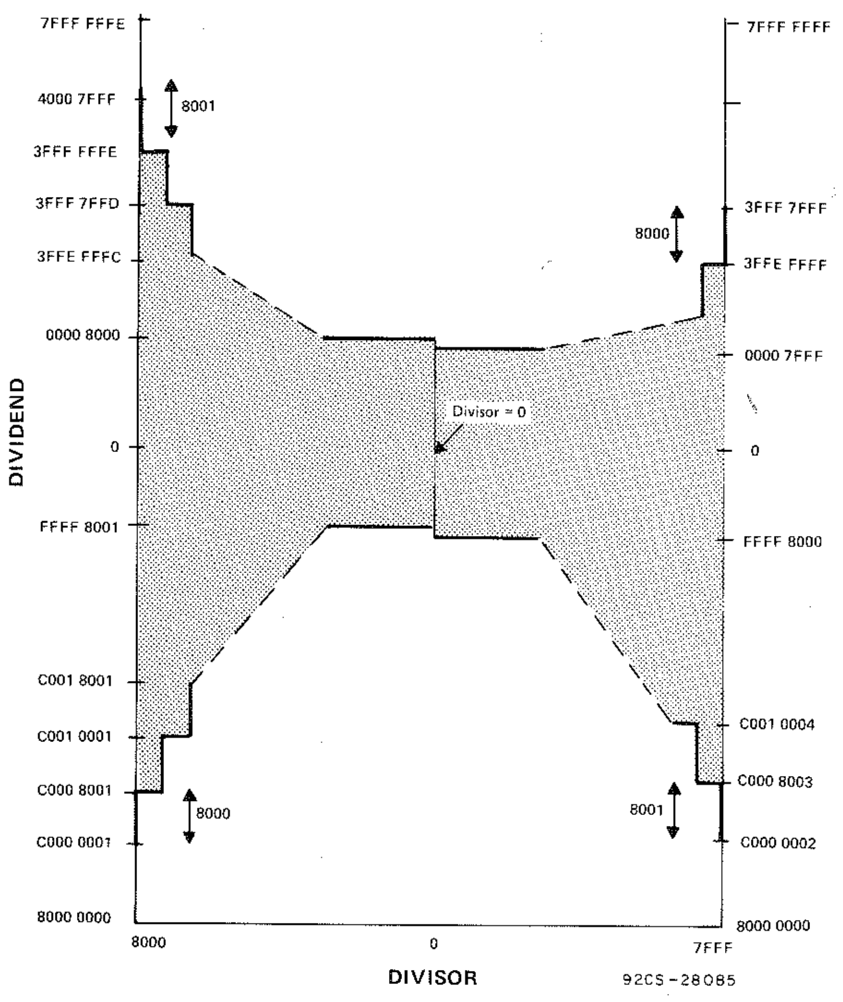
	CDP1801 version	CDP1802 version	
DIVOP	4	4	bytes
DIV0	20	18	bytes
DIV	170	168	bytes
DIVQ	173	114	bytes

10. **Example 1:**

```
[code]       . . assume M(DIVR) = X'0003'
             . . assume R(AC) = X'000A'
             . . enter DIVOP, DIV0, DIVQ
SEP CALL
,A(DIVOP)    . . followed by DIV0, DIVQ
,A(DIVR)     . . address of divisor
[code]       . . [R(MQ); R(AC)] is now 00010003
             . . where remainder = 0001
             . . and quotient = 0003
             . . with DF = 0
```

Example 2:

```
[code]       . . assume M(DVSR) = X'0002'
[code]       . . and [R(MQ); R(AC)] = 00500001
             . . enter DIVOP, DIV, DIVQ
SEP CALL
,A(DIVOP)    . . followed by DIV, DIVQ
,A(DVSR)     . . address of divisor
[code]       . . DF = 1 since [R(MQ); R(AC)]
             . . (= 00500001) divided by
             . . 0002 would result in
             . . overflow of AC (Quotient
             . . is too large to be able to
             . . store in 16 bits)
```



Shaded area indicates divisible numbers.

1. **Subroutine:** **CBD**
2. **Function:**
CBD converts the signed 2's complement number in R(AC) to binary-coded-decimal (BCD) numbers. The address for the storing is a two-byte hex number passed from the calling program. The number of digits (including sign) of this BCD number is a one-byte hexadecimal number passed also from the calling program to CBD. On return, the BCD result is stored in a sequence such that the sign of the BCD numbers is followed for the CDP1801 version by the least significant digit, followed by the second least significant digit, and for the CDP1802 version, by the most significant digit, followed by the second most significant digit, etc.
3. **Calling Procedure:**

```
SEP CALL
,A(CBD)
,A(BCD)     . . address for BCD result
,#nn        . . where nn is a one-byte
            . . constant in hex to indicate
            . . the maximum number of
            . . digits for result (including
            . . sign).
```

4. **Registers Used:**
R(LINK), R(AR), R(NR), R(AC), R(MA).
5. **Other Subroutines Used:**
SDCON, DZS, PUSHAC, DIVQ } For CDP1801 version only.
6. **On Return:**
DF = 0 means conversion was successful.
DF = 1 means the number of digits of BCD (and sign) is larger than the length argument passed from the calling program. (Overflowed!)
7. **Number:**

Representation:	Signed 2's complement R(AC).
Width:	R(AC): 16 bits
	BCD: $2 \leqslant \text{length} \leqslant 6$
Range:	$8000 \leqslant AC \leqslant 7FFF$
	$-32768 \leqslant BCD \leqslant 32767$
Sign of BCD:	#0B = +
	#0D = −

8. **Time:**

CLOCK RATE:		2 MHz	6.4 MHz	
CDP1801	Best	15.36	–	ms
	Worst	73.48	–	ms
CDP1802	Best	4.37	1.366	ms
	Worst	9.27	2.897	ms

9. **Length:**

CDP1801 version*	CDP1802 version	
339	83	bytes

*Including secondary subroutines.

10. **Example:**

```
            . . assume AC = FFE0 -32₁₀
            . . assume BCD is to be stored in memory
            . . starting at address A(RSLT).
SEP CALL
,A(CBD)
,A(RSLT)    . . address of BCD to be stored
,#05        . . only three digit locations will be used.
                The last two locations will be
                cleared to zeros.
            . . the BCD number will be stored in
                memory pointed to
            . . by R(AR).
```

For the CDP1801 version, the above conversion will produce the following:

R(AR) =RSLT	M(RSLT)	M(RSLT+1)	M(RSLT+2)	M(RSLT+3)	M(RSLT+4)	
→	0D	02	03	00	00	$= -32_{10}$
	SIGN	10^0	10^1	10^2	10^3	

5 bytes in length

For the CDP1802 version, the above conversion will produce the following:

R(AR) =RSLT	M(RSLT)	M(RSLT+1)	M(RSLT+2)	M(RSLT+3)	M(RSLT+4)	
→	0D	00	00	03	02	$= -32_{10}$
	SIGN	10^3	10^2	10^1	10^0	

5 bytes in length

1. **Subroutine:** **CDB**
2. **Function:**
CDB converts the binary-coded-decimal (BCD) number stored in memory, whose address is passed from the calling program, into a signed 2's complement number and stores it in R(AC). The length of the BCD number, i.e., the number of digits and the sign, is a one-byte argument that is also passed from the calling program. The BCD number should be stored in memory in a sequence such that the first byte is the sign and, for the CDP1801, the second byte is the least significant digit, the third byte is the second least significant digit, etc.; for the CDP1802, the second byte is the most significant digit, the third byte is the second most significant digit, etc.

The sign of this BCD number is represented by #0B(+) and #0D(−).
3. **Calling Procedure:**

```
SEP CALL
,A(CDB)
,A(BCD number)  . . address of the BCD number
,#nn            . . length of the BCD, includes
                . . sign.
```

4. **Registers Used:**
R(LINK), R(AR), R(NR), R(AC).
5. **Other Subroutines Used:**
PUSHAC, LODCON, STORE, ADDST, SDCON, MPY } For CDP1801 version only.
6. **On Return:**
DF = 0 means conversion was successful.
DF = 1 means conversion failed because the BCD number is either too large or too small to be representable as a signed 16-bit 2's complement number.
7. **Number:**

Representation:	Signed 2's complement R(AC).
Width:	R(AC): 16 bits
	BCD: $2 \leqslant \text{length} \leqslant 6$
Range:	$8000 \leqslant R(AC) \leqslant 7FFF$
	$-32768 \leqslant BCD \leqslant 32767$

8. **Time:**

CLOCK RATE:		2 MHz	6.4 MHz	
CDP1801	Best	5.84	–	ms
	Worst	29.24	–	ms
CDP1802	Best	0.31	0.097	ms
	Worst	2.67	0.834	ms

9. **Length:**

CDP1801 version*	CDP1802 version	
342	107	bytes

*Including secondary subroutines.

10. **Example:**
. . assume the BCD number, +1536, is to be
. . converted to signed 2's complement number
. . and the BCD is stored in the memory as

M(BCD)	M(BCD+1)	M(BCD+2)	M(BCD+3)	M(BCD+4)	
0B	X6	X3	X5	X1	$= 1536_{10}$
SIGN	10^0	10^1	10^2	10^3	

for the CDP1801 version, or

M(BCD)	M(BCD+1)	M(BCD+2)	M(BCD+3)	M(BCD+4)	
0B	X1	X5	X3	X6	$= 1536_{10}$
SIGN	10^3	10^2	10^1	10^0	

for the CDP1802 version.

where X is ignored by CDB, i.e., the BCD digits can be stored as ASCII numbers.

```
SEP CALL
,A(CDB)
,A(BCD)  . . address of BCD
,#05     . . 4 digits + sign = length of BCD
         . . R(AC) now contains 0600₁₆(= 1536₁₀)
```

1. **Subroutine:** PUSHAC
2. **Function:**
   This subroutine pushes the content of R(AC) onto stack with R(AC).1 on the top and R(AC).0 at the bottom. Upon returning, R(MA) is left pointing to R(AC).1 (one byte below the top of the stack, i.e., R(MA) = R(SP)+1).
3. **Calling Procedure:**
   ```
   SEP CALL
   ,A(PUSHAC)
   ```
4. **Registers Used:**
   R(MA), R(AC), R(SP).
5. **Other Subroutine Used:**
   None
6. **On Return:**
   DF is not changed by PUSHAC.
7. **Length:**

CDP1801 version	CDP1802 version	
21	19	bytes

1. **Subroutine:** POPAC POP
2. **Function:**
   POPAC pops the top two bytes off the stack and stores them in R(AC).1 and R(AC).0. On return, R(MA) is left pointing to the byte below the top of the stack, i.e., R(MA) = R(SP)+1.

   POP is another entry of the subroutine which pops two bytes off the stack without storing them in R(AC).
3. **Calling Procedure:**
   ```
   SEP CALL
   ,A(POPAC)   . . or POP
   ```
4. **Registers Used:**
   R(MA), R(AC) for POPAC only, R(SP).
5. **Other Subroutine Used:**
   None
6. **On Return:**
   DF = 0 means the top two bytes were not stored in R(AC).
   DF = 1 means R(AC) contains the top two bytes of stack.
7. **Length:**

	CDP1801 version	CDP1802 version	
POPAC	29	26	bytes
POP	25	23	bytes

1. **Subroutine:** PUSHCQ
2. **Function:**
   PUSHCQ pushes the contents of R(CR), R(MA), R(MQ) onto stack. R(MQ).1, R(MQ).0, R(MA).1, R(MA).0, R(CR).1, R(CR).0 are stored from top to bottom accordingly. R(MA) is changed by PUSHCQ.
3. **Calling Procedure:**
   ```
   SEP CALL
   ,A(PUSHCQ)
   ```
4. **Registers Used:**
   R(CR), R(MA), R(MQ), R(SP),(R(MA) is changed by PUSHCQ).
5. **Other Subroutine Used:**
   None
6. **On Return:**
   DF is not changed by PUSHCQ.
7. **Length:**

CDP1801 version	CDP1802 version	
33	28	bytes

1. **Subroutine:** POPCQ
2. **Function:**
   POPCQ pops each of the top six bytes of the stack into R(MQ).1, R(MQ).0, R(MA).1, R(MA).0, R(CR).1, R(CR).0 in sequence.
3. **Calling Procedure:**
   ```
   SEP CALL
   ,A(POPCQ)
   ```
4. **Registers Used:**
   R(MQ), R(MA), R(CR), R(SP).
5. **Other Subroutine Used:**
   None
6. **On Return:**
   DF is not changed by POPCQ.
7. **Length:**

CDP1801 version	CDP1802 version	
32	30	bytes

1. **Subroutine:** LOADOP LOAD
2. **Function:**
   LOADOP and LOAD are entries to a subroutine that load the contents of memory (two bytes), whose high order byte is pointed to by R(MA), into R(AC). R(MA) is incremented twice by LOAD.

   LOADOP fetches the address of the operand and stores it into R(MA) and then does a LOAD.

   LOAD assumes R(MA) is pointing to operand.
3. **Calling Procedure:**
   ```
                . . enter from LOADOP
   SEP CALL
   ,A(LOADOP)
   ,A(OPR)      . . Load M(OPR) and M(OPR+1)
                    into R(AC)

                . . enter from LOAD
   SEP CALL
   ,A(LOAD)     . . Load M[R(MA)] and M[R(MA+1)]
                    into R(AC)
   ```
4. **Registers Used:**
   R(LINK) for LOADOP only, R(MA), R(AC).
5. **Other Subroutines Used:**
   LOADOP: should be followed by LOAD.
   LOAD: none.
6. **On Return:**
   DF is not changed by LOADOP or LOAD.
7. **Length:**

	CDP1801 version	CDP1802 version	
LOADOP:	9	9	bytes
LOAD:	5	5	bytes

8. **Example:**
   ```
                . . assume M(OPR) contains X'AB'
                . . assume M(OPR+1) contains X'CD'
   SEP CALL
   ,A(LOADOP)
   ,A(OPR)      . . address of ABCD
                . . R(AC) now contains ABCD
                . . R(MA) contains A(OPR+2)
   ```

1. **Subroutine:** LODCON
2. **Function:**
   LODCON loads the two-byte constant following the subroutine call into register AC.
3. **Calling Procedure:**
   ```
   SEP CALL
   ,A(LODCON)
   ,X'nnnn'     . . the constant
   [code]
   ```
4. **Registers Used:**
   R(LINK), R(AC).
5. **Other Subroutine Used:**
   None
6. **On Return:**
   DF is not changed by LODCON.
7. **Length:**

CDP1801 version	CDP1802 version	
5	5	bytes

1. **Subroutine:** STOROP STORE
2. **Function:**
   STOROP and STORE are entries of a subroutine which stores the contents of R(AC) into the memory locations pointed to by R(MA). On return, R(MA) is pointing to the next 16-bit word, i.e., R(MA) is incremented twice.

   STOROP fetches the address of the operand and stores it into R(MA). It then does a STORE.

   STORE assumes R(MA) contains the address of the operand.
3. **Calling Procedure:**

```
                .. to enter from STOROP
SEP CALL
,A(STOROP)
,A(OPR)         .. address of operand
[code]          .. R(MA) is now A(OPR)+#02

                .. to enter from STORE
SEP CALL
,A(STORE)
[code]          .. R(MA) is now A(OPR)+#02
```

4. **Registers Used:**
   R(MA), R(AC), R(LINK) for STOROP only.
5. **Other Subroutines Used:**
   STOROP: should be followed in memory by STORE.
   STORE: none.
6. **On Return:**
   DF is not changed by STORE or STOROP.
7. **Length:**

	CDP1801 version	CDP1802 version	
STOROP:	11	11	bytes
STORE:	7	7	bytes

1. **Subroutine:** COMPOP COMP
2. **Function:**
   COMPOP and COMP are entries to a subroutine that compares R(AC) and the two-byte contents of memory pointed to by R(MA). DF is set to 1 if R(AC) is greater than or equal to the contents of memory.

   COMPOP fetches the operand address and stores it into R(MA). It then does a COMP.

   COMP assumes R(MA) is pointing to the operand.
3. **Calling Procedure:**

```
                .. enter from COMPOP
SEP CALL
,A(COMPOP)
,A(OPR)         .. address of operand

                .. enter from COMP
SEP CALL
,A(COMP)        .. do comparison
```

4. **Registers Used:**
   R(AC), R(MA), R(LINK) for COMPOP only.
5. **Other Subroutines Used:**
   COMPOP: should be followed in memory by COMP.
   COMP: none.
6. **On Return:**
   DF = 0 means R(AC) < M(MA).
   DF = 1 means R(AC $\geqslant$ M(MA).
7. **Number:**

Representation:	Signed 2's complement
Width:	16 bits
Range:	$8000 \leqslant \text{number}_{16} \leqslant 7FFF$
	$-32768 < \text{number}_{10} \leqslant 32767$

8. **Length:**

	CDP1801 version	CDP1802 version	
COMPOP:	22	22	bytes
COMP:	18	18	bytes

9. **Example:**

```
                .. assume R(AC) = 0001;
                .. M(OPR) contains X'FFFF'
SEP CALL
,A(COMPOP)
,A(OPR)         .. address for comparison
                .. DF is set to 1 since
                .. 0001 > FFFF(= -1)
```

1. **Subroutine:** TEST
2. **Function:**
   TEST tests if R(AC) is equal to zero. If so, TEST returns to the 3rd byte immediately following the SEP call of TEST in the calling program. If R(AC) is not zero, TEST returns to the 5th byte following the SEP call of TEST in the calling program.

   DF is set to 1 if AC is negative.
3. **Calling Procedure:**

```
SEP CALL
,A(TEST)
BR ZEROL          .. TEST returns here
                     if AC = 0.
[non-zero exit]   .. TEST returns here
                     if AC ≠ 0.
```

4. **Registers Used:**
   R(LINK), R(AC).
5. **Other Subroutine Used:**
   None
6. **On Return:**
   DF = 0 means number in R(AC) is positive.
   DF = 1 means number in R(AC) is negative.
7. **Length:**

CDP1801 version	CDP1802 version	
11	10	bytes

1. **Subroutine:** SWAPAQ
2. **Function:**
   This subroutine exchanges the contents of R(AC) and R(MQ). R(CR).0 is used as a scratch register.
3. **Calling Procedure:**

```
SEP CALL
,A(SWAPAQ)
```

4. **Registers Used:**
   R(CR), R(MA), R(AC).
5. **Other Subroutine Used:**
   None
6. **On Return:**
   DF is not changed by SWAPAQ.
7. **Length:**

CDP1801 version	CDP1802 version	
13	13	bytes

Appendix A - Standard Subroutine Call and Return Technique

I. CDP1801 Version:

A. Subroutine CALL:

```
CALLR:  SEP R3      . . to subroutine . . .
CALLS:  GHI R6      . . save R6 on stack.
        STR R2
        DEC R2
        GLO R6
        STR R2
        DEC R2
        GHI R3      . . save R3 in R6.
        PHI R6
        GLO R3
        PLO R6
        LDA R6      . . set subroutine address
        PHI R3      . . in R3.
        LDA R6
        PLO R3
        BR CALLR    . . jump to subroutine.
```

B. Subroutine RETURN:

```
RETR:   SEP R3      . . return to main
RET:    GHI R6      . . restore return address
        PHI R3      . . into R3.
        GLO R6
        PLO R3
        INC R2      . . restore old R6 saved on stack
        LDA R2      . . into R6
        PLO R6
        LDA R2
        PHI R6
        DEC R2
        BR RETR     . . return.
```

II. CDP1802 Version:

A. Subroutine CALL:

```
CALLR:  SEP R3      . . to subroutine . . .
CALLS:  SEX R2      . . point to stack
        GHI R6      . . save R6 on stack.
        STXD
        GLO R6
        STXD
        GHI R3      . . save R3 in R6.
        PHI R6
        GLO R3
        PLO R6
        LDA R6      . . set subroutine address
        PHI R3      . . in R3.
        LDA R6
        PLO R3
        BR CALLR    . . jump to subroutine.
```

B. Subroutine RETURN:

```
RETR:   SEP R3      . . return to main . . .
RET:    GHI R6      . . restore return address
        PHI R3      . . into R3.
        GLO R6
        PLO R3
        SEX R2      . . restore old R6 saved
        INC R2      . . on stack into R6.
        LDXA
        PLO R6
        LDX
        PHI R6
        BR RETR     . . return.
```

Appendix B(I) - Arithmetic Subroutine Listing for CDP1801

```
!M
0000 ;        .... COPYRIGHT 1975 RCA CORPORATION ....
0000 ;
0000 ;         ORG #0400
0400 ;         .. COSMAC ARITHMETIC SUBROUTINE PACKAGE
0400 ;         ..
0400 ;         .. EXTRACTED FOR BINARY ARITHMETIC
0400 ;         .. AND CONVERSIONS FOR DECIMAL
0400 ;         .. OCCUPIES 1K BYTES
0400 ;         ..
0400 ;         ..
0400 ;         .. DESIGNED FOR STANDARD CSDP SUBROUTINE CALL&RETURN
0400 ;         ..
0400 ;         .. FOR STANDARD LINKAGE,
0400 ;         SP= #02 ...IT SHOULD BE THE STACK POINTER.
0400 ;         PC= #03 ...IT IS THE PROGRAM COUNTER
0400 ;         .. USED BY THESE SUBROUTINES.
0400 ;         CALL= #04 ...IT SHOULD POINT TO THE ROUTINE
0400 ;         .. WHICH EFFECTS SUBROUTINE CALLS.
0400 ;         RETN= #05 ...IT SHOULD POINT TO THE ROUTINE
0400 ;         .. WHICH EFFECTS SUBROUTINE RETURN.
0400 ;         LINK= #06 ...IT SHOULD POINT TO CALL PARAMETER
0400 ;         .. .. (USUALLY OPERAND ADDRESSES AND/OR CONSTANT)
0400 ;         ..
0400 ;         ..
0400 ;         .. THE FOLLOWING REGISTERS MUST BE ASSIGNED
0400 ;      AR= #0A  ..(USED FOR RESULT ADDRESS)
0400 ;
0400 ;      NR= #0B  ..(USED FOR RESULT DIGIT COUNT)
0400 ;         .. 16-BIT BINARY ARITHMETIC ROUTINES.
0400 ;         .. THE FOLLOWING REGISTERS MUST BE ASSIGNED
0400 ;      AC= #0F .. 16-BIT ACCUMULATOR=RF.
0400 ;      MQ= #0E .. 16-BIT ACCUMULATOR=RE EXTENSION.
0400 ;      MA= #0D .. (TEMPOR=RDARY) OPERAND MEMORY ADDRESS.
0400 ;      CR= #0C .. (TEMPORARY) SCRATCHPAD AND COUNTER.
0400 ;         ..
0400 ;         ORG *+#08  ..FOR PAGE BOUNDARY
0408 ;         ..
0408 ;         .. SWAP AC WITH MQ REGISTERS
0408 9E;    SWAPAQ: GHI MQ .. SAVE MQ.1
0409 AC;        PLO CR .. IN CR.0 (COULD HAVE PUSHED ON STAC
040A 9F;        GHI AC .. NOW AC.1 TO MQ.1
040B BE;        PHI MQ
040C 8E;        GLO MQ .. SAVE MQ.0
040D BF;        PHI AC .. IN AC.1
040E 8F;        GLO AC .. THEN AC.0 TO MQ.0
040F AE;        PLO MQ
0410 9F;        GHI AC .. NOW SAVED MQ.0 TO AC.0
0411 AF;        PLO AC
0412 8C;        GLO CR .. FINALLY SAVED MQ.1
0413 BF;        PHI AC .. TO AC.1
0414 D5;        SEP RETN
0415 ;          .. 16-BIT SUBTRACT AC FROM CONSTANT
0415 ;          .. ****AC= CONSTANT-AC
0415 ;          .. ************ (TO CALL, WRITE) ***********
0415 ;          .. ****CALL SDCON ; ,CONSTANT
0415 86;    SDCON: GLO LINK .. (ESSENTIALLY SAME AS ADCON)
0416 AD;        PLO MA
```

```
0417 96;                  GHI LINK
0418 BD;                  PHI MA
0419 16;                  INC LINK
041A 16;                  INC LINK
041B 3021;                BR SD
041D ;                    ..
041D ;                    .. 16-BIT SUBTRACT AC FROM OPERAND
041D ;                    .. ****AC=OPRN-AC
041D ;                    .. ************ (TO CALL, WRITE) ***********
041D ;                    .. ****CALL SDOP ; ,A(OPRN)
041D ;                  ..
041D 46;            SDOP: LDA LINK .. FETCH OPERAND ADDRESS
041E BD;                  PHI MA .. TO MA REGISTER
041F 46;                  LDA LINK
0420 AD;                  PLO MA .. FALL INTO SD
0421 ;                    .. 16-BIT SUBTRACT AC FROM OPERAND
0421 ;                    .. CALL HERE IF OPERAND
0421 ;                    .. ADDRESS IN REGISTER MA
0421 ;                    .. ****AC=M(R(MA))-AC
0421 ;                    .. ************ (TO CALL, WRITE) ***********
0421 ;                    .. ****CALL SD
0421 ;                    ..
0421 ED;            SD:     SEX MA ..  SET X PTR TO MA
0422 9F;                  GHI AC  ..CHECK SIGN BIT OF AC
0423 F3;                  XOR   ..AND OPERAND @MA
0424 FA80;                ANI #80  ..RESULT A 1 IF DIFF
0426 52;                  STR SP   ..AND STORE ON STACK
0427 1D;                  INC MA  ..POINT TO LOW 8
0428 8F;                  GLO AC .. FETCH AC LOW 8
0429 F5;                  SD .. SUBTRACT FROM LOW 8 IN MEMORY
042A AF;                  PLO AC .. PUT IT BACK
042B 2D;                  DEC MA .. NOW HIGH 8
042C 9F;                  GHI AC
042D 3335;                BDF SDNB .. (NO BORROW)
042F FC01;                ADI #01 .. PROPAGATE LOW 8 BORROW
0431 3B35;                BNF SDNB
0433 F4;                  ADD .. SECOND BORROW; FORCE BORROW OUT
0434 38;                  ,#38 .. AND SKIP OVER REGULAR SUBTRACT
0435 F5;            SDNB: SD .. SUBTRACT HIGH 8
0436 BF;                  PHI AC .. PUT IT BACK
0437 42;                  LDA SP   ..LOAD STORED COMPARING BIT OF OPERANDS
0438 22;                  DEC SP  ..RESET STACK POINTER
0439 323D;                BZ SDFF  ..IF OPERAND'S SIGNS ARE SAME
043B 9F;                  GHI AC  ..NO OVERFLOW POSSIBLE
043C F3;                  XOR   ..OTHER WISE CHECK RESULT
043D FC80;          SDFF:     ADI #80  ..SET DF=0  IF OK
043F D5;                    SEP RETN   ..RETURN
0440 ;                    ..
0440 ;                    .. 16-BIT SUBTRACT FROM AC (ADDRESS IN CALL)
0440 ;                    .. ****AC=AC-OPRN
0440 ;                    .. ************ (TO CALL, WRITE) ***********
0440 ;                    .. ****CALL SMOP ; ,A(OPRN)
0440 ;                    ..
0440 46;            SMOP: LDA LINK
0441 BD;                  PHI MA
0442 46;                  LDA LINK
0443 AD;                  PLO MA
0444 ;                    .. 16-BIT SUBTRACT FROM AC (ADDRESS IN MA)
```

```
0444 ;                    .. CALL HERE IF OPERAND
0444 ;                    .. ADDRESS IN REGISTER MA
0444 ;                    .. ****AC=AC-M(R(MA))
0444 ;                    .. ************ (TO CALL, WRITE) ***********
0444 ;                    .. ****CALL SM
0444 ;                    ..
0444 ED;            SM:    SEX MA  ..SET X PTR
0445 9F;                  GHI AC  ..GET SIGN OF AC AND
0446 F6;                  SHR   ..AND STORE IN 7TH BIT OF CR
0447 52;                  STR SP  ..BUT PUT IN (SP) FIRST
0448 9F;                  GHI AC  ..AND SEE IF OPERANDS SIGNS ARE THE SAME
0449 F3;                  XOR
044A FA80;                ANI #80  ..TAKE OUT COMPARING SIGN BIT
044C E2;                  SEX SP  ..NOW STORE THAT BIT IN 8TH OF CR
044D F4;                  ADD   ..BY ADDING TO IT
044E 52;                  STR SP  ..AND STORE THESE TWO BITS ON STACK
044F 1D;                  INC MA  ..POINT TO LOW 8
0450 ED;                  SEX MA  ..REMEMBER TO SET X TO OPERANDS
0451 8F;                  GLO AC .. FETCH AC LOW 8
0452 F7;                  SM .. SUBTRACT MEMORY FROM IT
0453 AF;                  PLO AC .. PUT IT BACK.
0454 2D;                  DEC MA .. NOW HIGH 8
0455 9F;                  GHI AC
0456 335F;                BDF SMNB
0458 FF01;                SMI #01 .. PROPAGATE BORROW OF LOW 8
045A 335F;                BDF SMNB
045C F3;                  XOR .. SECOND BORROW; FORCE BORROW OUT,
045D 38;                  ,#38 .. WHILE SUBTRACTING HIGH 8.
045E F7;             SMNB: SM .. HIGH 8 SUBTRACT, NO BORROW ACROSS.
045F BF;                  PHI AC .. PUT HIGH 8 BACK
0460 E2;                  SEX SP  ..NOW CHECK IF  UNDERFLOWED
0461 F0;                  LDX   ..LOAD THE STORED TWO BITS
0462 FA80;               ANI #80  ..AND TAKE OUT THE COMPARING SIGN BIT
0464 326B;                BZ SM1 ..THE SAME, UNDERFLOW NOT POSSIBLE
0466 9F;                  GHI AC   ..OTHERWISE HAVE TO COMPARE SIGN OF RE
.SULT
0467 F6;                  SHR  ..WITH SIGN OF AC
0468 F3;                  XOR  ..SIGN BIT OF AC WAS STORED ON STACK
0469 FA40;                ANI #40   ..TAKE OUT THAT 7TH BIT
046B F6;            SM1:      SHR  ..SET DF=0 IF OK
046C 3270;                BZ SMRT  ..THE SAME, IT'S OK
046E FF00;                SMI #00   ..OTHERWISE SET DF=1
0470 D5;            SMRT:      SEP RETN  ..RETURN WITH DF=NO BORROW
0471 ;                    ..
0471 ;                    ..
0471 ;                    ..16X16 BIT SIGNED MULTIPLY(2'S COMPLEMENT)
0471 ;                    ..****AC=AC*OPRN
0471 ;                    ..************(TO CALL, WRITE)***********
0471 ;                    ..*****CALL MPYOP ; ,A(OPRN)
0471 ;                    ..
0471 46;            MPYOP:  LDA LINK  ..FETCH MULTIPLICAND ADDRS
0472 BD;                  PHI MA  ..INTO REGISTER A
0473 46;                  LDA LINK
0474 AD;                  PLO MA  ..FALL INTO MPY
0475 ;                    .. 16X16 BIT SIGNED MULTIPLY (2'S COMPLEMENT
0475 ;                    .. CALL HERE IF OPERAND ADDRESS
0475 ;                    .. IN REGISTER MA
```

```
0475 ;             .. ****AC=AC*M(R(MA))
0475 ;             .. ************ (TO CALL, WRITE) **********
0475 ;             .. ****CALL MPY
0475 ;             ..
0475 ED;      MPY: SEX MA .. SET X NOW
0476 9F;           GHI AC  ..CHECK IS THE SIGN OF MULTICAND
0477 F3;           XOR  ..THE SAME AS THE SIGN OF MULTIPR
0478 FA80;         ANI #80  ..AND STORE THAT BIT
047A BC;           PHI CR  ..INTO CR.1
047B F810;         LDI #10 .. SET COUNTER TO 16
047D AC;           PLO CR
047E F800;         LDI #00 .. INITIALIZE MQ TO 0
0480 BF;           PHI MQ ... TO HOLD PRODUCT.
0481 AF;           PLO MQ
0482 2C;      MPL:     DEC CR  ..IF NOT, DECREMENT IT.
0483 9F;           GHI AC .. SHIFT AC (=MULTIPLIER) RIGHT
0484 F6;           SHR
0485 BF;           PHI AC
0486 8F;           GLO AC
0487 3B8D;         BNF *+#06
0489 F6;           SHR .. SHIFT 1 ACROSS BYTES ..
048A F980;         ORI #80
048C 38;           ,#38
048D F6;           SHR .. SHIFT 0 ACROSS BYTES ..
048E AF;           PLO AC
048F 9F;           GHI MQ
0490 3B9B;         BNF MPB  ..IF NO BIT OUT, DON'T ADD.
0492 1D;           INC MA .. POINT TO LOW 8 OF MULTIPLICAND
0493 8C;           GLO CR .. IF NOT LAST ITERATION,
0494 3A9F;         BNZ MPA .. GO ADD.
0496 D4;           SEP CALL  ..CALL SUBTRACT MQ$(MA) ROUTINE
0497 04D8;         ,A(DSM)
0499 ED;           SEX MA  ..FIX X PTR
049A 9C;           GHI CR  ..NOW ARE THE SIGNS OF OPERANDS THE SAME
049B FC80;    MPB:   ADI #80  ..TEST FOR SIGN BIT
049D 30A8;         BR MPS+#05  ..IF NEGATIVE, SIGN EXTEND
049F D4;      MPA:   SEP CALL  ..CALL ADD MQ & (MA) SUB
04A0 04E8;         ,A(DVA)
04A2 ED;           SEX MA
04A3 33A8;    MPS:   BDF *+#05
04A5 F0;           LDX
04A6 FC80;         ADI #80..IF OPERAND IS NEGATIVE,THEN
04A8 9F;           GHI MQ  ..PUT MQ.1 INTO D
04A9 3BAF;         BNF *+#06  ..EXTEND A 1 FOR -SIGN BIT
04AB F6;           SHR .. SHIFT IN 1
04AC F980;         ORI #80
04AE 38;           ,#38
04AF F6;           SHR .. SHIFT IN 0
04B0 BF;           PHI MQ
04B1 8F;           GLO MQ .. CONTINUE TO LOW 8 OF MQ
04B2 3BB8;         BNF *+#06
04B4 F6;           SHR
04B5 F980;         ORI #80
04B7 38;           ,#38
04B8 F6;           SHR
04B9 AF;           PLO MQ
04BA 3BC0;         BNF MT .. IF NO CARRY OUT, ITERATE.
04BC 9F;           GHI AC .. ADD CARRY OUT INTO AC MSB.
04BD F980;         ORI #80
```

```
04BF BF;           PHI AC
04C0 8C;      MT:    GLO CR  ..CHECK COUNTER
04C1 3A82;         BNZ MPL   ..IF COUNTER IS NOT 0,GO BACK FOR MORE
04C3 9F;       MPX: GHI AC .. FINISHED;
04C4 FC80;         ADI #80 .. CHECK FOR PRODUCT > 15 BITS.
04C6 8F;           GLO MQ
04C7 3BCB;         BNF *+#04  ..THAT'S HIGH 17 BITS
04C9 FBFF;         XRI #FF .. ALL 00 OR FF.
04CB 3AD2;         BNZ *+#07 .. NAW.
04CD 9F;           GHI MQ
04CE 3BD2;         BNF *+#04
04D0 FBFF;         XRI #FF
04D2 FCFF;         ADI #FF .. SET DF IF PRODUCT > 15 BITS
04D4 D5;           SEP RETN .. RETURN.
04D5 ;           ...SUBROUTINE TO BE INCLUDED IN DIV,DIVQ,MPY ....
04D5 1D;      DABS:      INC MA  ..MQ.0-(MA)
04D6 3BE8;         BNF DVA  ..IF NO FLAG, MQ.+(MA)
04D8 8F;      DSM:     GLO MQ  ..LOAD MQ.0 TO D
04D9 F7;           SM  ..MQ.0-(MA.0)
04DA AF;           PLO MQ  ..AND STORE BACK INTO MQ.0
04DB 2D;           DEC MA  ..DO THE SAME FOR MQ.1
04DC 9F;           GHI MQ  ..LOAD MQ .1 INTO D
04DD 33E5;         BDF *+#08  ..IF NO BORROW ,SKIP
04DF FF01;         SMI #01  ..MINUS 1 FOR BORROW OUT
04E1 33E5;         BDF *+#04  ..IF MQ.1 NO BORROW OUT, SKIP
04E3 F3;           XOR  ..OTHERWISE ONLY DO XOR
04E4 38;           ,#38  ..AND SKIP OVER NORMAL SUBTRACTION
04E5 F7;           SM  ..MQ.1-(MA.1)
04E6 30F6;          BR DVB  ..SKIP OVER ADD STEPS
04E8 8F;      DVA:    GLO MQ  ..MQ+(MA)
04E9 F4;           ADD  ..MQ.0+(MA.0)
04EA AF;           PLO MQ  ..AND PUT BACK INTO MQ.0
04EB 2D;           DEC MA  ..DO THE SAME FOR MQ.1 &MA.1
04EC 9F;           GHI MQ  ..LOAD MQ.1 INTO D
04ED 3BF5;         BNF *+#08  ..IF NO CARRY OUT,SKIP CARRY OVER
04EF FC01;         ADI #01  ..CARRY OVER STEP
04F1 3BF5;         BNF *+#04  ..SKIP CARRY IF NO CARRY RESULT
04F3 F5;           SD  ..OTHERWISE IT'S 0 HERE,PUT IT BACK
04F4 38;          ,#38   ..AND SKIP NORMAL ADDITION
04F5 F4;           ADD   ..ADD MQ.1 AND (MA.1)
04F6 BF;      DVB:    PHI MQ  ..AND STORE INTO MQ.1
04F7 D5;           SEP RETN ..
04F8 ;            .. SUBROUTINE TO BE INCLUDED IN DIV,DIVQ ....
04F8 E2;      DSHL:     SEX SP  ..SET X TO STACK
04F9 8F;           GLO MQ  ..ADD MQ.0 TO ITSELF
04FA 52;           STR SP  ..CAUSE MQ.0 SHIFT TO LEFT
04FB F4;           ADD  ..
04FC AF;           PLO MQ   ..AND STORE BACK
04FD 9F;           GHI MQ  ..DO THE SAME FOR MQ.1
04FE 52;           STR SP
04FF 3B03;         BNF *+#04  ..IF NO CARRY OUT,SKIP CARRY OVER
0501 FC01;         ADI #01  ..ADD 1 FOR CARRY OVER
0503 F4;           ADD  ..ADD MQ.1 TO ITSELF
0504 BF;           PHI  MQ  ..STORE BACK
0505 D5;           SEP RETN  ..
0506 ;             .. 32/16 BIT SIGNED DIVIDE (2'S COMPLEMENT)
0506 ;             .. AC=MQ,AC/OPRN
0506 ;             .. QUOTIENT IN AC , REMAINDER IN MQ
```

```
0506 ;                .. ************ (TO CALL, WRITE) ***********
0506 ;                .. CALL DIVOP ; ,A(OPRN)
0506 ;                ..
0506 46;          DIVOP: LDA LINK .. FETCH OPERAND ADDRESS
0507 BD;              PHI MA .. TO REGISTER MA
0508 46;              LDA LINK
0509 AD;              PLO MA .. FALL INTO DIV0/DIV/DIVQ.
050A ;                .. 32/16 BIT SIGNED DIVIDE (2'S COMPLEMENT)
050A ;                ... OPTION #1: CLEAR MQ AND CHECK FOR ZERO D
050A ;                .. ****AC=MQ,AC/M(R(MA))
050A ;                .. ************ (TO CALL, WRITE) ***********
050A ;                .. ****CALL DIV0
050A ED;          DIV0: SEX MA .. SET X TO POINT TO DIVISOR (0)
050B 9F;              GHI AC .. LOOK AT AC SIGN (0)
050C FC80;            ADI #80 .. COPY IT TO DF (0)
050E F800;            LDI #00 .. EXTEND #00 IF POSITIVE, (0)
0510 3B14;            BNF *+#04 .. (0)
0512 F8FF;            LDI #FF .. #FF IF NEGATIVE. (0)
0514 BF;              PHI MQ .. GIVING +0 OR -0 IN MQ (0)
0515 AF;              PLO MQ .. (0)
0516 4D;              LDA MA .. CHECK FOR ZERO DIVISOR (0)
0517 F1;              OR .. (0)
0518 2D;              DEC MA .. (DON'T FORGET TO FIX POINTER)(0)
0519 FD00;            SDI #00 .. IF ZERO, CALL IT DIVIDE CHECK(0)
051B 3B1F;            BNF DIV  ..GO ON IF NO DIVIDE CHECK ERR
051D D5;              SEP RETN ..AND RETURN WITH DF=1 (0)
051E ;                ... OPTION #2: PERMIT 32-BIT DIVIDEND; (/)
051E ;                ... MAKE SURE QUOTIENT DOES NOT EXCEED 16 BITS(/)
051E ;                .. ************ (TO CALL, WRITE) ***********
051E ;                ..****** CALL DIV
051E 9F;          DIV:     GHI MQ   ..SAVE PARTIAL DIVIDEND(/)
051F BC;              PHI CR  ..INTO CR.1(/)
0520 8F;              GLO MQ    ..(/)
0521 AC;              PLO CR  ..AND CR.0(/)
0522 D4;              SEP CALL  ..CALL SHIFT LEFT ROUTINE(/)
0523 04E8;            ,A(DSHL)   ..(/)
0525 9F;              GHI AC  ..ALSO SHIFT HIGH OF AC(/)
0526 FA80;            ANI #80  ..(WHICH IS THE 16TH BIT)(/)
0528 322C;            BZ D7  ..BUT IF 0,SKIP SHIFT(/)
052A 1F;              INC MQ  ..IF 1, SHIFT INTO MQ(/)
052B 38;              ,#38  ...(/)
052C 8C;          D7:     GLO CR ..SEE IF MQ.0 =0(/)
052D 3A34;            BNZ D2   ..IF NOT GO THROUGH CHECKING STEPS(/)
052F 9C;              GHI CR  ..SEE IF MQ.1 IS #40(/)
0530 FB40;            XRI #40  ..WHICH SHOULD RESULT 0 IF TRUE(/)
0532 323D;            BZ D4       ..IF TRUE,SKIP NORMAL CHECKING(/)
0534 9C;          D2:     GHI CR    ..CHECK IF HIGH 2 BITS OF MQ ARE THE SAME(/)
0535 FAC0;            ANI #C0    ..TAKE OUT 2 HIGH BITS(/)
0537 FD00;            SDI #00  ..SEE IF THEY ARE THE SAME(/)
0539 FC80;            ADI #80    ..IF NOT,HIGH BIT IS 1(/)
053B 33C0;            BDF DVXX  ..SET DF AND RETURN(/)
053D 9C;          D4:     GHI CR   ..LOOK AT THE SIGNS OF DIVND AND DIVS R(/)
053E ED;              SEX MA   ..POINT X TO DIVSR(/)
053F F3;              XOR  ..(/)
0540 FA80;            ANI #80  ..IF DIFFERENT, RESULT 80(/)
0542 FD00;            SDI #00  ..SET DF IF SAME(/)
0544 D4;              SEP CALL ..CALL ABS(MQ-(MA)) SUB(/)
```

```
0545 04D5;            ,A(DABS)  ..(/)
0547 ED;              SEX MA    ..SET X TO DIVSOR(/)
0548 9C;              GHI CR  ..LOOK AT THE SIGN OF DIVIND(/)
0549 FC80;            ADI #80    ..DF=1 IF DIVND IS NEG(/)
054B 3B5C;            BNF DV2    ..IF NOT, DON'T COMPLMENT DIFFERENCE(/)
054D 9F;              GHI MQ  ..COMPLEMENT MQ.1(/)
054E FD00;            SDI #00   ..BY SUBTRACTING FROM 0(/)
0550 BF;              PHI MQ    ..STORE BACK INTO MQ.1(/)
0551 8F;              GLO MQ  ..DO THE SAME FOR MQ.0(/)
0552 325C;            BZ  DV2    ..BUT IF 0,NO NEED TO(/)
0554 9F;              GHI MQ  ..(/)
0555 FF01;            SMI #01   ..OTHERWISE DO BORROW FROM MQ.1(/)
0557 BF;              PHI MQ  ..(/)
0558 8F;              GLO MQ  ..DO COMPLMENTATION FOR MQ.0(/)
0559 FD00;            SDI #00   ..BY SUBTRACTING FROM 0(/)
055B AF;              PLO MQ  ..AND PUT BACK INTO MQ.0(/)
055C 9F;          DV2:     GHI MQ  ..NOW LOOK AT THE DIFF OF MQ&(MA)(/)
055D 3265;            BZ D10    ..IF 0, CHECK MQ=0 SPECIAL CASES(/)
055F FC80;            ADI #80   ..CHECK IF MQ IS NEGATIVE(/)
0561 33C5;            BDF  DDQ    ..IF YES, NO PROBLEM(/)
0563 30BF;            BR D9     ..IF NOT 0 NOR NEG, DIVND IS TOO LARGE(/)
0565 8F;          D10:   GLO MQ    ..HERE WE CHECK DIFF=0 CASES(/)
0566 FCFF;            ADI #FE   ..IF MQ.0 IS NOT EITHER 0 OR 1(/)
0568 33C0;            BDF DVXX    ..THEN DIVND IS STILL TOO LARGE(/)
056A FBFF;            XRI #FF   ..RESULT 0 IF MQ.0 WAS 1(/)
056C 3AA1;            BNZ DVH   ..IF NOT, MUST BE 0, GO TO DVH(/)
056E 9C;              GHI CR    ..SEE IF DIVND IS NEGATIVE(/)
056F FA80;            ANI #80    ..RESULT 80 IF YES(/)
0571 32BF;            BZ   D9   ..IF DIVND POSITIVE, IT CANNOT DIVIDE(/)
0573 F4;              ADD   ..SEE IF DIVISOR IS POSITIVE(/)
0574 33C0;            BDF DVXX    ..IF NEGATIVE, DIVND CANNOT DIVIDE(/)
0576 9F;              GHI AC    ..NOW CHECK IF AC+(MA)>0 IF LOW AC IS 0(/)
0577 F4;              ADD    ..AC.1+(MA.1)(/)
0578 BF;              PHI MQ   ..AND STORE SUM IN MQ.1(/)
0579 1D;              INC MA   ..DO THE SAME FOR LOW AC &(MA.0)(/)
057A 8F;              GLO AC    ..LOAD AC.0 INTO D(/)
057B F4;              ADD   ..AC.0 TO (MA.0)(/)
057C AF;              PLO MQ   ..(/)
057D 3B83;            BNF *+#06   ..SKIP CARRY OVER IF NO CARRY OUT(/)
057F 9F;              GHI MQ   ..DO CARRY OVER(/)
0580 FC01;            ADI #01    ..BY ADD 1 TO MQ.1(/)
0582 BF;              PHI MQ   ..(/)
0583 F0;              LDX   ..NOW HECK LOW BIT (MA) IS 0 OR 1(/)
0584 F6;              SHR   ..SHIFT THAT LOW BIT OUT(/)
0585 2D;              DEC MA   ..REMEMBER TO RESET MA(/)
0586 9F;              GHI MQ   ..READY TO ADD #80(/)
0587 3B8C;            BNF *+#05    ..IF LOW BIT OF AC IS 1(/)
0589 FC80;            ADI #80    ..TO MQ.1(/)
058B BF;              PHI MQ   ..SAVE STATUS(/)
058C FC80;            ADI #80   ..SEE IF MQ >0(/)
058E 9C;              GHI CR   ..SEE IF DIVND IS POSITIVE(/)
058F FA80;            ANI #80   ..BY TAKING OUT THE SIGN BIT(/)
0591 3A97;            BNZ *+#06    ..IF NOT,GO THROUGH DIFF CHECK(/)
```

```
0593 3BBE;           BNF D9  ..IF AC+(MA) >=0, BAD(/)
0595 30C5;           BR DDQ  ..OTHERWISE IS OK(/)
0597 9F;             GHI MQ  ..MQ=0?(/)
0598 33C0;           BDF DVXX  ..IF NEGATIVE, THEN OUT(/)
059A 3AC5;           BNZ DDQ  ..IF NOT, THEN NO PROBLEM(/)
059C 8F;             GLO MQ  ..MAKE SURE MQ.0 IS 0 TOO(/)
059D 32BE;           BZ D9  ..IF YES, THEN RETURN WITH DF=1(/)
059F 30C5;           BR  DDQ  ..OR ELSE GO TO DIVIDE(/)
05A1 9C;       DVH:     GHI CR  ..SEE IF THE OPERANDS SIGNS DIFF(/)
05A2 F3;             XOR  ..BY COMPARING SIGN BITS(/)
05A3 FA80;           ANI #80   ..AND TAKE OUT THAT BIT(/)
05A5 3A76;            BNZ D10+#11  ..IF SINGS DIFF, IT'S OK(/)
05A7 9C;             GHI CR  ..OTHERWISE TEST SIGN OF DIVND(/)
05A8 FC80;           ADI #80  ..IF POSITIVE, RETURN WITH DF=1(/)
05AA 3BBE;           BNF D9  ..RETURN WITH DF=1(/)
05AC 1D;             INC MA  ..SEE IF LOW BIT OF  AC IS 0 OR 1(/)
05AD F0;             LDX  ..LOAD THAT IN D(/)
05AE 2D;             DEC MA  ..REMEMBER TO RESET MA(/)
05AF F6;             SHR   ..SHIFT THAT BIT OUT(/)
05B0 8E;             GLO AC  ..IF AC IS NOT 0 ,NOT PROBLEM(/)
05B1 3AC5;           BNZ DDQ  ..GO TO DIVIDE(/)
05B3 9E;             GHI AC  ..READY TO CHECK #80 IF LOW BIT AC IS 1(/)
05B4 33BA;           BDF *+#06  ..(/)
05B6 32BE;           BZ  D9  ..NO GOOD, RETURN WITH DF=1(/)
05B8 30C5;           BR DDQ  ..ANY THING ELSE IS OK(/)
05BA FB80;           XRI #80  ..IF AC.1 IS'NT #80(/)
05BC 3AC5;           BNZ DDQ  ..IT'S OK(/)
05BE FF00;     D9:    SMI #00  ..DF IS SET TO 1(/)
05C0 9C;       DVXX:     GHI CR  ..PUT ORGINAL DIVND(/)
05C1 BF;             PHI MQ  ..INTO MQ(/)
05C2 8C;             GLO CR ..(/)
05C3 AF;             PLO MQ    ..(/)
05C4 D5;             SEP RETN   ..AND RETURN WITH DF=1(/)
05C5 9C;       DDQ:    GHI CR  ...PUT ORGINAL DIVND(/)
05C6 BF;             PHI MQ  ..BACK INTO MQ(/)
05C7 8C;             GLO CR ..(/)
05C8 AF;            PLO MQ  ..(/)
05C9 ;               ... OPTION #3: ASSUME BENIGN PROGRAM; NOCHECK
05C9 ;               .. ************ (TO CALL, WRITE) ***********
05C9 ;               .. ****CALL DIVQ
05C9 9F;       DIVQ:    GHI MQ  ..LOOK AT DIVIDEND SIGN
05CA FC80;           ADI #80 .. IF POSITIVE,
05CC F890;           LDI #90 .. PLAN TO BEGIN WITH SUBTRACT,
05CE 3BD2;           BNF *+#04 .. (ALSO SAVE SIGN OF DIVIDEND)
05D0 F850;           LDI #50 .. OTHER WISE BEGIN WITH ADD.
05D2 AC;             PLO CR .. SET ITERATION COUNT IN CR.0
05D3 D4;       DVL:     SEP CALL  ..CALL LEFT SHIFT SUB
05D4 04F8;           ,A(DSHL)
05D6 8E;             GLO AC .. NOW DO AC
05D7 52;             STR SP ..
05D8 F4;             ADD
05D9 AE;             PLO AC
05DA 9E;             GHI AC
05DB 52;             STR SP ..
05DC 3BE4;           BNF *+#08
05DF FC01;           ADI #01
05E0 3BE4;           BNF *+#04
05E2 F5;             SD .. DON'T LOSE CARRY OUT
05E3 38;             ,#38
```

```
05E4 F4;             ADD
05E5 BE;             PHI AC
05E6 3BE9;           BNF *+#03 .. BIT SHIFTED OUT OF AC.1,
05E8 1F;             INC MQ .. GOES INTO MQ.0
05E9 ED;             SEX MA ..
05EA 8C;             GLO CR .. NOW, WAS THAT ADD, OR SUBTRACT?
05EB F3;             XOR .. IT DEPENDS ON SAVED FLAG,
05EC FC80;           ADI #80 .. AND SIGN OF DIVISOR.
05EE D4;             SEP  CALL  ..CALL ABS(MQ-(MA)) SUB
05EF 04D5;           ,A(DABS)
05F1 ED;             SEX MA
05F2 2C;             DEC CR .. COUNT DOWN ITERATION COUNTER
05F3 8C;             GLO CR
05F4 FA7F;           ANI #7F
05F6 3BFB;           BNF *+#05 .. TEST CARRY OUT OF ADD/SUBTRACT
05F8 1E;             INC AC .. IF 1, SHIFT INTO QUOTIENT,
05F9 F980;           ORI #80 .. AND FLAG NEXT OP AS SUBTRACT.
05FB AC;             PLO CR .. OTHERWISE IT'S ADD.
05FC FA3F;           ANI #3F .. LOOK AT COUNTER;
05FE 3AD3;           BNZ DVL .. IF NOT 0, LOOP BACK;
0600 3320;           BDF DVR .. AT END, CHECK REMAINDER ADJUST.
0602 38;             ,#38
0603 1E;       DVC: INC AC .. (FINAL DIVIDE STEP)
0604 F3;             XOR .. BE SURE TO GET POLARITY
0605 FC80;           ADI #80 .. OF ADJUSTMENT RIGHT...
0607 1D;             INC MA .. YES, AND DIVISOR BACK ON,
0608 8F;             GLO MQ .. TO CORRECT FOR FINAL SUBTRACT,
0609 3316;           BDF DVM .. (ADDING NEGATIVE IS SUBT.)
060B F4;             ADD .. WHICH SHOULDN'T HAVE.
060C AF;             PLO MQ
060D 2D;             DEC MA
060E 9F;             GHI MQ
060F 3B13;           BNF *+#04
0611 FC01;           ADI #01 .. NO NEED TO SAVE CARRY OUT
0613 F4;             ADD
0614 301F;           BR DVR-#01
0616 F7;       DVM: SM .. SAME THING,
0617 AF;             PLO MQ .. EXCEPT, FOR NEGATIVE DIVISOR.
0618 2D;             DEC MA
0619 9F;             GHI MQ
061A 331F;           BDF *+#04
061C FF01;           SMI #01
061E F7;             SM
061F BF;             PHI MQ
0620 9F;       DVR: GHI MQ .. IF REMAINDER IS NOT ZERO,
0621 3A26;           BNZ *+#05
0623 8F;             GLO MQ
0624 3234;           BZ DVN .. BUT IT IS; NO PROBLEM.
0626 8C;             GLO CR .. IF NOT ZERO,
0627 FA40;           ANI #40 .. IT SHOULD BE SAME SIGN
0629 FCC0;           ADI #C0 .. AS ORIGINAL DIVIDEND.
062B 9F;             GHI MQ
062C 3330;           BDF *+#04
062E FB80;           XRI #80
0630 FC80;           ADI #80 .. IF NOT, WE NEED
0632 3B03;           BNF DVC .. ONE MORE DIVIDE ITERATION.
0634 F0;       DVN: LDX .. FINALLY, IF DIVISOR NEGATIVE,
0635 FC80;           ADI #80
0637 3B44;           BNF DVX   ..(IT'S NOT; WE ARE DONE)
0639 8E;             GLO AC .. COMPLEMENT QUOTIENT,
063A FBFF;          XRI #FF  .. BY INVERTING IT
063C AF;            PLO AC
063D 9F;            GHI AC
063E FBFF;          XRI #FF
0640 BF;            PHI AC
0641 1F;            INC AC .. THEN INCREMENTING
0642 FC00;          ADI #00 .. ALSO CLEAR DF.
0644 D5;       DVX: SEP RETN .. DF=0 IF DIVIDE SUCCESSFUL.
0645 ;              ..
0645 ;              .. BINARY TO DECIMAL CONVERSION
0645 ;              .. ****DECIMAL NUMBER = AC
0645 ;              .. DECIMAL NUMBER = SIGN,N0,N1,N2,N3,..
0645 ;              .. SIGN=#0B +
0645 ;              .. SIGN=#0D -
0645 ;              .. N0=10**0 DIGIT
0645 ;              .. N1=10**1 DIGIT, ETC
0645 ;              .. ************ (TO CALL, WRITE) ***********
0645 ;              .. ****CALL CBD ; ,A( NUMBER) ; LENGTH
0645 46;       CBD: LDA LINK .. GET DECIMAL ADDRESS
0646 BA;            PHI AR
0647 46;            LDA LINK
0648 AA;            PLO AR
0649 46;            LDA LINK .. GET SIZE (#)
064A AB;            PLO NR
064B BB;            PHI NR
064C F80D;          LDI #0D .. STORE MINUS (+)
064E 5A;            STR AR
064F 9F;            GHI AC .. SET DF IF POSITIVE
0650 FD7F;          SDI #7F
0652 EA;            SEX AR
0653 D4;            SEP CALL .. NOW CHANGE SIGN TO MATCH
0654 06A0;          ,A( DZS)
0656 335D;          BDF *+#07
0658 D4;            SEP CALL .. SET AC POSITIVE
0659 0415;          ,A( SDCON)
065B 00;            ,#00
065C 00;            ,#00
065D D4;            SEP CALL .. PUSH AC INTO STACK,
065E 06F9;          ,A( PUSHAC)
0660 F800;          LDI #00 .. ONLY TO REPLACE IT
0662 5D;            STR MA .. BY A DIVISOR OF 10
0663 1D;            INC MA
0664 F80A;          LDI #0A
0666 5D;            STR MA
0667 2D;            DEC MA
0668 2B;       CBL: DEC NR .. DECREMENT DIGIT COUNT
0669 8B;            GLO NR .. CHECK END
066A 327F;          BZ CBX .. ZERO IS OVERFLOW.
066C F800;          LDI #00
066E AE;            PLO MQ
066F BE;            PHI MQ
0670 D4;            SEP CALL .. DIVIDE BY 10
0671 05C9;          ,A( DIVQ)
0673 1A;            INC AR .. (<)
0674 8E;            GLO MQ .. REMAINDER IS THIS DIGIT
0675 5A;            STR AR
0676 8F;            GLO AC .. CHECK FOR NON-ZERO
0677 3A68;          BNZ CBL
0679 9F;            GHI AC
067A 3A68;
```

```
067A 3A68;          BNZ CBL  .. CONTINUE IF MORE DIGITS
067C F6;            SHR .. CLEAR DF FOR SUCCESS
067D 38;            ,#38
067E F5;       CBX: SD .. SET DF FOR FAILURE
067F 12;            INC SP .. POP STACK (THAT TEN)
0680 12;            INC SP
0681 D5;            SEP RETN .. DONE.
0682 ;              ..
0682 ;              ..
0682 ;              ..
0682 ;              .. RESTORE RESULT POINTER
0682 3B85;     DRP: BNF *+#03 .. SAVE FLAG IN NR.0
0684 1B;            INC NR .. (ASSUME WAS 0)
0685 E2;            SEX SP .. PUSH SAVED DIGIT COUNT
0686 9B;            GHI NR .. INTO STACK
0687 52;            STR SP
0688 8A;            GLO AR .. SUBTRACT FROM RESULT ADDRESS
0689 F7;            SM .. (<)
068A AA;            PLO AR
068B 3391;          BDF *+#06 .. (=)(<)
068D 9A;            GHI AR .. BORROW FROM HIGH BYTE (=)
068E FF01;          SMI #01 .. (=)(<)
0690 BA;            PHI AR .. (=)
0691 8B;            GLO NR .. RECOVER DF
0692 F6;            SHR
0693 F0;            LDX .. POP SAVED COUNT,
0694 AB;            PLO NR .. INTO NR.0
0695 D5;            SEP RETN .. RETURN
0696 ;              ..
0696 5A;       DZR: STR AR .. STORE A BYTE
0697 1A;            INC AR .. ADVANCE TO NEXT (<)
0698 2B;            DEC NR .. CHECK COUNT
0699 8B;            GLO NR
069A 3282;          BZ DRP .. AT 0, GO RESTORE POINTERS
069C F800;          LDI #00 .. CONTINUE STORING 0
069E 3096;          BR DZR
06A0 ;              .. COMPARE SIGNS, AND CLEAR RESULT (ADDRESS
06A0 F80B;     DZS: LDI #0B .. IF DF=0, TRY "+" (+)
06A2 3BA6;          BNF *+#04
06A4 F80D;         LDI #0D .. OTHERWISE "-" (+)
06A6 F3;            XOR .. COMPARE TO PRESENT M(R(X))
06A7 32AB;          BZ *+#04
06A9 F806;          LDI #06 .. NOT EQUAL, SO CONDITION MINUS(+)
06AB FB0B;          XRI #0B .. IF EQUAL SET PLUS (+)
06AD 3096;          BR DZR .. GO STORE, WITH ZERO IN RESULT.
06AF ;              ..
06AF ;              .. 16-BIT ADD TO AC, OPERAND ADDRESS IN CALL
06AF ;              .. ****AC=AC+OPRN .. OPRN=2 BYTE OPERAND
06AF ;              .. ************ (TO CALL, WRITE) ***********
06AF ;              .. ****CALL ADDOP ; ,A(OPRN)
06AF ;              ..
06AF ;              ..
06AF 46;     ADDOP: LDA LINK .. FETCH OPERAND ADDRESS
06B0 BD;            PHI MA .. TO REGISTER MA
06B1 46;            LDA LINK
06B2 AD;            PLO MA .. FALL INTO ADD
06B3 ;              .. 16-BIT ADD TO AC, OPERAND ADDRESS IN REGI
06B3 ;              .. CALL HERE IF OPRN ADDRESS
06B3 ;              .. IS IN REGISTER MA
06B3 ;              .. ****AC=AC+M(R(MA))
```

```
06B3 ;          .. ************ (TO CALL, WRITE) ***********
06B3 ;          .. ****CALL ADD
06B3 ;          ..
06B3 ED;     ADD:  SEX MA  ..CHECK SIGN BIT OF AC
06B4 9F;        GHI AC  ..GET THE OPERAND
06B5 F3;         XOR  ..AND OPERAND @MA
06B6 FA80;      ANI #80  ..RESULT A 1 IF DIFF
06B8 52;        STR SP  ..STORE ON STACK
06B9 1D;        INC MA  ..POINT TO LOW 8 BITS
06BA 8F;        GLO AC .. FETCH AC LOW 8
06BB F4;        ADD .. ADD LOW 8 FROM MEMORY
06BC AF;        PLO AC .. PUT IT BACK
06BD 2D;        DEC MA .. POINT TO HIGH 8 MEMORY LOCATION
06BE 9F;        GHI AC
06BF 3BC7;      BNF ADDNC
06C1 FC01;      ADI #01 .. ADD IN CARRY OUT OF LOW 8
06C3 3BC7;      BNF ADDNC .. (OMIT 3 LINES IF CARRY OUT NOT
06C5 F5;        SD .. OOPS, THAT CARRIED OUT, LEAVING 0:
06C6 38;        ,#38 .. FINISH ADD, FORCING CARRY TO 1.
06C7 F4;     ADDNC: ADD .. ADD HIGH 8
06C8 BF;        PHI AC .. PUT IN AC.
06C9 42;        LDA SP  ..LOAD THE STORED COMPARING SIGN BIT
06CA 22;        DEC SP  ..RESET STACK PTR
06CB 3AD6;      BNZ ADDRT  ..NOT POSSIBLE
06CD 9F;        GHI AC  ..OTHERWISE SEE IF SUM IS RIGHT
06CE F3;        XOR  ..BY COMPARING SIGN BITS
06CF FA80;      ANI #80  ..TAKE OUT THE COMPARING BIT
06D1 32D6;      BZ ADDRT  ..IF SAME,THEN NOT OVERFLOWED
06D3 FCFF;      ADI #FF ..OTHERWISE, SET DF=1
06D5 38;        ,#38
06D6 F6;     ADDRT:     SHR   ..SHIFT OUT 0 INTO DF
06D7 D5;        SEP RETN  ..RETURN TO MAIN
06D8 ;          ..
06D8 ;          .. 16-BIT CONSTANT ADD TO AC.
06D8 ;          .. CALL HERE FOR ADD CONSTANT TO AC
06D8 ;          .. ****AC=AC+CONSTANT
06D8 ;          .. ************ (TO CALL, WRITE) ***********
06D8 ;          .. ****CALL ADDCON ; ,CONSTANT
06D8 ;          ..
06D8 86;     ADDCON: GLO LINK .. COPY LINK TO MA
06D9 AD;        PLO MA
06DA 96;        GHI LINK
06DB BD;        PHI MA
06DC 16;        INC LINK .. INCREMENT PAST DATUM
06DD 16;        INC LINK
06DE 30B3;      BR ADD .. GO ADD.
06E0 ;          ..
06E0 ;          .. 16-BIT ADD FROM TOP OF STACK
06E0 92;     ADDST: GHI SP .. COPY STACK POINTER
06E1 BD;        PHI MA .. TO MA REGISTER
06E2 82;        GLO SP
06E3 AD;        PLO MA
06E4 1D;        INC MA .. ADVANCE TO SUB-TOP (@)
06E5 1D;        INC MA .. (@)
06E6 1D;        INC MA .. (@)
06E7 30B3;      BR ADD .. GO DO IT
06E9 ;          ..
06E9 ;          ..
06E9 ;          ..
06E9 ;          .. PUSH AC INTO STACK
```

```
06E9 ;          .. STACK POINTER = SP
06E9 ;          .. ************ (TO CALL, WRITE) ********
06E9 ;          .. ****CALL PUSHAC
06E9 ;          .. PUSH AC (UNDER TOP OF STACK)
06E9 92;     PUSHAC: GHI SP .. COPY STACK POINTER TO MA
06EA BD;        PHI MA
06EB 82;        GLO SP
06EC AD;        PLO MA
06ED 1D;        INC MA .. NOW SLICE OFF TOP 2 BYTES,
06EE 1D;        INC MA
06EF ED;        SEX MA
06F0 F0;        LDX .. TO MAKE A 2-BYTE HOLE.
06F1 52;        STR SP
06F2 2D;        DEC MA
06F3 22;        DEC SP
06F4 F0;        LDX
06F5 52;        STR SP
06F6 9F;        GHI AC .. NOW STUFF AC INTO THE HOLE.
06F7 5D;        STR MA
06F8 8F;        GLO AC
06F9 1D;        INC MA
06FA 5D;        STR MA
06FB 2D;        DEC MA .. LEAVE MA POINTING TO IT.
06FC 22;        DEC SP
06FD D5;        SEP RETN .. (AC UNCHANGED)
06FE ;          .. PUSH CR, MA, MQ (UNDER TOP OF STACK)
06FE 8D;     PUSHCQ: GLO MA .. FIRST PUSH MA ONTO TOP
06FF 52;        STR SP
0700 22;        DEC SP
0701 9D;        GHI MA
0702 52;        STR SP
0703 92;        GHI SP .. NOW COPY SP TO MA
0704 BD;        PHI MA
0705 82;        GLO SP
0706 AD;        PLO MA
0707 22;        DEC SP
0708 1D;        INC MA .. THEN ADJUST IT
0709 1D;        INC MA
070A 1D;        INC MA .. TO POINT INTO OLD TOP
070B 8F;        GLO MQ .. CONTINUE PUSHING, MQ
070C 52;        STR SP
070D 22;        DEC SP
070E 9F;        GHI MQ
070F 52;        STR SP
0710 22;        DEC SP
0711 ED;        SEX MA .. NOW COPY OLD TOP TO NEW TOP
0712 F0;        LDX
0713 52;        STR SP
0714 22;        DEC SP
0715 2D;        DEC MA
0716 F0;        LDX
0717 52;        STR SP
0718 22;        DEC SP
0719 9C;        GHI CR .. FINALY INSERT CR IN HOLE
071A 5D;        STR MA
071B 1D;        INC MA
071C 8C;        GLO CR
071D 5D;        STR MA
071E D5;        SEP RETN .. (MA IS GARBAGE OUT)
071F ;          ..
```

```
071F ;                  .. POP STACK INTO MQ, MA, CR (UNDER TOP OF S
071F 12;        POPCQ: INC SP
0720 92;            GHI SP .. COPY STACK POINTER TO MA
0721 BD;            PHI MA
0722 82;            GLO SP
0723 AD;            PLO MA
0724 1D;            INC MA .. ADJUST POINTER TO CR DATUM
0725 1D;            INC MA
0726 1D;            INC MA
0727 1D;            INC MA
0728 1D;            INC MA
0729 1D;            INC MA
072A 4D;            LDA MA .. FETCH IT
072B BC;            PHI CR
072C 4D;            LDA MA
072D AC;            PLO CR
072E 2D;            DEC MA .. COPY TOP OF STACK INTO GAP
072F 2D;            DEC MA
0730 42;            LDA SP
0731 5D;            STR MA
0732 1D;            INC MA
0733 42;            LDA SP
0734 5D;            STR MA
0735 42;            LDA SP .. THEN POP MQ
0736 BF;            PHI MQ
0737 42;            LDA SP
0738 AF;            PLO MQ
0739 42;            LDA SP .. FINALLY POP MA
073A BD;            PHI MA
073B 42;            LDA SP
073C AD;            PLO MA
073D 22;            DEC SP
073E D5;            SEP RETN
073F ;              .. 16-BIT ACCUMULATOR LOAD (ADDRESS IN CALL)
073F ;              .. ****AC=OPRN
073F ;              .. ************ (TO CALL, WRITE) ***********
073F ;              .. ****CALL LOADOP ; ,A(OPRN)
073F ;            ..
073F 46;        LOADOP: LDA LINK .. FETCH ADDRESS
0740 BD;            PHI MA .. TO MA REGISTER
0741 46;            LDA LINK
0742 AD;            PLO MA .. FALL INTO LOAD
0743 ;              .. 16-BIT ACCUMULATOR LOAD (ADDRESS IN MA)
0743 ;              .. CALL HERE IF OPERAND
0743 ;              .. ADDRESS IN REGISTER MA
0743 ;              .. ****AC=M(R(MA))
0743 ;              .. ************ (TO CALL, WRITE) ***********
0743 ;              .. ****CALL LOAD
0743 ;              ..
0743 4D;        LOAD: LDA MA .. FETCH HIGH 8
0744 BF;            PHI AC
0745 4D;            LDA MA .. NOW LOW 8
0746 AF;            PLO AC .. LEAVE MA AT NEXT DOUBLE-BYTE
0747 D5;            SEP RETN .. GEE, THAT WAS QUICK.
0748 ;              ..
0748 ;              .. 16-BIT ACCUMULATOR LOAD FROM CONSTANT IN
0748 ;              .. ****AC=CONSTANT
0748 ;              .. ************ (TO CALL, WRITE) ***********
0748 ;              .. ****CALL LODCON ; ,CONSTANT
0748 ;              ..
```

```
0748 46;        LODCON: LDA LINK .. FETCH HIGH 8
0749 BF;            PHI AC
074A 46;            LDA LINK .. THEN LOW 8
074B AF;            PLO AC
074C D5;            SEP RETN
074D ;              ..
074D ;              .. 16-BIT ACCUMULATOR STORE (ADDRESS IN CALL
074D ;              .. ****OPRN=AC
074D ;              .. ************ (TO CALL, WRITE) ***********
074D ;              .. ****CALL STOROP ; ,A(OPRN)
074D ;              ..
074D 46;        STOROP: LDA LINK .. FETCH ADDRESS INTO MA
074E BD;            PHI MA
074F 46;            LDA LINK
0750 AD;            PLO MA .. THEN FALL INTO STORE
0751 ;              .. 16-BIT ACCUMULATOR STORE (ADDRESS IN MA)
0751 ;              .. ****M(R(MA))=AC
0751 ;              .. ************ (TO CALL, WRITE) ***********
0751 ;              .. ****CALL STORE
0751 ;              ..
0751 9F;        STORE: GHI AC .. FIRST HIGH 8
0752 5D;            STR MA
0753 1D;            INC MA .. INCREMENT MA, SINCE STR DOESN'T
0754 8F;            GLO AC .. NOW LOW 8
0755 5D;            STR MA
0756 1D;            INC MA .. LEAVE MA POINTING TO NEXT WORD
0757 D5;            SEP RETN .. QUIT
0758 ;              ..
0758 ;              ..
0758 ;              .. 16-BIT COMPARE, OPERAND ADDRESS IN CALL
0758 ;              .. ****AC-OPRN  (DF SET IF 0 OR +)
0758 ;              .. ************ (TO CALL, WRITE) ***********
0758 ;              .. ****CALL COMOP ; ,A(OPRN)
0758 ;              ..
0758 46;        COMPOP: LDA LINK .. FETCH ADDRESS
0759 BD;            PHI MA .. TO MA REGISTER
075A 46;            LDA LINK
075B AD;            PLO MA
075C ;              .. 16-BIT COMPARE, OPERAND ADDRESS IN REGIST
075C ;            .. CALL HERE IF OPERAND
075C ;              .. ADDRESS IN REGISTER MA
075C ;              .. ****AC-M(R(MA))  (DF SET IF 0 OR +)
075C ;              .. ************ (TO CALL, WRITE) ***********
075C ;              .. ****CALL COMP
075C ;              ..
075C ED;        COMP: SEX MA .. COMPARE HIGH 8 FRST
075D 9F;            GHI  AC  ..CHECK IF SIGN OF OPERANDS ARE SAME
075E F3;            XOR  ..BY LOOKING AT THE HIGHEST SIGN BIT
075F FA80;          ANI #80  ..RESULT A '1' IF NEGATIVE
0761 3A6C;         BNZ CNE  ..IF NOT, THEN GO TO CNE
0763 9F;            GHI AC
0764 F7;            SM
0765 3A6D;          BNZ CMPX .. NOT EQUAL QUITS
0767 1D;            INC MA .. TRY LOW 8
0768 8F;            GLO AC
0769 F7;            SM
076A 2D;            DEC MA .. LEAVE MA POINTING TO IT
076B 38;            ,#38
076C F4;        CNE:     ADD  ..SEE IF OPERAND IS NEGATIVE
076D D5;        CMP
```

```
076D D5;      CMPX:      SEP RETN  ..DF=1 IF AC>=(MA)
076E ;             ..
076E ;             ..
076E ;             .. DECIMAL TO BINARY CONVERSION
076E ;             .. ****AC=DECIMAL NUMBER OF N BYTES
076E ;             .. DECIMAL NUMBER = SIGN,N0,N1,N2,N3 ...
076E ;             .. SIGN = #0B +
076E ;             .. SIGN = #0D -
076E ;             .. N0=10**0 DIGIT
076E ;             .. N1=10**1 DIGIT, ETC
076E ;             .. ************ (TO CALL, WRITE) ***********
076E ;             .. ****CALL CDB ; ,A(NUMBER) ; LENGTH
076E ;             ..
076E 46;      CDB: LDA LINK .. GET DECIMAL ADDRESS
076F BA;           PHI AR
0770 46;           LDA LINK
0771 AA;           PLO AR
0772 46;           LDA LINK .. AND SIZE (#)
0773 AB;           PLO NR
0774 2B;           DEC NR  ..MINUS ONE FOR SIGN
0775 EA;           SEX AR .. LOOK AT SIGN (!)
0776 4A;           LDA AR .. (<)
0777 FB0D;         XRI #0D .. IS IT MINUS? (+)
0779 327D;    CDS: BZ *+#04 .. #00 IF YES,
077B F8FF;         LDI #FF .. #FF IF NO:
077D BB;           PHI NR .. REMEMBER THAT
077E F800;         LDI #00 .. INITIALIZE AC TO 0
0780 AF;           PLO AC
0781 BF;           PHI AC
0782 8B;           GLO NR .. GO TO OTHER END
0783 52;           STR SP .. BY ADDING SIZE
0784 E2;           SEX SP
0785 8A;           GLO AR .. TO ADDRESS
0786 F4;           ADD .. (<)
0787 AA;           PLO AR
0788 3B8F;         BNF CDL .. (=)(<)
078A 9A;           GHI AR .. PROPAGATE CARRY (=)
078B FC01;         ADI #01 .. ACROSS BYTES (=)(<)
078D BA;           PHI AR .. (=)
078E 8B;      CDL: GLO NR .. ANY MORE DIGITS
078F 32C1;         BZ CDX .. NO
0791 2B;           DEC NR .. YES.
0792 D4;           SEP CALL .. MULTIPLY AC * 10
0793 06F9;         ,A( PUSHAC)
0795 D4;           SEP CALL .. FROM STACK
0796 0748;         ,A( LODCON)
0798 00;           ,#00
0799 0A;           ,#0A
079A D4;           SEP CALL
079B 0475;         ,A( MPY)
079D 3BA2;         BNF *+#05  ..IF NOT OVERFLOWED, SKIP EXIT
079F 12;           INC SP  ..RESET STACK
07A0 12;           INC SP  ..
07A1 D5;           SEP RETN   ..OVERFLOW EXIT
07A2 D4;           SEP CALL .. SAVE PRODUCT IN STACK
07A3 0751;         ,A( STORE)
07A5 2A;           DEC AR .. (<)
07A6 4A;           LDA AR .. NOW GET DIGIT
07A7 2A;           DEC AR
07A8 FA0F;         ANI #0F .. REMOVE OVERBITS
```

```
07AA AF;           PLO AC
07AB F800;         LDI #00
07AD BF;           PHI AC
07AE D4;           SEP CALL .. ADD TO PREVIOUS NUMBER
07AF 06E0;         ,A( ADDST)
07B1 12;      CDO: INC SP
07B2 12;           INC SP
07B3 3B8E;         BNF CDL .. ADVANCE TO NEXT DIGIT
07B5 8B;           GLO NR  ..NO MORE DIGIT?
07B6 3AC0;         BNZ  CDRT   ..OVERFLOWED
07B8 9B;           GHI NR  ..POSITIVE NUMBER?
07B9 3AC0;         BNZ CDRT  ..OVERFLOWED
07BB 2F;           DEC AC  ..IF AC IS #8000 IS OK
07BC 9F;           GHI AC  ..TEST THAT BIT
07BD FC80;         ADI #80   ..
07BF 1F;           INC AC  ..REMEMBER TO RESET AC
07C0 D5;      CDRT:      SEP RETN  ..OVERFLOW RETURN
07C1 9B;      CDX:      GHI NR  ..NO. LOOK AT SIGN
07C2 3AC9;         BNZ *+#07 .. POSITIVE
07C4 D4;           SEP CALL
07C5 0415;         ,A( SDCON) .. NEGATIVE. SUBTRACT FROM 0
07C7 00;           ,#00
07C8 00;           ,#00
07C9 FC00;         ADI #00 .. CLEAR DF
07CB D5;           SEP RETN .. DF=0 IF OK.
07CC ;             ..
07CC ;            ..TEST 16-BIT ACCUMULATOR SIGN/ZERO ......
07CC 16;           INC LINK  ..SKIP OVER NON ZERO RETURN
07CD 16;           INC LINK  ..
07CE D5;           SEP RETN
07CF 9F;      TEST: GHI AC .. FIRST LOOK AT SIGN
07D0 FC80;         ADI #80  ..SET DF IF MINUS
07D2 9F;           GHI AC .. NOW CHECK FOR 0
07D3 3ACC;         BNZ TEST-#03 .. NO.
07D5 8F;           GLO AC
07D6 32CE;         BZ TEST-#01 .. GO TAKE ZERO EXIT.
07D8 30CC;         BR TEST-#03 .. GO TAKE NON-ZERO EXIT.
07DA ;             .. POP STACK (INTO AC) (UNDER TOP)
07DA ;             .. POPS 2 BYTES FROM STACK INTO AC
07DA ;             .. ************ (TO CALL, WRITE) ********
07DA ;             .. ****CALL POPAC
07DA FF00;    POPAC: SMI #00 .. SET F TO REMEMBER THIS ENTRY
07DC 30E0;         BR *+#04
07DE FC00;    POP: ADI #00 .. CLEAR DF FOR THIS ENTRY
07E0 12;           INC SP
07E1 92;           GHI SP .. COPY SP TO MA
07E2 BD;           PHI MA
07E3 82;           GLO SP
07E4 AD;           PLO MA
07E5 1D;           INC MA .. ADJUST TO SUB-TOP OF STACK
07E6 1D;           INC MA
07E7 3BEF;         BNF *+#08
07E9 4D;           LDA MA .. POPPING INTO AC. GET DATUM
07EA BF;           PHI AC
07EB 4D;           LDA MA
07EC AF;           PLO AC
07ED 2D;           DEC MA .. RESTORE POINTER
07EE 2D;           DEC MA
07EF 42;           LDA SP .. NOW CLOSE UP THE GAP
07F0 5D;           STR MA
07F1 1D;           INC MA
07F2 42;           LDA SP
07F3 5D;           STR MA
07F4 1D;           INC MA
07F5 22;           DEC SP
07F6 D5;           SEP RETN .. MA POINTS TO NEW SUB-TOP
07F7 ;             END
0000
```

Appendix B(II) - Arithmetic Subroutine Listing for CDP1802

```
0000 ;
0000 ;
0000 ;             ..COPYRIGHT 1976 RCA CORPORATION
0000 ;            .. COSMAC ARITHMETIC SUBROUTINE PACKAGE
0000 ;            ..
0000 ;            .. VERSION 1.1 FOR CDP1802
0000 ;            ..
0000 ;            .. EXTRACTED FOR BINARY ARITHMETIC
0000 ;            .. AND CONVERSIONS FOR DECIMAL
0000 ;            .. OCCUPIES 1K BYTES
0000 ;            ..
0000 ;            ..
0000 ;            .. DESIGNED FOR STANDARD CSDP SUBROUTINE CALL & RETURN
0000 ;            ..
0000 ;            .. FOR STANDARD LINKAGE,
0000 ;           SP= #02 ...IT SHOULD BE THE STACK POINTER.
0000 ;           PC= #03 ...IT IS THE PROGRAM COUNTER
0000 ;            .. USED BY THESE SUBROUTINES.
0000 ;           CALL= #04 ...IT SHOULD POINT TO THE ROUTINE
0000 ;            .. WHICH EFFECTS SUBROUTINE CALLS.
0000 ;           RETN= #05 ...IT SHOULD POINT TO THE ROUTINE
0000 ;            .. WHICH EFFECTS SUBROUTINE RETURN.
0000 ;           LINK= #06 ...IT SHOULD POINT TO CALL PARAMETER
0000 ;            .. .. (USUALLY OPERAND ADDRESSES AND/OR CONSTANT)
0000 ;            ..
0000 ;            ..
0000 ;            .. THE FOLLOWING REGISTERS MUST BE ASSIGNED
0000 ;       AR= #0A  ..(USED FOR RESULT ADDRESS)
0000 ;
0000 ;       NR= #0B  ..(USED FOR RESULT DIGIT COUNT)
0000 ;            .. 16-BIT BINARY ARITHMETIC ROUTINES
0000 ;            .. THE FOLLOWING REGISTERS MUST BE ASSIGNED
0000 ;       AC= #0F .. 16-BIT ACCUMULATOR=RF.
0000 ;       MQ= #0E .. 16-BIT ACCUMULATOR=RE EXTENSION.
0000 ;       MA= #0D .. (TEMPORARY) OPERAND MEMORY ADDRESS.
0000 ;       CR= #0C .. (TEMPORARY) SCRATCHPAD AND COUNTER.
0000 ;            ..
0000 ;            ..
0000 ;            .. SWAP AC WITH MQ REGISTERS
0000 9E;     SWAPAQ: GHI MQ .. SAVE MQ.1
0001 AC;          PLO CR .. IN CR.0 (COULD HAVE PUSHED ON STAC
0002 9F;          GHI AC .. NOW AC.1 TO MQ.1
0003 BE;          PHI MQ
0004 8E;          GLO MQ .. SAVE MQ.0
0005 BF;          PHI AC .. IN AC.1
0006 8F;          GLO AC .. THEN AC.0 TO MQ.0
0007 AE;          PLO MQ
0008 9F;          GHI AC .. NOW SAVED MQ.0 TO AC.0
0009 AF;          PLO AC
000A 8C;          GLO CR .. FINALLY SAVED MQ.1
000B BF;          PHI AC .. TO AC.1
```

```
000C D5;             SEP RETN
000D ;               .. 16-BIT SUBTRACT AC FROM CONSTANT
000D ;               .. ****AC= CONSTANT-AC
000D ;               .. ************ (TO CALL, WRITE) ***********
000D ;               .. ****CALL SDCON ; ,CONSTANT
000D 86;         SDCON: GLO LINK .. (ESSENTIALLY SAME AS ADCON)
000E AD;             PLO MA
000F 96;             GHI LINK
0010 BD;             PHI MA
0011 16;             INC LINK
0012 16;             INC LINK
0013 3019;           BR SD
0015 ;               ..
0015 ;               .. 16-BIT SUBTRACT AC FROM OPERAND
0015 ;               .. ****AC=OPRN-AC
0015 ;               .. ************ (TO CALL, WRITE) ***********
0015 ;               .. ****CALL SDOP ; ,A(OPRN)
0015 ;             ..
0015 46;         SDOP: LDA LINK .. FETCH OPERAND ADDRESS
0016 BD;             PHI MA .. TO MA REGISTER
0017 46;             LDA LINK
0018 AD;             PLO MA .. FALL INTO SD
0019 ;               .. 16-BIT SUBTRACT AC FROM OPERAND
0019 ;               .. CALL HERE IF OPERAND
0019 ;               .. ADDRESS IN REGISTER MA
0019 ;               .. ****AC=M(R(MA))-AC
0019 ;               .. ************ (TO CALL, WRITE) ***********
0019 ;               .. ****CALL SD
0019 ;               ..
0019 ED;         SD:   SEX MA ..  SET X PTR TO MA
001A 9F;             GHI AC  ..CHECK SIGN BIT OF AC
001B F3;             XOR  ..AND OPERAND @MA
001C 52;             STR SP   ..AND STORE ON STACK
001D 1D;             INC MA  ..POINT TO LOW 8
001E 8F;             GLO AC .. FETCH AC LOW 8
001F F5;             SD .. SUBTRACT FROM LOW 8 IN MEMORY
0020 AF;             PLO AC .. PUT IT BACK
0021 2D;             DEC MA .. NOW HIGH 8
0022 9F;             GHI AC
0023 75;         SDNB: SDB .. SUBTRACT HIGH 8
0024 BF;             PHI AC .. PUT IT BACK
0025 02;             LDN SP   ..LOAD STORED COMPARING BIT OF OPERANDS
0026 FE;             SHL  ..CHECK STORED SIGN COMPARISION BIT
0027 3B2C;           BNF SDFF  ..IF OPERAND'S SIGNS ARE SAME
0029 9F;             GHI AC  ..NO OVERFLOW POSSIBLE
002A F3;             XOR  ..OTHER WISE CHECK RESULT
002B FE;             SHL  ..SET DF=0  IF OK
002C D5;         SDFF:    SEP RETN   ..RETURN
002D ;               ..
002D ;               .. 16-BIT SUBTRACT FROM AC (ADDRESS IN CALL)
002D ;               .. ****AC=AC-OPRN
002D ;               .. ************ (TO CALL, WRITE) ***********
002D ;               .. ****CALL SMOP ; ,A(OPRN)
002D ;               ..
002D 46;         SMOP: LDA LINK
002E BD;             PHI MA
002F 46;             LDA LINK
0030 AD;             PLO MA
0031 ;               .. 16-BIT SUBTRACT FROM AC (ADDRESS IN MA)
```

```
0031 ;               .. CALL HERE IF OPERAND
0031 ;               .. ADDRESS IN REGISTER MA
0031 ;               .. ****AC=AC-M(R(MA))
0031 ;               .. ************ (TO CALL, WRITE) ***********
0031 ;               .. ****CALL SM
0031 ;               ..
0031 ED;         SM:   SEX MA  ..SET X PTR
0032 9F;             GHI AC  ..GET SIGN OF AC AND
0033 F6;             SHR  ..AND STORE IN 7TH BIT OF CR
0034 52;             STR SP  ..BUT PUT IN (SP) FIRST
0035 9F;             GHI AC  ..AND SEE IF OPERANDS SIGNS ARE THE SAME
0036 F3;             XOR
0037 FA80;           ANI #80  ..TAKE OUT COMPARING SIGN BIT
0039 E2;             SEX SP  ..NOW STORE THAT BIT IN 8TH OF CR
003A F4;             ADD  ..BY ADDING TO IT
003B 52;             STR SP  ..AND STORE THESE TWO BITS ON STACK
003C 1D;             INC MA  ..POINT TO LOW 8
003D ED;             SEX MA  ..REMEMBER TO SET X TO OPERANDS
003E 8F;             GLO AC .. FETCH AC LOW 8
003F F7;             SM .. SUBTRACT MEMORY FROM IT
0040 AF;             PLO AC .. PUT IT BACK.
0041 2D;             DEC MA .. NOW HIGH 8
0042 9F;             GHI AC
0043 77;         SMNB: SMB .. HIGH 8 SUBTRACT, NO BORROW ACROSS.
0044 BF;             PHI AC .. PUT HIGH 8 BACK
0045 E2;             SEX SP  ..NOW CHECK IF  UNDERFLOWED
0046 F0;             LDX  ..LOAD THE STORED TWO BITS
0047 FE;            SHL  ..AND TAKE OUT THE COMPARING SIGN BIT
0048 3B4E;           BNF SMRT ..THE SAME, UNDERFLOW NOT POSSIBLE
004A 52;             STR SP  ..PUT IT BACK TO STACK
004B 9F;             GHI AC   ..OTHERWISE HAVE TO COMPARE SIGN OF RESULT
004C F3;             XOR  ..SIGN BIT OF AC WAS STORED ON STACK
004D FE;             SHL   ..TAKE OUT THAT 7TH BIT
004E D5;         SMRT:  SEP RETN  ..DF=0 IF SUBTRACTION OK
004F ;               ..
004F ;               ..
004F ;               ..16X16 BIT SIGNED MULTIPLY(2'S COMPLEMENT)
004F ;               ..****AC=AC*OPRN
004F ;               ..************(TO CALL, WRITE)***********
004F ;               ..*****CALL MPYOP ; ,A(OPRN)
004F ;               ..
004F 46;         MPYOP:  LDA LINK  ..FETCH MULTIPLICAND ADDRS
0050 BD;             PHI MA  ..INTO REGISTER A
0051 46;             LDA LINK
0052 AD;             PLO MA  ..FALL INTO MPY
0053 ;               .. 16X16 BIT SIGNED MULTIPLY (2'S COMPLEMENT)
0053 ;               .. CALL HERE IF OPERAND ADDRESS
0053 ;               .. IN REGISTER MA
0053 ;               .. ****AC=AC*M(R(MA))
0053 ;               .. ************ (TO CALL, WRITE) ***********
0053 ;               .. ****CALL MPY
0053 ;               ..
0053 ED;          MPY: SEX MA .. SET X NOW
0054 9F;             GHI AC  ..CHECK IS THE SIGN OF MULTICAND
0055 F3;             XOR  ..THE SAME AS THE SIGN OF MULTIPR
0056 FA80;           ANI #80  ..AND STORE THAT BIT
0058 BC;             PHI CR  ..INTO CR.1
0059 F810;           LDI #10 .. SET COUNTER TO 16
```

```
005B AC;                    PLO CR
005C F800;                  LDI #00 .. INITIALIZE MQ TO 0
005E BF;                    PHI MQ ... TO HOLD PRODUCT.
005F AF;                    PLO MQ
0060 2C;               MPL:    DEC CR  ..IF NOT, DECREMENT IT.
0061 9E;                    GHI AC .. SHIFT AC (=MULTIPLIER) RIGHT
0062 F6;                    SHR
0063 BE;                    PHI AC
0064 8E;                    GLO AC
0065 76;                    SHRC .. SHIFT 0 ACROSS BYTES ..
0066 AE;                    PLO AC
0067 9F;                    GHI MQ
0068 3B76;                  BNF MPB  ..IF NO BIT OUT, DON'T ADD.
006A 1D;                    INC MA .. POINT TO LOW 8 OF MULTIPLICAND
006B 8C;                    GLO CR .. IF NOT LAST ITERATION,
006C 3A79;                  BNZ MPA .. GO ADD.
006E 8F;                    GLO MQ  ..LOAD MQ.0 INTO D
006F F7;                    SM  ..MQ.0-(MA.0)
0070 AF;                    PLO MQ  ..AND STORE BACK INTO MQ.0
0071 2D;                    DEC MA  ..NOW DO THE HIGH BYTE
0072 9F;                    GHI MQ   ..
0073 77;                    SMB      ..REMEMBER THE BORROW BIT
0074 BF;                    PHI MQ  ..AND PUT BACK INTO MQ.1
0075 9C;                    GHI CR  ..NOW ARE THE SIGNS OF OPERANDS THE SAME
0076 FE;               MPB:   SHL  ..TEST FOR SIGN BIT
0077 3083;                  BR MPS+#03  ..IF NEGATIVE, SIGN EXTEND
0079 8F;               MPA:     GLO MQ  ..DO MQ+(MA)
007A F4;                    ADD   ..MQ.0+(MA.0)
007B AF;                    PLO MQ  ..AND PUT BACK INTO MQ.0
007C 2D;                    DEC MA  ..SAME FOR HIGH BYTE
007D 9F;                    GHI MQ  ..
007E 74;                    ADC   ..ADD WITH CARRY
007F BF;                    PHI MQ  ..
0080 CF;               MPS:   LSDF
0081 F0;                    LDX
0082 FE;                    SHL..IF OPERAND IS NEGATIVE,THEN
0083 9F;                    GHI MQ  ..PUT MQ.1 INTO D
0084 76;                    SHRC .. SHIFT IN CARRY
0085 BF;                    PHI MQ
0086 8F;                    GLO MQ .. CONTINUE TO LOW 8 OF MQ
0087 76;                    SHRC
0088 AF;                    PLO MQ
0089 3B8F;                  BNF MT .. IF NO CARRY OUT, ITERATE.
008B 9E;                    GHI AC .. ADD CARRY OUT INTO AC MSB.
008C F980;                  ORI #80
008E BE;                    PHI AC
008F 8C;               MT:    GLO CR  ..CHECK COUNTER
0090 3A60;                  BNZ MPL   ..IF COUNTER IS NOT 0,GO BACK FOR MORE
0092 9E;                MPX: GHI AC .. FINISHED;
0093 FE;                    SHL .. CHECK FOR PRODUCT > 15 BITS.
0094 8F;                    GLO MQ
0095 C7;                    LSNF  ..THAT'S HIGH 17 BITS
0096 FBFF;                  XRI #FF .. ALL 00 OR FF.
0098 3A9E;                  BNZ *+#06 .. NAW.
009A 9F;                    GHI MQ
009B C7;                    LSNF
009C FBFF;                  XRI #FF
009E FCFF;                  ADI #FF .. SET DF IF PRODUCT > 15 BITS
```

```
00A0 D5;                    SEP RETN  .. RETURN
00A1 ;                      .. 32/16 BIT SIGNED DIVIDE (2'S COMPLEMENT)
00A1 ;                      .. AC=MQ,AC/OPRN
00A1 ;                      .. QUOTIENT IN AC , REMAINDER IN MQ
00A1 ;                      .. ************ (TO CALL, WRITE) ***********
00A1 ;                      .. CALL DIVOP ; ,A(OPRN)
00A1 ;                      ..
00A1 46;               DIVOP: LDA LINK .. FETCH OPERAND ADDRESS
00A2 BD;                    PHI MA .. TO REGISTER MA
00A3 46;                    LDA LINK
00A4 AD;                    PLO MA .. FALL INTO DIV0/DIV/DIVQ.
00A5 ;                      .. 32/16 BIT SIGNED DIVIDE (2'S COMPLEMENT)
00A5 ;                      ... OPTION #1: CLEAR MQ AND CHECK FOR ZERO D
00A5 ;                      .. ****AC=MQ,AC/M(R(MA))
00A5 ;                      .. ************ (TO CALL, WRITE) ***********
00A5 ;                      .. ****CALL DIV0
00A5 ED;               DIV0: SEX MA .. SET X TO POINT TO DIVISOR (0)
00A6 9E;                    GHI AC .. LOOK AT AC SIGN (0)
00A7 FE;                    SHL .. COPY IT TO DF (0)
00A8 F800;                  LDI #00 .. EXTEND #00 IF POSITIVE, (0)
00AA C7;                    LSNF .. (0)
00AB F8FF;                  LDI #FF .. #FF IF NEGATIVE. (0)
00AD BF;                    PHI MQ .. GIVING +0 OR -0 IN MQ (0)
00AE AF;                    PLO MQ .. (0)
00AF 4D;                    LDA MA .. CHECK FOR ZERO DIVISOR (0)
00B0 F1;                    OR .. (0)
00B1 2D;                    DEC MA .. (DON'T FORGET TO FIX POINTER)(0)
00B2 FD00;                  SDI #00 .. IF ZERO, CALL IT DIVIDE CHECK(0)
00B4 3BB7;                  BNF DIV  ..GO ON IF NO DIVIDE CHECK ERR
00B6 D5;                    SEP RETN ..AND RETURN WITH DF=1 (0)
00B7 ;                      ... OPTION #2: PERMIT 32-BIT DIVIDEND; (/)
00B7 ;                      ..
00B7 ;                      ..MAKE SURE QUOTIENT DOES NOT EXCEED 16 BITS(/)
00B7 ;                      .. ****** (TO CALL, WRITE) ******(/)
00B7 ;                      .. ***** CALL DIV *****(/)
00B7 9F;               DIV:  GHI MQ  ..SAVE PARTIAL DIVIDEND(/)
00B8 BC;                    PHI CR  ..INTO CR.1(/)
00B9 8F;                    GLO MQ   ..(/)
00BA AC;                    PLO CR  ..AND CR.0(/)
00BB FE;                    SHL   ..(/)
00BC AF;                    PLO MQ  ..(/)
00BD 9F;                    GHI MQ  ..DO THE SAME FOR HIGH BYTE(/)
00BE 7E;                    SHLC   ..REMEMBER CARRY(/)
00BF BF;                    PHI MQ  ..(/)
00C0 9E;                    GHI AC  ..ALSO SHIFT IN AC HIGH(/)
00C1 FE;                    SHL   ..(/)
00C2 33CD;                  BDF D2-#01  ..(/)
00C4 8C;                     GLO CR ..SEE IF MQ.0 =0(/)
00C5 3ACE;                  BNZ D2   ..IF NOT GO THROUGH CHECKING STEPS(/)
00C7 9C;                    GHI CR  ..SEE IF MQ.1 IS #40(/)
00C8 FB40;                  XRI #40  ..WHICH SHOULD RESULT 0 IF TRUE(/)
00CA 32D7;                  BZ D4    ..IF TRUE,SKIP NORMAL CHECKING(/)
00CC 38;                    ,#38
00CD 1F;                    INC MQ  ..IF 1, SHIFT INTO MQ(/)
00CE 9C;               D2:     GHI CR   ..CHECK IF HIGH 2 BITS OF MQ ARE THE SAME(/)
00CF FAC0;                  ANI #C0   ..TAKE OUT 2 HIGH BITS(/)
00D1 FD00;                  SDI #00  ..SEE IF THEY ARE THE SAME(/)
00D3 FE;                    SHL  ..SHIFT OUT COMPARISON (/)
00D4 C30156;                LBDF DVXX  ..SET DF AND RETURN(/)
```

```
00D7 9C;          D4:   GHI CR   ..LOOK AT THE SIGNS OF DIVND AND DIVSR(/)
00D8 ED;                SEX MA   ..POINT X TO DIVISOR(/)
00D9 F3;                XOR   ..(/)
00DA FE;                SHL   ..SET DF IF THE SAME(/)
00DB 1D;                INC MA   ..MQ.0-(MA)(/)
00DC 8E;                GLO MQ   ..MQ.0 TO D(/)
00DD 33E6;              BDF DVA   ..IF FLAG ,MQ.0+(MA)(/)
00DF F7;                SM    ..MQ.0-(MA.0)(/)
00E0 AE;                PLO MQ    ..AND STORE BACK INTO MQ.0(/)
00E1 2D;                DEC MA   ..DO THE SAME FOR HIGH BYTE(/)
00E2 9E;                GHI MQ   ..LOAD MQ.1 INTO D(/)
00E3 77;                SMB   ..MQ.1-(MA.1) WITH CARRY(/)
00E4 30EB;              BR DVB    ..SKIP OVER ADD STEP(/)
00E6 F4;          DVA:   ADD   ..MQ+(MA)(/)
00E7 AE;                PLO MQ   ..STORE BACK TO MQ.0(/)
00E8 2D;                DEC MA ..SAME FOR HIGH BYTE(/)
00E9 9E;                GHI MQ   ..MQ.1+(MA.1)(/)
00EA 74;                ADC   ..ADD MQ.1 AND (MA.1) WITH CARRY(/)
00EB BE;          DVB:   PHI MQ   ..AND STORE BACK INTO MQ.1(/)
00EC 9C;                GHI CR   ..LOOK AT THE SIGN OF DIVIND(/)
00ED FE;                SHL   ..SET DF IF DIVIND IS NEGATIVE
00EE CB00FA;            LBNF DV2   ..IF NOT, DON'T COMPLEMENT DIFF(/)
00F1 9E;                GHI MQ   ..COMPLEMENT MQ(/)
00F2 FBFF;              XRI #FF   ..BY XOR TO #FF(/)
00F4 BE;                PHI MQ    ..(/)
00F5 8E;                GLO MQ   ..(DO THE SAME FOR LOW BYTE)(/)
00F6 FBFF;              XRI #FF   ..(/)
00F8 AE;                PLO MQ   ..(/)
00F9 1E;                INC MQ   ..REMEMBER TO ADD 1 (/)
00FA 9E;          DV2:       GHI MQ   ..NOW LOOK AT THE DIFF OF MQ&(MA)(/)
00FB C20103;            LBZ D10   ..IF 0,CHECK MQ=0(/)
00FE FE;                SHL   ..CHECK IF MQ IS NEGATIVE(/)
00FF 335B;              BDF  DDQ   ..IF YES, NO PROBLEM(/)
0101 3054;              BR D9   ..IF NOT 0 NOR NEG,DIVND IS TOO LARGE (/)
0103 8E;          D10:   GLO MQ   ..HERE WE CHECK DIFF=0 CASES(/)
0104 FCFE;              ADI #FE   ..IF MQ.0 IS NOT EITHER 0 OR 1(/)
0106 3356;              BDF DVXX    ..THEN DIVND IS STILL TOO LARGE(/)
0108 FBFF;              XRI #FF   ..RESULT 0 IF MQ.0 WAS 1(/)
010A 3A39;              BNZ DVH   ..IF NOT, MUST BE 0, GO TO DVH(/)
010C 9C;                GHI CR    ..SEE IF DIVND IS NEGATIVE(/)
010D FA80;              ANI #80    ..RESULT 80 IF YES(/)
010F 3254;              BZ   D9   ..IF DIVND POSITIVE, IT CANNOT DIVIDE(/)
0111 F4;                ADD   ..SEE IF DIVISOR IS POSITIVE(/)
0112 3356;              BDF DVXX    ..IF NEGATIVE, DIVND CANNOT DIVIDE(/)
0114 1D;                INC MA   ..AC+(MA) TO MQ(/)
0115 8F;                GLO AC    ..DO AC LOW FIRST(/)
0116 F4;                ADD   ..(/)
0117 AE;                PLO MQ   ..STORE BACK TO LOW MQ(/)
0118 2D;                DEC MA   ..DO THE SAME FOR HIGH BYTE(/)
0119 9F;                GHI AC   ..(/)
011A 74;                ADC   ..ADD WITH CARRY(/)
011B BE;                PHI MQ   ..(/)
011C 1D;                INC MA ..LEAVE MA POINTING TO LOW DIVISOR(/)
011D F0;                LDX   ..NOW HECK LOW BIT (MA) IS 0 OR 1(/)
011E F6;                SHR   ..SHIFT THAT LOW BIT OUT(/)
```

```
011F 2D;                DEC MA   .. REMEMBER TO RESET MA(/)
0120 9E;                GHI MQ   ..READY TO ADD #80(/)
0121 C7;                LSNF    ..IF LOW BIT OF AC IS 1(/)
0122 FC80;              ADI #80    ..TO MQ.1(/)
0124 BE;                PHI MQ   ..STORE STATUS(/)
0125 FE;                SHL   ..SEE IF MQ >0(/)
0126 9C;                GHI CR   ..SEE IF DIVND POSTIVE(/)
0127 FA80;              ANI #80   ..BY TAKE OUT SIGN(/)
0129 3A2F;              BNZ *+#06   ..(/)
012B 3B54;              BNF D9   ..EXIT WITH DF=1(/)
012D 305B;              BR DDQ   ..OK(/)
012F 3356;              BDF DVXX   ..IF NEGATIVE, THEN OUT(/)
0131 9E;                GHI MQ   ..SEE IF MQ=0?(/)
0132 3A5B;              BNZ DDQ   ..IF NOT, THEN NO PROBLEM(/)
0134 8E;                GLO MQ   ..MAKE SURE MQ.0 IS 0 TOO(/)
0135 3254;              BZ D9   ..IF YES, THEN RETURN WITH DF=1(/)
0137 305B;              BR   DDQ   ..OR ELSE GO TO DIVIDE(/)
0139 9C;          DVH:       GHI CR   ..SEE IF THE OPERANDS SIGNS DIFF(/)
013A F3;                XOR   ..BY COMPARING SIGN BITS(/)
013B FE;                SHL    ..AND TAKE OUT THAT BIT(/)
013C 3314;               BDF D10+#11   ..IF SINGS DIFF, IT'S OK(/)
013E 9C;                GHI CR   ..OTHERWISE TEST SIGN OF DIVND(/)
013F FE;                SHL   ..IF POSITIVE, RETURN WITH DF=1(/)
0140 3B54;              BNF D9   ..RETURN WITH DF=1(/)
0142 1D;                INC MA   ..SEE IF LOW BIT OF  AC IS 0 OR 1(/)
0143 F0;                LDX   ..LOAD THAT IN D(/)
0144 2D;                DEC MA   ..REMEMBER TO RESET MA(/)
0145 F6;                SHR    ..SHIFT THAT BIT OUT(/)
0146 8F;                GLO AC   ..IF AC IS NOT 0 ,NOT PROBLEM(/)
0147 3A5B;              BNZ DDQ   ..GO TO DIVIDE(/)
0149 9F;                GHI AC   ..READY TO CHECK #80 IF LOW BIT AC IS 1(/)
014A 3350;              BDF *+#06   ..(/)
014C 3254;              BZ  D9   ..NO GOOD, RETURN WITH DF=1(/)
014E 305B;              BR DDQ   ..ANY THING ELSE IS OK(/)
0150 FB80;              XRI #80   ..IF AC.1 IS'NT #80(/)
0152 3A5B;              BNZ DDQ   ..IT'S OK(/)
0154 FF00;        D9;   SMI #00   ..DF IS SET TO 1(/)
0156 9C;          DVXX:     GHI CR   ..PUT ORIGINAL DIVND(/)
0157 BE;                PHI MQ   ..INTO MQ(/)
0158 8C;                GLO CR ..(/)
0159 AE;                PLO MQ    ..(/)
015A D5;                SEP RETN    ..AND RETURN WITH DF=1(/)
015B 9C;          DDQ:   GHI CR   ...PUT ORIGINAL DIVND(/)
015C BE;                PHI MQ   ..BACK INTO MQ(/)
015D 8C;                GLO CR ..(/)
015E AE;              PLO MQ   ..(/)
015F ;                  ... OPTION #3: ASSUME BENIGN PROGRAM; NOCHECK
015F ;                  .. ************ (TO CALL, WRITE) **********
015F ;                  .. ****CALL DIVQ
015F 9E;          DIVQ:     GHI MQ   ..LOOK AT DIVIDEND SIGN
0160 FE;                SHL .. IF POSITIVE,
0161 F890;              LDI #90 .. PLAN TO BEGIN WITH SUBTRACT,
0163 C7;                LSNF .. (ALSO SAVE SIGN OF DIVIDEND)
0164 F850;              LDI #50 .. OTHER WISE BEGIN WITH ADD.
0166 AC;                PLO CR .. SET ITERATION COUNT IN CR.0
0167 8E;          DVL:  GLO MQ   ..SHIFT MQ LEFT 1 BIT
0168 FE;                SHL   ..SHIFT LEFT MQ.0
0169 AE;                PLO MQ   ..
016A 9E;                GHI MQ   ..SAME FOR HIGH BYTE
```

```
016B 7E;                SHLC   .. SHIFT LEFT WITH CARRY
016C BE;                PHI MQ ..
016D 8F;                GLO AC   ..SHIFT AC LEFT 1 BIT
016E FE;                SHL   ..SHIFT AC.0
016F AF;                PLO AC   ..
0170 9F;                GHI AC   ..
0171 7E;                SHLC    ..SHIFT WITH CARRY
0172 BF;                PHI AC
0173 3B76;              BNF *+#03 .. BIT SHIFTED OUT OF AC.1,
0175 1E;                INC MQ .. GOES INTO MQ.0
0176 ED;                SEX MA ..
0177 8C;                GLO CR .. NOW, WAS THAT ADD, OR SUBTRACT?
0178 F3;                XOR .. IT DEPENDS ON SAVED FLAG,
0179 FE;                SHL   ..AND SIGN OF DIVISOR
017A 1D;                INC MA   ..MQ.0-(MA.0)
017B 8E;                GLO MQ   ..
017C 3B85;              BNF DSA   ..IF NO FLAG,MQ.0+(MA.0)
017E F7;                SM   ..
017F AE;                PLO MQ   ..
0180 2D;                DEC MA   ..FIX X PTR
0181 9E;                GHI MQ   ..DO THE SAME FOR HIGH BYTE
0182 77;                SMB   ..REMEMBER THAT BORROW BIT
0183 308A;              BR DSM   ..SKIP OVER ADD STEPS
0185 F4;           DSA:  ADD   ..MQ+(MA)
0186 AE;                PLO MQ   ..
0187 2D;                DEC MA ..DO THE SAME FOR HIGH BYTE
0188 9E;                GHI MQ   ..
0189 74;                ADC   ..ADD WITH CARRY
018A BE;           DSM:    PHI MQ   ..STORE BACK TO MQ.1
018B 2C;                DEC CR .. COUNT DOWN ITERATION COUNTER
018C 8C;                GLO CR
018D FA7F;              ANI #7F
018F 3B94;              BNF *+#05 .. TEST CARRY OUT OF ADD/SUBTRACT
0191 1F;                INC AC .. IF 1, SHIFT INTO QUOTIENT,
0192 F980;              ORI #80 .. AND FLAG NEXT OP AS SUBTRACT.
0194 AC;                PLO CR .. OTHERWISE IT'S ADD.
0195 FA3F;              ANI #3F .. LOOK AT COUNTER:
0197 3A67;              BNZ DVL .. IF NOT 0, LOOP BACK;
0199 33B0;              BDF DVR .. AT END, CHECK REMAINDER ADJUST.
019B 38;                ,#38
019C 1F;           DVC: INC AC .. (FINAL DIVIDE STEP)
019D F3;                XOR .. BE SURE TO GET POLARITY
019E FE;                SHL .. OF ADJUSTMENT RIGHT...
019F 1D;               INC MA .. YES, ADD DIVISOR BACK ON,
01A0 8E;                GLO MQ .. TO CORRECT FOR FINAL SUBTRACT,
01A1 33AA;              BDF DVM .. (ADDING NEGATIVE IS SUBT.)
01A3 F4;                ADD .. WHICH SHOULDN'T HAVE.
01A4 AE;                PLO MQ
01A5 2D;                DEC MA
01A6 9E;                GHI MQ
01A7 74;                ADC   ..ADD WITH CARRY
01A8 30AF;              BR DVR-#01
01AA F7;           DVM: SM .. SAME THING,
01AB AE;                PLO MQ .. EXCEPT, FOR NEGATIVE DIVISOR.
01AC 2D;                DEC MA
01AD 9E;                GHI MQ
01AE 77;                SMB
01AF BE;                PHI MQ
01B0 9E;           DVR: GHI MQ .. IF REMAINDER IS NOT ZERO,
01B1 3AB6;              BNZ *+#05
```

```
01B3 8E;                GLO MQ
01B4 32C1;              BZ DVN .. BUT IT IS; NO PROBLEM.
01B6 8C;                GLO CR .. IF NOT ZERO,
01B7 FE;                SHL .. IT SHOULD BE SAME SIGN
01B8 FE;                SHL .. AS ORIGINAL DIVIDEND.
01B9 9E;                GHI MQ
01BA CF;                LSDF
01BB FB80;              XRI #80
01BD FC80;              ADI #80 .. IF NOT, WE NEED
01BF 3B9C;              BNF DVC .. ONE MORE DIVIDE ITERATION.
01C1 F0;           DVN: LDX .. FINALLY, IF DIVISOR NEGATIVE,
01C2 FE;                SHL
01C3 3BD0;              BNF DVX   ..(IT'S NOT; WE ARE DONE)
01C5 8F;                GLO AC .. COMPLEMENT QUOTIENT,
01C6 FBFF;              XRI #FF .. BY INVERTING IT,
01C8 AF;                PLO AC
01C9 9F;                GHI AC
01CA FBFF;              XRI #FF
01CC BF;                PHI AC
01CD 1F;                INC AC .. THEN INCREMENTING
01CE FC00;              ADI #00 .. ALSO CLEAR DF.
01D0 D5;           DVX: SEP RETN .. DF=0 IF DIVIDE SUCCESSFUL.
01D1 ;                  ..
01D1 ;                  .. 16-BIT ADD TO AC, OPERAND ADDRESS IN CALL
01D1 ;                  .. ****AC=AC+OPRN .. OPRN=2 BYTE OPERAND
01D1 ;                  .. ************ (TO CALL, WRITE) ***********
01D1 ;                  .. ****CALL ADDOP ; ,A(OPRN)
01D1 ;                  ..
01D1 ;                  ..
01D1 46;           ADDOP: LDA LINK .. FETCH OPERAND ADDRESS
01D2 BD;                PHI MA .. TO REGISTER MA
01D3 46;                LDA LINK
01D4 AD;                PLO MA .. FALL INTO ADD
01D5 ;                  .. 16-BIT ADD TO AC, OPERAND ADDRESS IN REGI
01D5 ;                  .. CALL HERE IF OPRN ADDRESS
01D5 ;                  .. IS IN REGISTER MA
01D5 ;                  .. ****AC=AC+M(R(MA))
01D5 ;                  .. ************ (TO CALL, WRITE) ***********
01D5 ;                  .. ****CALL ADD
01D5 ;                  ..
01D5 ED;           ADD:  SEX MA   ..CHECK SIGN BIT OF AC
01D6 9F;                GHI AC   ..GET THE OPERAND
01D7 F3;                 XOR   ..AND OPERAND @MA
01D8 FB80;              XRI #80   ..RESULT A 1 IF DIFF
01DA 52;                STR SP   ..STORE ON STACK
01DB 1D;                INC MA   ..POINT TO LOW 8 BITS
01DC 8F;                GLO AC .. FETCH AC LOW 8
01DD F4;                ADD .. ADD LOW 8 FROM MEMORY
01DE AF;                PLO AC .. PUT IT BACK
01DF 2D;                DEC MA .. POINT TO HIGH 8 MEMORY LOCATION
01E0 9F;                GHI AC
01E1 74;                ADC   ..ADD HIGH BYTE WITH CARRY
01E2 BF;                PHI AC .. PUT IN AC.
01E3 02;                LDN SP   ..LOAD THE STORED COMPARING SIGN BIT
01E4 FE;                SHL   ..RESET STACK PTR
01E5 3BEA;              BNF ADDRT+#01   ..NOT POSSIBLE
01E7 9F;                GHI AC   ..OTHERWISE SEE IF SUM IS RIGHT
01E8 F3;                XOR   ..BY COMPARING SIGN BITS
01E9 FE;           ADDRT:     SHL    ..SHIFT OUT 0 INTO DF
01EA D5;                SEP RETN   ..RETURN TO MAIN
```

```
01EB ;          ..
01EB ;          .. 16-BIT CONSTANT ADD TO AC.
01EB ;          .. CALL HERE FOR ADD CONSTANT TO AC
01EB ;          .. ****AC=AC+CONSTANT
01EB ;          .. ************ (TO CALL, WRITE) ***********
01EB ;          .. ****CALL ADDCON ; ,CONSTANT
01EB ;          ..
01EB 86;      ADDCON: GLO LINK .. COPY LINK TO MA
01EC AD;          PLO MA
01ED 96;          GHI LINK
01EE BD;          PHI MA
01EF 16;          INC LINK .. INCREMENT PAST DATUM
01F0 16;          INC LINK
01F1 30D5;        BR ADD .. GO ADD.
01F3 ;            ..
01F3 ;            .. 16-BIT ADD FROM TOP OF STACK
01F3 92;      ADDST: GHI SP .. COPY STACK POINTER
01F4 BD;          PHI MA .. TO MA REGISTER
01F5 82;          GLO SP
01F6 AD;          PLO MA
01F7 1D;          INC MA .. ADVANCE TO SUB-TOP (@)
01F8 1D;          INC MA .. (@)
01F9 1D;          INC MA .. (@)
01FA 30D5;        BR ADD .. GO DO IT
01FC ;            ..
01FC ;            ..
01FC ;            ..
01FC ;            .. PUSH AC INTO STACK
01FC ;            .. STACK POINTER = SP
01FC ;            .. ************ (TO CALL, WRITE) ********
01FC ;            .. ****CALL PUSHAC
01FC ;            .. PUSH AC (UNDER TOP OF STACK)
01FC 92;      PUSHAC: GHI SP .. COPY STACK POINTER TO MA
01FD BD;          PHI MA
01FE 82;          GLO SP
01FF AD;          PLO MA
0200 1D;          INC MA .. NOW SLICE OFF TOP 2 BYTES,
0201 1D;          INC MA
0202 E2;          SEX SP
0203 0D;          LDN MA .. TO MAKE A 2-BYTE HOLE.
0204 73;          STXD
0205 2D;          DEC MA
0206 0D;          LDN MA
0207 73;          STXD
0208 9F;          GHI AC .. NOW STUFF AC INTO THE HOLE.
0209 5D;          STR MA
020A 8F;          GLO AC
020B 1D;          INC MA
020C 5D;          STR MA
020D 2D;          DEC MA .. LEAVE MA POINTING TO IT.
020E D5;          SEP RETN .. (AC UNCHANGED)
020F ;            .. PUSH CR, MA, MQ (UNDER TOP OF STACK)
020F 8D;      PUSHCQ: GLO MA .. FIRST PUSH MA ONTO TOP
0210 E2;          SEX SP
0211 73;          STXD
0212 9D;          GHI MA
0213 52;          STR SP
0214 92;          GHI SP .. NOW COPY SP TO MA
0215 BD;          PHI MA
```

```
0216 82;          GLO SP
0217 AD;          PLO MA
0218 22;          DEC SP
0219 1D;          INC MA .. THEN ADJUST IT
021A 1D;          INC MA
021B 1D;          INC MA .. TO POINT INTO OLD TOP
021C 8E;          GLO MQ .. CONTINUE PUSHING, MQ
021D 73;          STXD
021E 9E;          GHI MQ
021F 73;          STXD
0220 0D;          LDN MA
0221 73;          STXD
0222 2D;          DEC MA
0223 0D;          LDN MA
0224 73;          STXD
0225 9C;          GHI CR .. FINALY INSERT CR IN HOLE
0226 5D;          STR MA
0227 1D;          INC MA
0228 8C;          GLO CR
0229 5D;          STR MA
022A D5;          SEP RETN .. (MA IS GARBAGE OUT)
022B ;            ..
022B ;            .. POP STACK INTO MQ, MA, CR (UNDER TOP OF S
022B 12;      POPCQ: INC SP
022C 92;          GHI SP .. COPY STACK POINTER TO MA
022D BD;          PHI MA
022E 82;          GLO SP
022F AD;          PLO MA
0230 1D;          INC MA .. ADJUST POINTER TO CR DATUM
0231 1D;          INC MA
0232 1D;          INC MA
0233 1D;          INC MA
0234 1D;          INC MA
0235 1D;          INC MA
0236 4D;          LDA MA .. FETCH IT
0237 BC;          PHI CR
0238 0D;          LDN MA
0239 AC;          PLO CR
023A 2D;          DEC MA .. COPY TOP OF STACK INTO GAP
023B 42;          LDA SP
023C 5D;          STR MA
023D 1D;          INC MA
023E 42;          LDA SP
023F 5D;          STR MA
0240 42;          LDA SP .. THEN POP MQ
0241 BE;          PHI MQ
0242 42;          LDA SP
0243 AE;          PLO MQ
0244 42;          LDA SP .. FINALLY POP MA
0245 BD;          PHI MA
0246 02;          LDN SP
0247 AD;          PLO MA
0248 D5;          SEP RETN
0249 ;            .. 16-BIT ACCUMULATOR LOAD (ADDRESS IN CALL)
0249 ;            .. ****AC=OPRN
0249 ;            .. ************ (TO CALL, WRITE) ***********
0249 ;            .. ****CALL LOADOP ; ,A(OPRN)
0249 ;            ..
0249 46;      LOADOP: LDA LINK .. FETCH ADDRESS
024A BD;          PHI MA .. TO MA REGISTER
```

```
024B 46;          LDA LINK
024C AD;          PLO MA .. FALL INTO LOAD
024D ;            .. 16-BIT ACCUMULATOR LOAD (ADDRESS IN MA)
024D ;            .. CALL HERE IF OPERAND
024D ;            .. ADDRESS IN REGISTER MA
024D ;            .. ****AC=M(R(MA))
024D ;            .. ************ (TO CALL, WRITE) **********
024D ;            .. ****CALL LOAD
024D ;            ..
024D 4D;      LOAD: LDA MA .. FETCH HIGH 8
024E BF;          PHI AC
024F 4D;          LDA MA .. NOW LOW 8
0250 AF;          PLO AC .. LEAVE MA AT NEXT DOUBLE-BYTE
0251 D5;          SEP RETN .. GEE, THAT WAS QUICK.
0252 ;
0252 ;            ..
0252 ;            .. 16-BIT ACCUMULATOR LOAD FROM CONSTANT IN
0252 ;            .. ****AC=CONSTANT
0252 ;            .. ************ (TO CALL, WRITE) **********
0252 ;            .. ****CALL LODCON ; ,CONSTANT
0252 ;            ..
0252 46;      LODCON: LDA LINK .. FETCH HIGH 8
0253 BF;          PHI AC
0254 46;          LDA LINK .. THEN LOW 8
0255 AF;          PLO AC
0256 D5;          SEP RETN
0257 ;            ..
0257 ;            .. 16-BIT ACCUMULATOR STORE (ADDRESS IN CALL
0257 ;            .. ****OPRN=AC
0257 ;            .. ************ (TO CALL, WRITE) **********
0257 ;            .. ****CALL STOROP ; ,A(OPRN)
0257 ;            ..
0257 46;      STOROP: LDA LINK .. FETCH ADDRESS INTO MA
0258 BD;          PHI MA
0259 46;          LDA LINK
025A AD;          PLO MA .. THEN FALL INTO STORE
025B ;            .. 16-BIT ACCUMULATOR STORE (ADDRESS IN MA)
025B ;            .. ****M(R(MA))=AC
025B ;            .. ************ (TO CALL, WRITE) **********
025B ;            .. ****CALL STORE
025B ;            ..
025B 9F;      STORE: GHI AC .. FIRST HIGH 8
025C 5D;          STR MA
025D 1D;          INC MA .. INCREMENT MA, SINCE STR DOESN'T
025E 8F;          GLO AC .. NOW LOW 8
025F 5D;          STR MA
0260 1D;          INC MA .. LEAVE MA POINTING TO NEXT WORD
0261 D5;          SEP RETN .. QUIT
0262 ;            ..
0262 ;            ..
0262 ;            .. 16-BIT COMPARE, OPERAND ADDRESS IN CALL
0262 ;            .. ****AC-OPRN  (DF SET IF 0 OR +)
0262 ;            .. ************ (TO CALL, WRITE) **********
0262 ;            .. ****CALL COMOP ; ,A(OPRN)
0262 ;            ..
0262 46;      COMPOP: LDA LINK .. FETCH ADDRESS
0263 BD;          PHI MA .. TO MA REGISTER
0264 46;          LDA LINK
0265 AD;          PLO MA
0266 ;            .. 16-BIT COMPARE, OPERAND ADDRESS IN REGIST
```

```
0266 ;            .. CALL HERE IF OPERAND
0266 ;            .. ADDRESS IN REGISTER MA
0266 ;            .. ****AC-M(R(MA))  (DF SET IF 0 OR +)
0266 ;            .. ************ (TO CALL, WRITE) **********
0266 ;            .. ****CALL COMP
0266 ;            ..
0266 ED;      COMP: SEX MA .. COMPARE HIGH 8 FRST
0267 9F;          GHI  AC  ..CHECK IF SIGN OF OPERANDS ARE SAME
0268 F3;          XOR  ..BY LOOKING AT THE HIGHEST SIGN BIT
0269 FA80;        ANI #80  ..RESULT A '1' IF NEGATIVE
026B 3A76;       BNZ CNE  ..IF NOT, THEN GO TO CNE
026D 9F;          GHI AC
026E F7;          SM
026F 3A77;        BNZ CMPX .. NOT EQUAL QUITS
0271 1D;          INC MA .. TRY LOW 8
0272 8F;          GLO AC
0273 F7;          SM
0274 2D;          DEC MA .. LEAVE MA POINTING TO IT
0275 38;          ,#38
0276 F4;     CNE:      ADD  ..SEE IF OPERAND IS NEGATIVE
0277 D5;     CMPX:      SEP RETN  ..DF=1 IF AC>=(MA)
0278 ;            ..
0278 ;            ..
0278 ;            ..
0278 ;            ..
0278 ;           ..TEST 16-BIT ACCUMULATOR SIGN/ZERO ......
0278 16;          INC LINK  ..SKIP OVER NON ZERO RETURN
0279 16;          INC LINK  ..
027A D5;          SEP RETN
027B 9F;      TEST: GHI AC .. FIRST LOOK AT SIGN
027C FE;          SHL  ..SET DF IF MINUS
027D 9F;          GHI AC .. NOW CHECK FOR 0
027E 3A7B;        BNZ TEST-#03 .. NO.
0280 8F;          GLO AC
0281 327A;        BZ TEST-#01 .. GO TAKE ZERO EXIT.
0283 307B;        BR TEST-#03 .. GO TAKE NON-ZERO EXIT.
0285 ;            .. POP STACK (INTO AC) (UNDER TOP)
0285 ;            .. POPS 2 BYTES FROM STACK INTO AC
0285 ;            .. ************ (TO CALL, WRITE) ********
0285 ;            .. ****CALL POPAC
0285 FF00;    POPAC: SMI #00 .. SET DF TO REMEMBER THIS ENTRY
0287 C8;          LSKP
0288 FC00;    POP: ADI #00 .. CLEAR DF FOR THIS ENTRY
028A 12;          INC SP
028B 92;          GHI SP .. COPY SP TO MA
028C BD;          PHI MA
028D 82;          GLO SP
028E AD;          PLO MA
028F 1D;          INC MA .. ADJUST TO SUB-TOP OF STACK
0290 1D;          INC MA
0291 3B98;        BNF *+#07
0293 4D;          LDA MA .. POPPING INTO AC. GET DATUM
0294 BF;          PHI AC
0295 0D;          LDN MA
0296 AF;          PLO AC
0297 2D;          DEC MA
0298 42;          LDA SP .. NOW CLOSE UP THE GAP
0299 5D;          STR MA
029A 1D;          INC MA
029B 02;          LDN SP
```

```
029C 5D;                STR MA
029D 1D;                INC MA
029E D5;                SEP RETN .. MA POINTS TO NEW SUB-TOP
029F ;                      ..
029F ;                      .. DECIMAL TO BINARY CONVERSION
029F ;                      .. ***** AC=DECIMAL NUMBER OF N BYTES
029F ;                      .. DECIMAL NUMBER = SIGN,NN,......,N1,N0
029F ;                      .. SIGN=#0B +
029F ;                      .. SIGN=#0D -
029F ;                      .. N0=10**0 DIGIT
029F ;                      .. N1=10**1 DIGIT
029F ;                      .. ********** (TO CALL, WRITE) *******
029F ;                      .. **** CALL CDB; ,A(NUMBER); ,LENGTH
029F 46;          CDB:   LDA LINK    ..GET NUM ADDRESS
02A0 BA;                PHI AR   ..AND STORE IN RA.
02A1 46;                LDA LINK
02A2 AA;                PLO AR
02A3 46;                LDA LINK  ..GET LENGTH
02A4 FF01;              SMI #01   ..MINUS SIGN BYTE
02A6 AB;                PLO NR   ..AND STORE IN RB.
02A7 F800;              LDI #00  ..CLEAR RF
02A9 AF;                PLO AC
02AA BF;                PHI AC
02AB 0A;                LDN AR    ..CHECK SIGN BYTE
02AC FB0D;              XRI #0D   ..MINUS?
02AE BB;                PHI NR  ..INTO NR.1
02AF 1A;          LOOP:  INC AR   ..GRAB THE FIRST DIGIT
02B0 E2;                SEX SP  ..FIX X PNTR
02B1 0A;                LDN AR  ..CLEAR HIGH BYTE (FOR ASCII)
02B2 FA0F;              ANI #0F
02B4 52;                STR SP  ..PUT IT BACK
02B5 8F;                GLO AC  ..ADD THAT DIGIT TO ACCUM
02B6 F4;                ADD
02B7 AF;                PLO AC
02B8 9F;                GHI AC
02B9 7C00;              ADCI #00  ..REMEMBER CARRY OVER
02BB BF;                PHI AC
02BC 33FF;              BDF OVFLW    ..EXCEEDS ACCUM LIMIT?
02BE 2B;                DEC NR  ..DEC DIGIT COUNTER
02BF 8B;                GLO NR  ..SEE IF IT IS 0?
02C0 32F0;              BZ FINAL  ..YES, THEN DONE
02C2 8F;                GLO AC  ..OTHERWISE MULTIPLY THE ACC BY 10
02C3 FE;                SHL
02C4 73;                STXD
02C5 9F;                GHI AC
02C6 7E;                SHLC   ..CARRY OVER
02C7 73;                STXD
02C8 33FD;              BDF OVFLW-#02  ..EXCEEDED ACC LIMIT
02CA F802;             LDI #02
02CC 52;          MPY3:    STR SP  ..LOOP COUNT
02CD 8F;                GLO AC
02CE FE;                SHL   ..NOW SHIFT AC OVER 4 TIMES MORE
02CF AF;                PLO AC
02D0 9F;                GHI AC   ..SAME FOR AC.1
02D1 7E;                SHLC
02D2 BF;                PHI AC
02D3 33FD;              BDF OVFLW-#02  ..IF OVFLOW, RESET STACK
02D5 02;               LDN SP  ..CHECK LOOP COUNT
02D6 32DC;             BZ MPY10   ..AFTER MULPLY BY 8
02D8 FF01;             SMI #01  ..OR ELSE DEC LOOP COUNT
```

```
02DA 30CC;                BR MPY3  ..BACK FOR MORE ADDITION
02DC 12;          MPY10:    INC SP  ..RECOVER LOOP COUNT

02DD 9F;               GHI AC  ..ADD HIGH BYTE
02DE F4;                ADD   ..ADD 8ACC TO 2ACC
02DF BF;                PHI AC
02E0 33FF;              BDF OVFLW-#01  ..RESULT DF IF OVFLOW
02E2 12;                INC SP  ..RESET STACK PNTR
02E3 8F;                GLO AC   ..SAME FOR AC.0
02E4 F4;                ADD
02E5 AF;                PLO AC
02E6 9F;                GHI AC  ..FOR CARRY OUT
02E7 7C00;              ADCI #00
02E9 BF;                PHI AC
02EA 3BAF;              BNF LOOP  ..IF NOT OVFLW, GO BACK FOR MORE
02EC C8;                LSKP   ..SKIP STACK RESET
02ED 12;                INC SP  ..RESET STACK PTR
02EE 12;                INC SP
02EF D5;          OVFLW:  SEP RETN  ..DF=1
02F0 9F;          FINAL:  GHI AC  ..CHECK IF EXCEED MAX POS NUM LIMT
02F1 FC80;              ADI #80
02F3 3AFC;              BNZ CP  ..IF NOT POSSIBLE, SKIP
02F5 8F;                GLO AC
02F6 3AFC;              BNZ CP  ..IF NOT, GO TO COMP
02F8 9B;                GHI NR  ..SEE IF IT IS POSITIVE
02F9 FCFF;              ADI #FF  ..SET DF ACCORDINGLY
02FB D5;                SEP RETN
02FC 33FB;        CP:   BDF *-#01  ..OVERFLOWED!
02FE 9B;                GHI NR  ..TEST FOR SIGN
02FF 3A09;              BNZ EXIT  ..IF POS, DONE
0301 8F;                GLO AC  ..IF NEG, SUBTRACT FROM 0
0302 FD00;              SDI #00
0304 AF;                PLO AC
0305 9F;                GHI AC
0306 ;

0306 7D00;              SDBI #00
0308 BF;                PHI AC
0309 ;                      ..
0309 ;                      .. BINARY TO DECIMAL CONVERSION
0309 ;                      .. ****DECIMAL NUMBER = AC
0309 ;                      .. DECIMAL NUMBER = SIGN,NN,......,N1,N0
0309 ;                      .. SIGN=#0B +
0309 ;                      .. SIGN=#0D -
0309 ;                      .. N0=10**0 DIGIT
0309 ;                      .. N1=10**1 DIGIT, ETC
0309 ;                      .. ********* (TO CALL, WRITE) ********
0309 ;                      .. ****** CALL CBD; ,A(NUMBER); ,LENGTH
0309 D5;          EXIT:  SEP RETN
030A 46;          CBD:  LDA LINK  ..GET THE ADDRESS
030B BA;                PHI AR  ..AND STORE IN RA
030C 46;                LDA LINK  ..SAME FOR LOW BYTE
030D AA;                PLO AR
030E 46;                LDA LINK  ..GET LENGTH
030F FF01;              SMI #01  ..SUBTRAC FOR SIGN BYTE
0311 AB;                PLO NR   ..STORE IN NR.0
0312 BB;                PHI NR   ..AND NR.1
0313 F80F;              LDI #0F  ..NUM OF ITERATIONS
0315 AD;                PLO MA   ..STORE IN MA.0
0316 9F;                GHI AC   ..TEST FOR SIGN
0317 FE;               SHL
```

```
0318 F80B;          LDI #0B  .. IF DF=0, IT IS POS
031A 3B29;          BNF POS   ..
031C 8F;            GLO AC  ..OTHERWISE CONVERT IT TO POS
031D FD00;          SDI #00
031F AF;            PLO AC
0320 9F;            GHI AC
0321 7D00;          SDBI #00
0323 BF;            PHI AC
0324 F80D;          LDI #0D  ..MINUS SIGN
0326 C8;            LSKP  ..
0327 F800;          LDI #00
0329 5A;       POS:     STR AR  ..PUT IT IN SIGN BYT
032A 8B;            GLO NR   ..CHECK DIGIT COUNTER
032B 3232;          BZ LOOP1-#02  ..GO BACK FOR MORE ITERATION
032D 1A;            INC AR  ..GO TO NEXT DIGIT
032E 2B;            DEC NR  ..DEC DIGIT CNTR
032F 3027;          BR POS-#02   ..GO BACK FOR MORE CLEAR
0331 E2;           SEX SP
0332 9B;            GHI NR  ..RESET DIGIT CNTR
0333 AB;            PLO NR    ..
0334 8F;       LOOP1:   GLO AC  ..SHIFT BIT OF AC OUT
0335 FE;            SHL  ..
0336 AF;            PLO AC
0337 9F;            GHI AC
0338 7E;            SHLC .. SAME FOR AC.1
0339 BF;            PHI AC
033A 0A;            LDN AR
033B 7C00;          ADCI #00   ..ADD TO LOWEST DIGIT
033D 5A;            STR AR
033E 8D;            GLO MA   ..FOR MORE ITERATION?
033F 3A42;         BNZ *+#03  ..CONTINUE IF MORE ITERATION
0341 D5;        END:   SEP RETN
0342 0A;       NEXT:  LDN AR  ..LOAD DIGIT
0343 7E;            SHLC  ..SHIFT LEFT OVER ONCE
0344 5A;            STR AR  ..PUT IT BACK
0345 FF0A;          SMI #0A  ..NEED TO INC NEXT DIGIT?
0347 3B4A;         BNF *+#03  ..SKIP IF NOT>10
0349 5A;            STR AR  ..ELSE UPDATE DIGIT
034A 2A;            DEC AR  ..GO TO NEXT DIGIT
034B 2B;            DEC NR   ..DEC DIGIT COUNT
034C 8B;           GLO NR   ..CHECK IF 0?
034D 3A42;         BNZ  NEXT  ..IF NOT, DO THE SAME FOR NEXT DIGIT
034F 3341;         BDF  END  ..OVERFLOWED
0351 2D;           DEC MA  ..DEC NO OF SHIFTS
0352 9B;           GHI NR  ..RESET ADDRESS PTR
0353 52;            STR SP  ..PUT DIGIT ON STACK
0354 8A;            GLO AR
0355 F4;           ADD
0356 AA;            PLO AR
0357 9A;            GHI AR
0358 7C00;         ADCI #00
035A BA;           PHI AR
035B 3032;          BR LOOP1-#02   ...
035D ;                 END
```

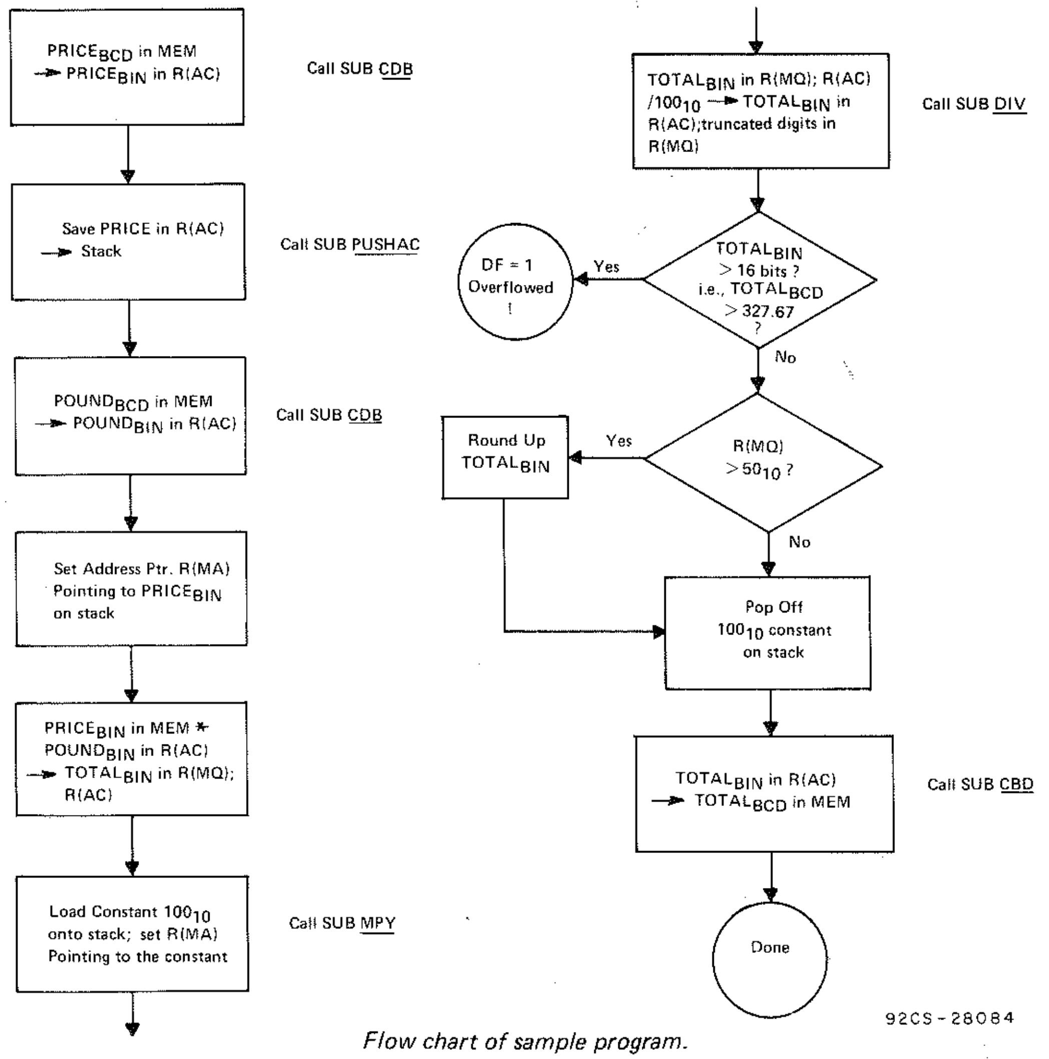
Appendix C - Sample Program

A sample program to demonstrate the use of the Arithmetic Subroutine Package is given in this appendix. This program accepts two input parameters–PRICE and POUND, calculates the PRICE·POUND product (TOTAL), and outputs TOTAL after rounding and truncation.

The input parameters are in BCD form and a maximum of four digits is allowed. A decimal point is also assumed on the left of the second least significant digit, i.e., $00.00 \leqslant$ PRICE, POUND $\leqslant 99.99$.

These parameters are then converted into binary numbers and TOTAL is calculated by multiplying PRICE and POUND in binary. This binary product is divided by 100 to eliminate the last two decimal digits of the eventual BCD TOTAL. If the remainder of the division is greater than fifty, one is added to the 16-bit quotient (for rounding purposes). This latter is then converted back into BCD form.

Note that since the binary-to-BCD routine is designed for 16-bit conversion only, an overflow condition will occur if TOTAL is greater than 327.67. The DF flag sets if overflow occurs.



PRICE_BCD in MEM → PRICE_BIN in R(AC)
Call SUB CDB
Save PRICE in R(AC) → Stack
Call SUB PUSHAC
POUND_BCD in MEM → POUND_BIN in R(AC)
Call SUB CDB
Set Address Ptr. R(MA) Pointing to PRICE_BIN on stack
PRICE_BIN in MEM * POUND_BIN in R(AC) → TOTAL_BIN in R(MQ); R(AC)
Load Constant 100_10 onto stack; set R(MA) Pointing to the constant
Call SUB MPY
TOTAL_BIN in R(MQ); R(AC) /100_10 → TOTAL_BIN in R(AC);truncated digits in R(MQ)
Call SUB DIV
TOTAL_BIN > 16 bits ? i.e., TOTAL_BCD > 327.67 ?
Yes
DF = 1 Overflowed !
No
R(MQ) > 50_10 ?
Yes
Round Up TOTAL_BIN
No
Pop Off 100_10 constant on stack
TOTAL_BIN in R(AC) → TOTAL_BCD in MEM
Call SUB CBD
Done
92CS-28084

Flow chart of sample program.

```
0215  ;                      ...............................................
0215  D4;                    SEP CALL . . DO DECIMAL TO BINARY CONVERSION
0216  076E;                  ,A(CDB)   . . "
0218  027D;                  ,A(PRICE) . . CONVERT PRICE INTO BINARY
021A  05;                    ,#05 . . PRICE IS 5 CHARS LONG
021B  ;                              . . . AC NOW CONTAINS BINARY VALUE
021B  ;                               . . OF PRICE PER POUND
021B  ;                      ...............................................
021B  D4;                    SEP CALL . . PUSH CONTENT OF AC INTO STACK
021C  06E9;                  ,A(PUSHAC) . . "
021E  ;                      ...............................................
021E  D4;                    SEP CALL . . DO DECIMAL TO BINARY CONVERSION
021F  076E;                  ,A(CDB) . . "
0221  0282;                  ,A(LBS)  . . CONVERT QUANTITY (LBS)
0223  ;                                 . . . INTO BINARY
0223  05;                    ,#05 . . QUANTITY IS 5 CHARS LONG
0224  ;                            . . AC NOW CONTAINS BINARY VALUE
0224  ;                            . . OF QUANTITY (LBS)
0224  ;                      ...............................................
0224  82;                    GLO SP . . COPY STACK POINTER TO MA
0225  AD;                    PLO MA . . "
0226  1D;                    INC MA . . POINT TO AC. 1
0227  ;                      ...............................................
0227  D4;                    SEP CALL . . DO THE MULTIPLICATION
0228  0475;                  ,A(MPY) . . "
022A  ;                      ...............................................
022A  ;                      . . . DIVIDE BY 100 TO REMOVE LAST TWO
022A  ;                      . . . DECIMAL DIGITS
022A  12;                    INC  SP      . . MOVE SP TWO BYTES
022B  12;                    INC  SP      . . . BELOW TOP OF STACK, POP PRICE
022C  ;                                   . . . OFF STACK
022C  F864;                  LDI  100     . . LOAD 100 INTO STACK WITH
022E  52;                    STR  SP      . . . SP POINTING TO THE HIGH BYTE
022F  22;                    DEC  SP      . . . "
0230  F800;                  LDI  00      . . . "
0232  52;                    STR  SP      . . . "
0233  82;                    GLO  SP     . . COPY STACK POINTER TO MA
0234  AD;                    PLO  MA     . . POINT TO HIGH BYTE OF 100
0235  22;                    DEC  SP     . . POINT TO FREE SPACE
0236  D4;                    SEP CALL
0237  051E;                  ,A(DIV) . . DIVIDE PRODUCT BY 100
0239  334A;                  BDF LAB3 . . IF OVERFLOW GO TO LAB3
023B  ;                      ...............................................
023B  ;                      . . . CHECK IF REMAINDER IS GREATER
023B  ;                      . . . THAN  50, IF SO, ROUND UP
023B  8E;                    GLO MQ . . MQ CONTAINS THE REMAINDER
023C  FF32;                  SMI  50
023E  3B41;                  BNF  LAB1    . . IF NO ROUND UP
0240  ;                                   . . . GO TO LAB1
0240  1F;                    INC AC   . . IF ROUND UP ADD 1 TO
0241  ;                                . . . THE LEAST SIGNIFICANT DIGIT
0241  ;                      ...............................................
0241  12;          LAB1:     INC SP . . MOVE SP DOWN TWO BYTES
0242  12;                    INC SP . . . BELOW TOP OF STACK
0243  ;                      ...............................................
0243  D4;                    SEP CALL . . DO BINARY TO DECIMAL CONVERSION
0244  0645;                  ,A(CBD)  . . "
0246  0287;                  ,A(TPR)  . . CONVERT TOTAL PRICE INTO DECIMAL
0248  06;                    ,#06 . . TOTAL PRICE IS 6 CHARS LONG
0249  ;                            . . TOTAL PRICE IS STORED IN M(TPR)
0249  ;
```

Partial assembly-language listing of the calculation subroutine.