3890/XP Series

Stacker Control Instructions Reference

SC31-2703-00

3890/XP Series

Stacker Control Instructions Reference

SC31-2703-00

First Edition (March 1990)

Information in this manual is subject to change from time to time. Before using this publication in connection with the operation of IBM systems, consult the *IBM System/370, 30xx, 4300, and 9370 Processors: Bibliography of Industry Systems and Application Programs,* GC20-0370, for the editions that are applicable and current.

IBM does not stock publications at the address given below; requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Information Development, Department 78C, 1001 W. T. Harris Boulevard West, Charlotte, NC 28257, U.S.A. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Special Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights or other legally protectible rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, programs, or services, except those expressly designated by IBM, are the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

Copying and Distributing Softcopy Files

For online versions of this book, we authorize you to:

- Copy, modify, and print the documentation contained on the media, for use within your enterprise, provided you reproduce the copyright notice, all warning statements, and other required statements on each copy or partial copy.
- Transfer the original unaltered copy of the documentation when you transfer the related IBM product (which may be either machines you own, or programs, if the program's license terms permit a transfer). You must, at the same time, destroy all other copies of the documentation.

You are responsible for payment of any taxes, including personal property taxes, resulting from this authorization.

THERE ARE NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Some jurisdictions do not allow the exclusion of implied warranties, so the above exclusion may not apply to you.

Your failure to comply with the terms above terminates this authorization. Upon termination, you must destroy your machine readable documentation.

Trademarks

The following terms, denoted by an asterisk (*) elsewhere in this publication, are trademarks of the IBM Corporation in the United States and/or other countries:

IBM	Operating System/2	OS/2
System/370	Personal System/2	PS/2

About This Manual

The Stacker Control Instructions (SCIs) support all models of the IBM* 3890/XP Series Document Processor, which includes the IBM 3890/XP Document Processor, the IBM 3891/XP Document Processor, and the IBM 3892/XP Document Processor.

This manual describes stacker control programming, including the SCIs, condition codes, and execution-time formulas.

Some of the information in this manual was previously in the following publications:

- *IBM 3890/XP VSE Support Program Reference Manual*, SC31-3887
- *IBM 3890/XP MVS Support Program Reference Manual*, SC31-3886
- *IBM 3890/XP Document Processor Programming Reference*, GC31-3913.

Who Should Read This Manual

This manual is for the system analysts and application and system programmers who install, diagnose, and use MVS Support or VSE Support in medium and large financial institutions. Field Support personnel may also use the information for problem diagnosis.

Terms That You Should Know

This manual uses the term **MVS Support** to mean the 3890/XP MVS Support licensed program and the term **VSE Support** to mean the 3890/XP VSE Support licensed program.

This manual uses the term **3890/XP** or the term **document processor** to mean all models of the 3890/XP series document processor.

This manual uses the term **routing number** to mean the routing-and-transit field on MICR documents.

This manual uses the term **code line data matching** to mean image matching and also image processing.

This manual uses the term **native** to mean any programming language that processes in the IBM Personal System/2* (PS/2*) under Operating System/2* (OS/2*).

How to Use This Manual

This manual assumes both MVS and VSE unless MVS or VSE is specified. “For MVS Only” or “For VSE Only” is followed by a box that contains the information that applies to MVS or VSE.

How This Manual Is Organized

This manual consists of the following three chapters:

- Chapter 1, “Introduction to Stacker Control Instructions (SCIs),” introduces SCI programming, emphasizes the coding with and without MVS Support or VSE Support macros, describes the coding conventions, and explains the difference between mnemonics and machine instructions.
- Chapter 2, “Stacker Control Instructions and Macros,” explains how you use SCIs to make various comparisons and tests on document data to arrive at the pocket-select decision.
- Chapter 3, “Stacker Control Instruction Codes,” lists and explains SCIs, SCI condition codes, and SCI execution-time formulas.

This manual also contains a glossary and an index.

Related Publications

For other information related to SCIs, see the following publications:

- *IBM 3890/XP Series Programming Guide*, GC31-2662
- *IBM 3890/XP MVS Support and 3890/XP VSE Support Program Reference*, SC31-2654, referred to in this manual as *3890/XP MVS and VSE Support Program Reference*
- *IBM 3890/XP Series SPXServ Reference*, GC31-2704
- *IBM 3890/XP Document Processor General Information Manual*, GA34-2012
- *IBM 3890/XP Document Processor Operator’s Guide*, GA34-2013
- *IBM 3890 Document Processor Machine and Programming Description*, GA24-3612
- *IBM 3890/XP Document Processor Programming Reference*, GC31-3913, is available for quick programming information that is related to:
  - SCIs
  - Condition codes
  - Record headers
  - Control unit storage assignments
  - Other pertinent programming information.

Contents

Figures

Chapter 1. Introduction to Stacker Control Instructions (SCIs)

This chapter contains the following main sections:

- Overview of the SCI support
- Coding conventions
- An explanation of the difference between mnemonics and machine instructions
- ENTR macro
- Two- and four-byte addressing.

Overview of the SCI Support

This section discusses the functions that make up SCI programming. With SCIs, you can write and test document processor programs at the host. SCIs access the data areas in the document processor and they let you control the tables in the document processor.

The IBM 3890/XP MVS (Multiple Virtual Storage) Support licensed program and the IBM 3890/XP VSE (Virtual Extended Storage) Support licensed program include the support for the SCIs. This support consists of the following functions:

SCI macros
: The instructions that you can use in your sort program. When run in an XP series document processor, the SCIs control the stacker selection for the documents.

Access method
: Device control macros and modules that let the host application program control the operation of the document processor and supply error recovery and recording.

Test aid
: A way to test the SCI programs without the 3890/XP hardware.

Simulator

For MVS Only

A way to test the application programs without the hardware. Simulator tells test aid to simulate the running of the SCI programs.

Understanding the Macro Descriptions

This section describes the instruction format, mnemonics, and coding conventions.

General Format

Use the following general rules when you write the format for a macro instruction, as shown in Figure 1-1:

Figure 1-1. Macro-Instruction Format Example

Name	Operation	Operand(s)
Symbol or blank	Macro name	None, one, or more (separated by commas) operands

Name field: Can contain any symbolic address, or you can leave it blank.

Operation field: Must contain the exact mnemonic operation code.

Operand field: Parameters must be in the same format as those in the instruction example in Chapter 2, “Stacker Control Instructions and Macros.”

Operands specify the services and the options performed. Use the following rules:

- If the operand that you select is in full capital letters (for example, DEVD = MR,MF = L,SF), code the operand exactly as the instruction example shows in Chapter 2, “Stacker Control Instructions and Macros.”
- If the operand that you select is in lowercase letters, substitute the indicated value, address, or name.
- If the operand that you select is a combination of capital and lowercase letters separated by an equal sign (DDNAME = symbol), code the capital letters and equal sign as the instruction example shows in Chapter 2, “Stacker Control Instructions and Macros”; then substitute the indicated value.
- Code commas and parentheses exactly as the instruction example shows in Chapter 2, “Stacker Control Instructions and Macros” and omit a comma following the last operand.

Mnemonics

This manual lists the mnemonic operation code with a reconstruction of the source code for the SCI macro. The following example of the LPS (Load Work Register from Program Storage) instruction shows this:

```
                          (C1 1l aa aa)
LPS                       (D7 1l aa aa aa aa)
 [Symbol]     LPS         l, label or a
```

In this example, LPS is the mnemonic. The 2-byte machine instruction is (C1 1l aa aa). The 4-byte machine instruction is (D7 1l aa aa aa aa). For an explanation of the 2-byte and the 4-byte machine instructions, see the section about the ENTR macro in the *3890/XP MVS and VSE Support Program Reference*.

You can combine an SCI program and an SCI subroutine that you already assembled separately. For more information, see the section about using SCI2 and SCI4 routines together in the *3890/XP MVS and VSE Support Program Reference*.

Chapter 2, “Stacker Control Instructions and Macros” shows the mnemonic and the machine instruction for each macro.

Coding Conventions

This manual uses the following conventions in the syntax for each instruction:

[ ] Brackets indicate optional operands. You need not code the operand enclosed in the brackets, depending on whether you desire the associated option. If you enclose more than one item in brackets, you can code one or more of the items.

{ } Braces indicate a choice. You must code one of the operands within the braces, depending on which of the associated services you desire.

v Or indicates an OR condition. An underscored option is the default.

... Ellipses indicate that more than one set of operands can be designated in the same macro instruction.

Do not code these symbols; they only indicate how you can write a macro.

Continuation Lines

You can continue the operand field of a macro instruction on one or more additional lines, as follows:

1. Enter a continuation character (not a blank and not part of the operand coding) in column 72 of the line.
2. Continue the operand field on the next line, starting in column 16. All columns to the left of column 16 must be blank.

If you continue an operand field to an additional line, you can code it in one of two ways:

- Code the operand field through column 71, do not use blanks, and continue in column 16 of the next line.
- Truncate the operand field with a comma, where a comma normally falls; use at least one blank before column 71; and then continue in column 16 of the next line. Figure 1-2 shows an example of each method.

Figure 1-2. Continuation Coding

Name	Operation	Operand	Comments
NAME1	LFE	OPERAND1,OPERAND2, OPERAND3,OPERAND4	X
NAME2	LSE	OPERAND1,OPERAND2, OPERAND3, OPERAND4	X X

Chapter 2. Stacker Control Instructions and Macros

The 3890/XP control program performs a series of SCIs to control special features and document routing. The SCIs make various comparisons and tests on document data that affect the pocket-select decision.

You develop the SCI program in one of two ways. You can code the SCIs directly, using DCs (define constants) and the machine format, or you can use SCI macros (supplied by the 3890/XP MVS Support licensed program or the 3890/XP VSE Support licensed program) to produce the object code (machine language). Data management then loads the object code into the 3890/XP by the channel attachment.

Macros produce two-, four-, or six-byte SCIs for performing required functions. Some macros support both two- and four-byte addresses. This chapter shows each macro coded in the standard IBM System/370* format and enclosed in a box, as in the example below. The machine format, where applicable, appears in the space above the box. If a macro supports both 2-byte addresses and 4-byte addresses,
the example shows the machine format for both.

Machine Format (2-byte addresses)
XXXX Machine Format (4-byte addresses)

[Symbol]	Command	Operands

Code the SCI program in the following sequence:

1. SCIs
2. Tables and constants.

Note: All instructions and tables (except the tables for search-binary table) must occupy the first 64K bytes. Only the SBT, LPS, and SPS instructions can gain access to storage areas above 64K bytes. Any other SCI that has a 4-byte address requires the high-order byte of the address set to zero.

Abbreviations and Values

Abbreviation	Value	Explanation
a		Program storage or auxiliary storage address.
c	{0-15}	Specifies one of 16 two-byte binary counters.
d	0-15	Displacement in digits from the units position. A displacement of zero points to the units position; a displacement of one specifies the tens position; and so forth.
f	0-15	Read field. It specifies the field to load into the work register. Zero specifies the header; 1-7 specifies fields of document data; 8-15 specifies fields of information that come from either the user-supplied sort program or the host program when code line data matching is successful.
h	0-11	Specifies one of 12 header-bytes within the header (field 0) of the process buffer.
i		Immediate data can be one or two bytes long, and each macro states the required length. Each macro also states the range of values that can be used.
l	1-16	Length specifies the number of digit positions or bytes to process.
m	0-15	Specifies mask for testing condition codes.
mp	11-66	Specifies the module pocket (MP) code. The MP code is reset to 00 before SCI processing, set by various SCIs, and becomes the destination pocket after the EXIT SCI runs. Each SCI that sets the MP code overrides all previous MP-setting SCIs.
r	{0-3}	Specifies one of four link registers.
s	0-510	Specifies a save area that must be specified as an even value.
label		The label of a constant, a table, or an SCI. This label is resolved into its correct address during assembly.

Terms Used in SCI Instructions

Counter: Sixteen 2-byte counters for storing, adding to, and comparing with a value.

First element: All data to the right of the rightmost SS4 in a field (or the entire field if no SS4 is present).

Halfword: A two-byte group starting on an even-address boundary.

Header: A 12-byte field that defines the type of record and any errors detected by the system unit. You can specify any of the 12 bytes to obtain additional information about the document that was read. When code line data matching, the SCI program can set several bytes for interrogation by the host program, and the host program can set several bytes for interrogation by the SCI program.

Immediate data: One or two bytes of data specified in either decimal or hexadecimal format, depending on the instruction used.

Link register: Four link registers for saving addresses. These are normally for branch-and-link operations.

Process buffer: A variable-length area in auxiliary storage, that contains document data and record header information. Your initialization data controls the length, and your SCI program loads information from this buffer into the work register.

Save area: An area in auxiliary storage for storing data from the work register. The area consists of 512 digit positions, accessible on any two-digit boundary, and is addressed right to left.

Second element: All data to the left of the rightmost SS4 including any other SS4s that might appear. There is no second element if there is no SS4.

Table: Any macro-created table used with SCIs for base conversion, number self-checking, pocket-list comparisons, and data comparisons.

User stats: A storage byte in auxiliary storage that supplies eight-bit switches for control within an SCI program or from one SCI program to another SCI program. This byte can be reset during initialization. This byte can be used as a first-time switch.

Work register: A 32-digit (16-byte) area in 3890-emulated auxiliary storage into which SCIs can load individual fields or selected data from storage. Document data must be loaded into the work register from the process buffer if it is to be processed.

SCI-error notations: After each of the following SCI macro explanations, where applicable, is a list of SCI errors that can result when that macro is coded incorrectly. If you are using one of the 3890/XP support licensed programs (see page 2-1), these coding errors are detected during assembly of the SCI program, and you should correct them before you load the program into the 3890/XP.

List of Machine Instructions and Mnemonics

This section lists the SCI machine instructions by group, then describes each instruction.

Process Buffer/Work Register
- Load Work Register from First Element (LFE)
- Load Work Register from Second Element (LSE)
- Load Work Register from Field (LF)

Operator Communication
- Load Work Register from Program Switches (LSW)
- Set Operator Message from Work Register (SOM)

Work Register/Immediate
- Test Work Register Immediate (TI)
- Insert into Work Register Immediate (II)
- Compare Work Register Immediate (CI)
- Subtract from Work Register Immediate (SI)

Work Register/Save Area
- Store into Save Area from Work Register (STA)
- Insert from Save Area from Work Register (IA)
- Compare Work Register to Save Area (CA)

Load/Store Work Register
- Load Work Register from Auxiliary Storage (LAS)
- Load Work Register from Program Storage (LPS)
- Store Work Register into Auxiliary Storage (SAS)
- Store Work Register into Program Storage (SPS)

Work Register/Storage
- Compare Work Register to Table (CT)
- Compare Table (CTBL)
- Compare and Select (CST)
- Scan Table (CSTBL)
- Search Binary (SBT)
- Search Binary Table (SBTBL)
- Subtract Decimal Data (SS)
- Verify (V)
- Verification Table (VTBL)

Number Self-Checking
- Modulus 10
- Modulus 11
- Convert Base (CV)
- Convert Base Table (CVTBL)

Figure 2-1 (Part 1 of 2). SCI Machine Instructions and Mnemonics

Base-Number Sorting Techniques
- Base-Number Conversion
- Base-Number Conversion with Compression

Header/Immediate
- AND Header Immediate (NHI)
- OR Header Immediate (OHI)
- Compare Header Immediate (CHI)
- Test Header Under Immediate Mask (THI)

Counter/Immediate
- Add Counter Immediate (ACI)
- Load Counter Immediate (LCI)
- Compare Counter Immediate (CCI)

User Stats/Immediate
- AND User Stats Immediate (NSI)
- OR User Stats Immediate (OSI)
- Test User Stats Immediate (TSI)

Branching
- Branch On Condition (BCS)
- Branch and Link (BLS)
- Branch On Register (BUR)
- Jump On Table Index (JX)
- Digit Index Table (JXTBL)
- Call Native (CALLNATV)

Select Pocket/Feature
- Select MP Immediate (SPI)
- Select MP by Counter (SPC)
- Select MP by Table Index (SPTX)
- Pocket Table (STBL)
- Select FC Immediate (SFI)
- Select FC/MP by Header (SPH)

Miscellaneous SCIs
- Disengage (DSG)
- Disengage and Set Operator Message Attention (DSGA)
- Feed from Merge (FM)
- Initialize Feature Data (IFD)
- Extended Code Line Data Match (IMAT)
- Exit - End SCI (EXT)
- Define (DWF)
- Execute (EXC)

Figure 2-1 (Part 2 of 2). SCI Machine Instructions and Mnemonics

ACI (Add Counter Immediate)

ACI ***(AA c0 ii ii)***

[Symbol]	ACI	c, i

The **ACI** (add-counter-immediate) macro permits you to increase the specified counter by the value of immediate data with a decimal range from 0 to 65 535 or a hexadecimal range from X'0000' to X'FFFF'. Treat both the counter and immediate values as unsigned, 16-bit, binary numbers.

A subtraction can be performed by adding the two's complement of a number to the counter.

Resulting Condition Code:

0 Sum zero, no carry
1 Sum not zero, no carry
2 Sum zero, carry
3 Sum not zero, carry

SCI Error: None.

Examples:

Subtract 1 from counter 3.

```
.
.
ACI  3,X'FFFF'
.
```

Add 1 to counter 1.

```
.
.
ACI  1,X'00001'
.
```

Add 1 to counter 8.

```
.
.
ACI  8,1
.
```

Base-Number Sorting Techniques

Sorting is under the control of your user-supplied sort program. The configuration can have up to 36 pockets. This gives you a broad distribution of documents on a single pass through the machine.

For example, if you have 17 pockets available for a fine sort (18 pockets including reject), you could distribute a field value of 0 through 16 to the 17 pockets on a single pass, rather than on the normal two passes (unit's-position sort followed by the ten's-position sort).

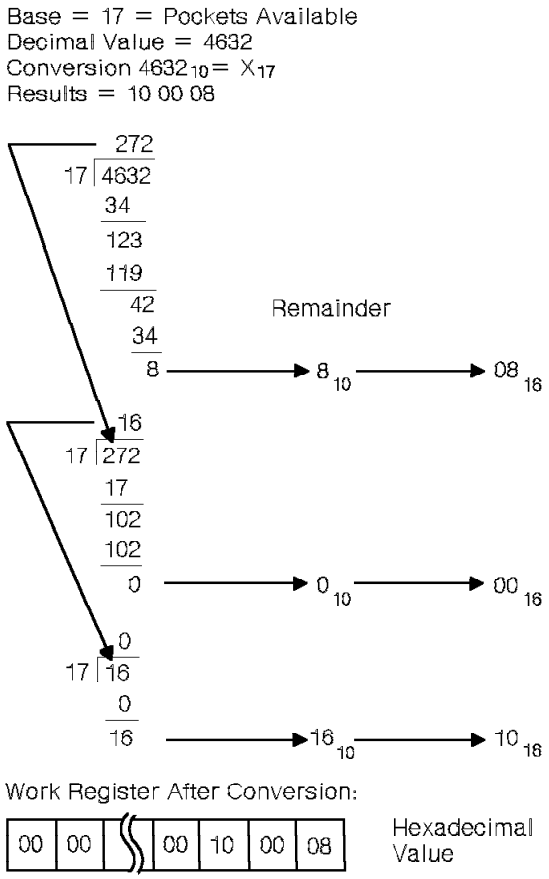
For a 17-pocket sort, convert the field value to its base-17 equivalent and sort the results. SCIs can perform this conversion and set up pocket tables for correct pocket selection.

Base-Number Conversion

Base-number conversion is an automatic function of several SCIs. The following description should give you a better understanding of this function.

To convert to the hexadecimal representation of the new base, divide the decimal value by the base and collect the remainders in reverse sequence.

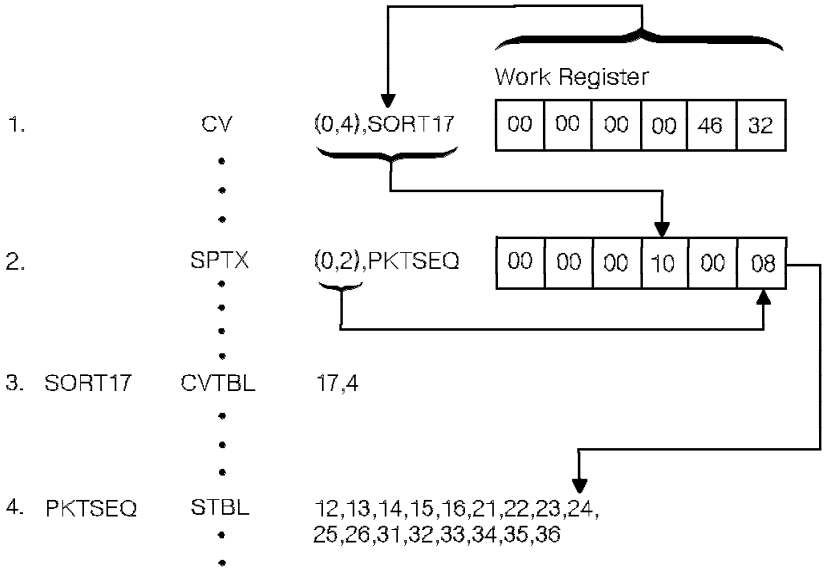
Example:



In this example, a four-position decimal value was decreased to a three-byte, base-17 value. Performing a sort on this converted number would eliminate one complete sorting pass.

Depending on the value to be sorted and the number of pockets available, it is possible to save several passes through the machine. This results in less document handling and less machine time.

Sample Program



1. Converts the low-order four positions of the work register to its base-17 equivalent.
2. The first operand points to the low-order byte of converted work register and uses its contents to index into the pocket-select table at PKTSEQ (line 4). The module pocket code selected is 24.
3. Creates the base-17 table of four positions. The CV macro uses the table at line 1.
4. Creates the pocket-select table, starting with module-1/pocket-2, up to module-3/pocket-6, for a total of 17 consecutive sorting pockets.

Sample Sort

The following sample sort (Figure 2-2 to Figure 2-4) shows how the documents enter at each sorting pass and how they are distributed in the pockets. Each pocket is assigned a hexadecimal value through the pocket table macro, and the sorting is based on base-number conversion.

Each document appears with its decimal value and its base-17 equivalent in parentheses. The hopper feeds from bottom to top; the pockets empty from left to right.

You can see the result of the sort by scanning the pocket distribution for the third pass.

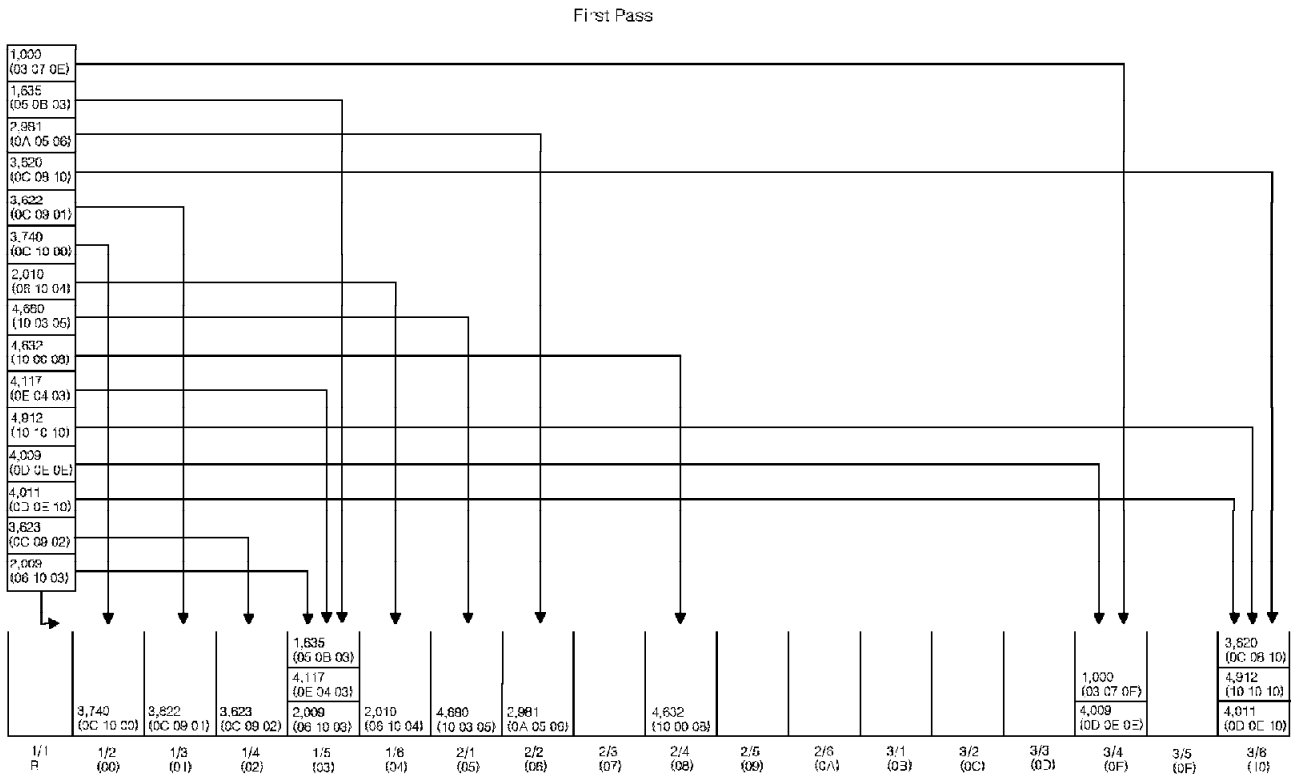


Figure 2-2. Sample Sort—First Pass

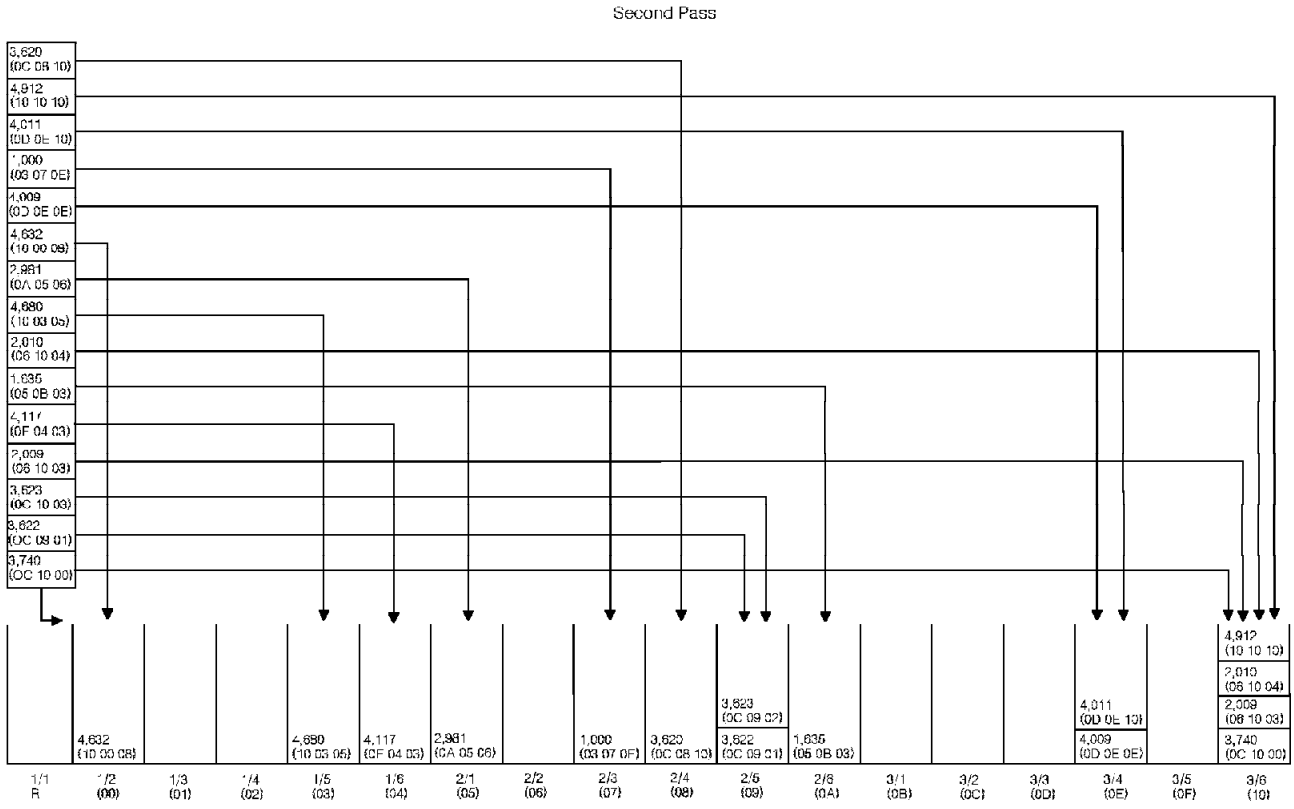


Figure 2-3. Sample Sort—Second Pass

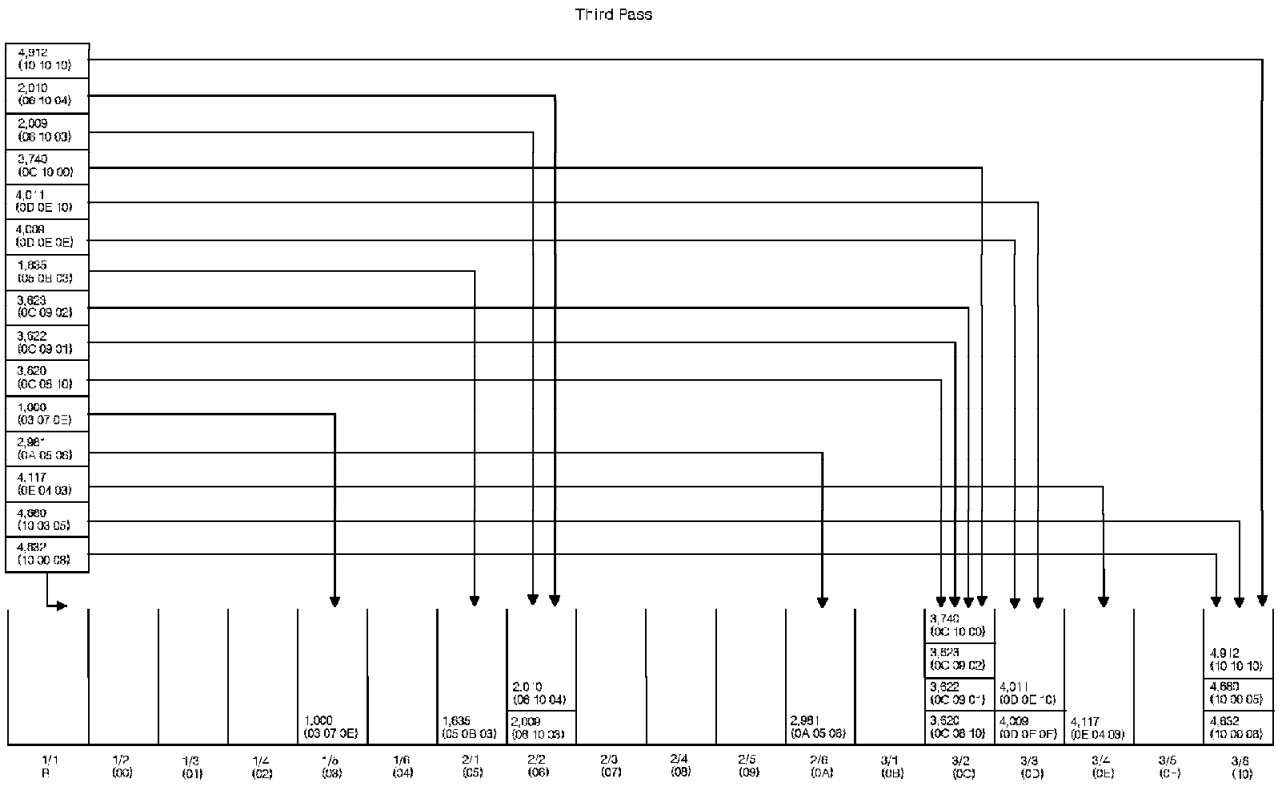


Figure 2-4. Sample Sort—Third Pass

Base-Number Conversion with Compression

In addition to base-number conversion as a method of decreasing the number of sorting passes, you can also use a process named *compression* to decrease even more the number of times documents must pass through the 3890/XP during fine-sort operations.

Compression eliminates large gaps of unused numbers and decreases the remaining numbers.

Example:

Account numbers with known ranges.

```
 10,005,559
  9,834,321----------requires 7 passes to sort
  Gap
  5,804,561
  5,045,304
  Gap
  1,234,689
  1,120,134
```

Take lowest range and reduce.

```
  1,120,134 -  1,234,689 =       0 - 114,555
  5,045,304 -  5,804,561 = 114,556 - 873,813
  9,834,321 - 10,005,559 = 873,814 - 1,045,052
```

In this example, the gaps are eliminated and the highest account number is decreased by one position, with a lower number of item passes.

Sorting Charts

You can use the following charts to find the number of passes required to perform a fine sort using the converted-base technique.

You must know the number of pockets available for the sort and the field value on which you are sorting.

Number of Pockets Available (Base) = 11

Maximum Field Value	Passes Required	Maximum Field Value	Passes Required
0 to 10	1	1 771 560	6
120	2	19 487 170	7
1 330	3	214 358 880	8
14 640	4	2 357 947 690	9
161 050	5	25 937 424 600	10

Number of Pockets Available (Base) = 12

Maximum Field Value	Passes Required	Maximum Field Value	Passes Required
0 to 11	1	2 985 983	6
143	2	35 831 807	7
1 727	3	429 981 695	8
20 735	4	5 159 780 351	9
248 831	5	61 917 364 223	10

Number of Pockets Available (Base) = 13

Maximum Field Value	Passes Required	Maximum Field Value	Passes Required
0 to 12	1	4 826 808	6
168	2	62 748 516	7
1 196	3	815 730 720	8
28 560	4	10 604 499 372	9
371 292	5	137 858 491 848	10

Number of Pockets Available (Base) = 14

Maximum Field Value	Passes Required	Maximum Field Value	Passes Required
0 to 13	1	7 529 535	6
195	2	105 413 503	7
2 743	3	1 475 789 055	8
38 415	4	20 661 046 783	9
537 823	5	289 254 654 975	10

Number of Pockets Available (Base) = 15

Maximum Field Value	Passes Required	Maximum Field Value	Passes Required
0 to 14	1	11 390 624	6
224	2	170 859 374	7
3 374	3	2 562 890 624	8
50 624	4	38 443 359 374	9
759 374	5	576 650 390 624	10

Number of Pockets Available (Base) = 16

Maximum Field Value	Passes Required	Maximum Field Value	Passes Required
0 to 15	1	16 777 215	6
255	2	268 435 455	7
4 095	3	4 294 967 295	8
65 535	4	68 719 476 735	9
1 048 575	5	1 099 511 627 775	10

Number of Pockets Available (Base) = 17

Maximum Field Value	Passes Required	Maximum Field Value	Passes Required
0 to 16	1	24 137 568	6
288	2	410 338 672	7
4 912	3	6 975 757 440	8
83 520	4	118 587 876 496	9
1 419 856	5	2 015 993 900 449	10

Number of Pockets Available (Base) = 18

Maximum Field Value	Passes Required	Maximum Field Value	Passes Required
0 to 17	1	34 012 223	6
323	2	612 220 031	7
5 831	3	11 019 960 575	8
104 975	4	198 359 290 367	9
1 889 567	5	3 570 467 226 623	10

Number of Pockets Available (Base) = 19

Maximum Field Value	Passes Required	Maximum Field Value	Passes Required
0 to 18	1	47 045 880	6
360	2	893 871 738	7
6 858	3	16 983 653 040	8
130 320	4	3228 687 876 778	9
2 476 098	5	6 131 066 257 800	10

Number of Pockets Available (Base) = 20

Maximum Field Value	Passes Required	Maximum Field Value	Passes Required
0 to 19	1	63 999 999	6
399	2	1 279 999 999	7
7 999	3	25 599 999 999	8
159 999	4	511 999 999 999	9
3 199 999	5	10 239 999 999 999	10

Number of Pockets Available (Base) = 21

Maximum Field Value	Passes Required	Maximum Field Value	Passes Required
0 to 20	1	85 766 120	6
440	2	1 801 088 540	7
9 260	3	37 822 859 360	8
194 480	4	794 280 046 580	9
4 084 100	5	16 679 880 978 200	10

Number of Pockets Available (Base) = 22

Maximum Field Value	Passes Required	Maximum Field Value	Passes Required
0 to 21	1	113 379 903	6
483	2	2 494 357 887	7
10 647	3	54 875 873 535	8
234 255	4	1 207 269 217 791	9
5 153 631	5	26 559 922 791 423	10

Number of Pockets Available (Base) = 23

Maximum Field Value	Passes Required	Maximum Field Value	Passes Required
0 to 22	1	148 035 888	6
528	2	3 404 825 446	7
12 166	3	78 310 985 280	8
279 840	4	1 801 152 661 462	9
6 436 342	5	41 426 511 213 648	10

Number of Pockets Available (Base) = 24

Maximum Field Value	Passes Required	Maximum Field Value	Passes Required
0 to 23	1	191 102 975	6
575	2	4 586 471 423	7
13 823	3	110 075 314 175	8
331 775	4	2 641 807 540 223	9
7 962 623	5	63 403 380 965 375	10

Number of Pockets Available (Base) = 25

Maximum Field Value	Passes Required	Maximum Field Value	Passes Required
0 to 24	1	244 140 624	6
624	2	6 103 515 624	7
15 624	3	152 587 890 624	8
390 624	4	3 814 697 265 624	9
9 765 624	5	95 367 431 640 624	10

Number of Pockets Available (Base) = 26

Maximum Field Value	Passes Required	Maximum Field Value	Passes Required
0 to 25	1	308 915 775	6
675	2	8 031 810 175	7
17 575	3	208 827 064 575	8
456 975	4	5 429 503 678 975	9
11 881 375	5	141 167 095 653 375	10

Number of Pockets Available (Base) = 27

Maximum Field Value	Passes Required	Maximum Field Value	Passes Required
0 to 26	1	387 420 488	6
728	2	10 460 353 202	7
19 682	3	282 429 536 480	8
531 440	4	7 625 597 484 986	9
14 348 906	5	205 891 132 094 648	10

Number of Pockets Available (Base) = 28

Maximum Field Value	Passes Required	Maximum Field Value	Passes Required
0 to 27	1	481 890 303	6
783	2	13 492 928 511	7
21 951	3	377 801 998 335	8
614 655	4	10 578 455 953 407	9
17 210 367	5	296 196 766 695 423	10

Number of Pockets Available (Base) = 29

Maximum Field Value	Passes Required	Maximum Field Value	Passes Required
0 to 28	1	594 823 320	6
840	2	17 249 876 308	7
24 388	3	500 246 412 960	8
707 280	4	14 507 145 975 868	9
20 511 148	5	420 707 233 300 200	10

Number of Pockets Available (Base) = 30

Maximum Field Value	Passes Required	Maximum Field Value	Passes Required
0 to 29	1	728 999 999	6
899	2	21 869 999 999	7
26 999	3	656 099 999 999	8
809 999	4	19 682 999 999 999	9
24 299 999	5	590 489 999 999 999	10

Number of Pockets Available (Base) = 31

Maximum Field Value	Passes Required	Maximum Field Value	Passes Required
0 to 30	1	887 503 680	6
960	2	27 512 614 110	7
29 790	3	852 891 037 440	8
923 520	4	26 439 622 160 670	9
28 629 150	5	819 628 286 980 800	10

Number of Pockets Available (Base) = 32

Maximum Field Value	Passes Required	Maximum Field Value	Passes Required
0 to 31	1	1 073 741 823	6
1 023	2	34 359 738 367	7
32 767	3	1 099 511 627 775	8
1 048 575	4	35 184 372 088 831	9
33 554 431	5	1 125 899 906 842 623	10

Number of Pockets Available (Base) = 33

Maximum Field Value	Passes Required	Maximum Field Value	Passes Required
0 to 32	1	1 291 467 968	6
1 088	2	42 618 442 976	7
35 936	3	1 406 408 618 240	8
1 185 920	4	46 411 484 401 952	9
39 135 392	5	1 531 578 985 264 448	10

Number of Pockets Available (Base) = 34

Maximum Field Value	Passes Required	Maximum Field Value	Passes Required
0 to 33	1	1 544 804 415	6
1 155	2	52 523 350 143	7
39 303	3	1 785 793 904 895	8
1 336 335	4	60 716 992 766 463	9
45 435 423	5	2 064 377 754 059 775	10

Number of Pockets Available (Base) = 35

Maximum Field Value	Passes Required	Maximum Field Value	Passes Required
0 to 34	1	1 838 265 624	6
1 224	2	64 339 296 874	7
42 874	3	2 251 875 390 624	8
1 500 624	4	78 815 638 671 874	9
52 521 874	5	2 758 547 353 515 624	10

BCS (Branch On Condition)

(BC m0 aa aa)
BCS (D0 m0 aa aa aa aa)

[Symbol]	BCS	m, label

The **BCS** (branch-on-condition) macro permits the address of **label** to replace the updated SCI-address if the condition-code state is as specified by **m**.

The BCS macro supports 2-byte addresses or 4-byte addresses. If you use the 4-byte format, set the high-order byte of the address to zero.

The **m** parameter is a four-bit mask. The four bits of the mask correspond, from left to right, with the four condition codes (0, 1, 2, and 3) as follows:

Mask Position Value	Condition Code
8	0
4	1
2	2
1	3

The branch is successful when the condition code has a corresponding mask-bit of one. When a branch is made on more than one condition, the pertinent condition codes are specified in the mask as the sum of their mask-position values. A mask of 12, for example, specifies that a branch is made on condition codes 0 and 1. When all four mask-bits are ones, (that is, the mask-position value is 15), the branch is unconditional. When all four mask-bits are zeroes, the branch does not take place.

Resulting Condition Code: Not changed.

SCI Error: Address not valid—specified address is odd. An address past the end of program storage results in an address error when the next instruction is processed.

Example:

```
        THI 8,X'02'   Test for high-order zero.
        BCS 1,HOZRT   If on, branch to High-Order Zero Correction routine.
        THI 8,X'01'   Test for symbol error correction.
        BCS 1,SECRT   If on, branch to Symbol Error Correction routine.
SECRT   SPI 36        Select document to pocket 6, module 3.
         .
         .
HOZRT   SPI 35        Select document to pocket 5, module 3.
```

BLS (Branch and Link)

BLS **(BD r0 aa aa)**
(D1 r0 aa aa aa aa)

[Symbol]	BLS	r, label

The **BLS** (branch-and-link) macro saves the address of the next sequential SCI in a register and gives control to the SCI specified by **label**. The register can be any one of four registers, 0-3.

The BLS macro supports 2-byte addresses or 4-byte addresses. If you use the 4-byte format, set the high-order byte of the address to zero.

Resulting Condition Code: Not changed.

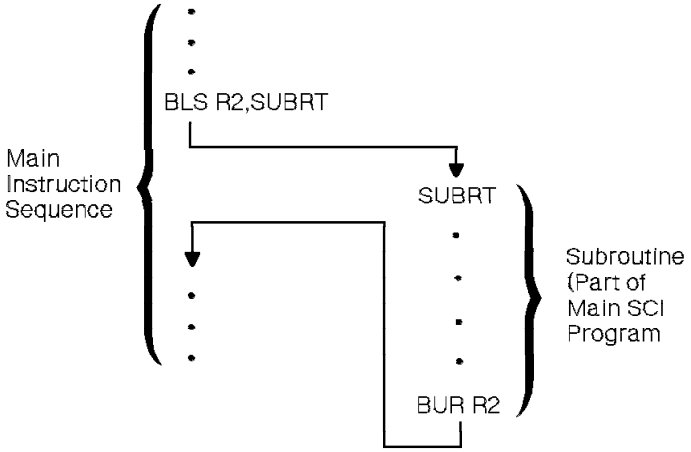
SCI Error:

Address not valid—specified address is odd. An address past the end of program storage results in an address error when the next instruction is processed.

Link register not valid—Register specified is not 0-3.

Example:

Branch and link is normally used to branch to a subroutine with the option of later returning to the main routine through the BUR.



BUR (Branch On Register)

BUR ***(BE r0)***

[Symbol]	BUR	r

The **BUR** (branch-on-register) macro gives the capability of returning to an SCI whose address was saved earlier in a link register. The register can be any one of four registers, 0-3. For an example of how to use this instruction, see the BLS (branch-and-link) macro.

Resulting Condition Code: Not changed.

SCI Error: Link register not valid—specified register is not 0-3.

CA (Compare Work Register to Save Area)

CA ***(4B dl 0s ss)***

[Symbol]	CA	(d, l), s

The **CA** (compare-work-register-to-save-area) macro causes the work register digits specified by **d** and **l** (length in digits) to be compared logically to the specified location in the save area. The comparison is binary and moves from right to left.

Resulting Condition Code:

0 Work register equal to save area
1 Work register less than save area
2 Work register more than save area
3 -

SCI Error: Save area location not valid—**s** is odd or larger than 510.

CALLNATV (Call Native)

CALLNATV ***(F8 00 aa aa aa aa)***

[Symbol]	CALLNATV	label

The **CALLNATV** (call-native) macro *calls* a native language subroutine from the SCI program. This macro has a single parameter, **label**, that specifies the address of the name of the native language subroutine to call. The name of the subroutine can be from 1 to 31 EBCDIC characters in length and must end with a blank.

Resulting Condition Code:

0 Good
2 Non-zero code returned in AX (accumulator) register.

Note: A return code of zero indicates that the function completed without error. No action results from the non-zero return code except the loading of the return code into the native-return-code field of auxiliary storage and the setting of this condition code.

SCI Error: Not found.

CCI (Compare Counter Immediate)

CCI ***(AC c0 ii ii)***

[Symbol]	CCI	c, i

The **CCI** (compare-counter) macro lets you compare a specified counter to the value of the two bytes of immediate data. Treat both values as unsigned, 16-bit, binary numbers with a decimal range from 0 to 65 535 or a hexadecimal range from X'0000' to X'FFFF'. **'c'** is one of 16 two-byte, binary counters that are numbered 0-15.

Resulting Condition Code:

0 Counter equal to immediate
1 Counter less than immediate
2 Counter greater than immediate
3 -

SCI Error: None

CHI (Compare Header Immediate)

CHI ***(8h ii)***

[Symbol]	CHI	h, i

The **CHI** (compare-header-immediate) macro permits comparison of the contents of the specified header byte to the value of the single byte of immediate data which has a decimal range from 0 to 255 or a hexadecimal range from X'00' to X'FF'. Treat both values as unsigned, binary data. **h** can be any one of the header bytes 0-11.

Resulting Condition Code:

0 Header equal to immediate
1 Header less than immediate
2 Header greater than immediate
3 -

SCI Error: Header byte not valid—header byte specified is not 0-11.

CI (Compare Work Register Immediate)

CI ***(48 dl ii ii)***

[Symbol]	CI	(d, l), i

The **CI** (compare-work-register-immediate) macro logically compares the work register digits specified by **d** and **l** to immediate data of up to four digits. Both areas are processed from right to left and treat all data as a 4-, 8-, 12-, or 16-bit, binary number with a decimal range from 0 to 65 535 or a hexadecimal range from X'0000' to X'FFFF'.

Resulting Condition Code:

0 Work register equal to immediate
1 Work register less than immediate
2 Work register greater than immediate
3 -

SCI Error: Length not valid—length specified is not 1-4.

Example:

```
RDSW    LSW                 Load sort type from switches
                            into work register.
        CI    (0,2),X'17'   Is this base 17 sort?
        BCS   8,CONV17      If yes, branch to base
                            17 sort routine.
```

CST (Compare and Select)

CST		(5C dl aa aa) (D2 dl aa aa aa aa)
[Symbol]	CST	(d, l), label

The **CST** (compare-and-select) macro sets the MP code by logically comparing the work register digits specified by **d** and **l** (length in digits) to sequential table entries at **label**. The CSTBL macro creates this table. The comparison moves from left to right.

If the accessed digits of the work register match an entry in the table being scanned, the MP code is taken from that table.

The CST macro supports 2-byte addresses or 4-byte addresses.

For an example of the combined use of the CST and CSTBL macros, see "CSTBL (Scan Table)" on page 2-25. If you use the 4-byte format, set the high-order byte of the address to zero.

Resulting Condition Code:

0 Equal entry found
1 Equal entry not found
2 -
3 -

SCI Error:

Address not valid—part or all of the specified data is at an address above the end of program storage.

CSTBL (Scan Table)

CSTBL

[Symbol]	CSTBL	p, a, b, ..., T

The **CSTBL** (scan table) macro creates a table to be used with the CST (compare-and-select) macro.

Normally, one scan table definition macro is used for each pocket within the specific job application and these are coded sequentially, with the last scan table definition being the default or your reject pocket. The first table includes the symbol in the name field corresponding to the label in the CST macro. Later table entries have optional name entries. The parameters for the CSTBL macro have the following meanings:

p A self-defining term, two digits that specify the MP code. Each digit has a value range from one to six (a maximum of six modules with six pockets each).

a,b,...
A variable number of equal-length operands that are from 1 to 16 digits, separated by commas. The length of the longest operand is determined, and shorter operands are padded on the left with zeroes.

T The last entry for a scan table sequence with the MP code for the default or your reject pocket. When you enter this, specify only the **p** and **T** operands. If you specify **p** and do not specify **T**, the present MP code is not changed.

This table has the following format:

Byte 0 Number of entries

Byte 1 MP code

Bytes 2-n Table entries (n varies with the number of table entries).

Note: You can define a variable number of CSTBLs. The last CSTBL has a value of zero in byte 0.

Example:

This example shows the combined use of the CST and CSTBL macros.

The document is routed to pocket 2 of module 2, and condition code **0** (equal entry found) is returned to the CST instruction.

CT (Compare Work Register to Table)

		(5B dl aa aa)
CT		**(D3 dl aa aa aa aa)**
[Symbol]	CT	(d, l), label

The **CT** (compare-work-register-to-table) macro logically compares the work register digits specified by **d** and **l** (length in digits) to the sequential table entries at **label**. The CTBL (compare table) macro creates the table.

Each comparison searches from left to right for either an equal condition or the end of the table. Odd lengths are filled in the high-order digit with zeros.

The CT macro supports 2-byte addresses or 4-byte addresses. If you use the 4-byte format, set the high-order byte of the address to zero.

Resulting Condition Code:

0 Equal entry found
1 No equal entry found
2 -
3 -

SCI Error:

Address not valid—part or all of the specified data is at an address above the end of program storage.

CTBL (Compare Table)

CTBL

[Symbol]	CTBL	a, b, ...

The **CTBL** (compare table) macro creates a table for use with the CT (compare-work-register-to-table) macro.

The parameters for the CTBL macro have the following meanings:

a,b,...
A variable number of operands of from 1 to 16 digits, separated by commas. The length of the longest operand is determined, and shorter operands are padded on the left with zeroes.

This table has the following format:

Bytes 0-1 Number of entries

Bytes 2-n Table entries (n varies with the number of table entries).

CV (Convert Base)

CV	(5F dl aa aa) (D4 dl aa aa aa aa)	
[Symbol]	CV	(d, l), label

The **CV** (convert base) macro converts the work register digits specified by **d** and **l** (length in digits) from base 10 to the new base specified in the base table at **label**. The CVTBL macro creates the base table.

Any number of unused digits or dashes (SS4s) can be interspersed in the work register data but must be included in the **l** (length). The CVTBL coding is responsible for skipping over this type of information.

The converted result is in hexadecimal and occupies one byte for each new digit in the new base. The resulting field is right-aligned and replaces the previous contents in the work register. Unused high-order bytes of the work register are loaded with zeroes.

The CV macro supports 2-byte addresses or 4-byte addresses. If you use the 4-byte format, set the high-order byte of the address to zero.

Acceptable new bases are 10-36. For an example of the combined use of the CV and CVTBL (convert base table) macros in a sample sort, see “Base-Number Sorting Techniques” on page 2-7.

Resulting Condition Code:

0 Converted as specified
1 Converted and short field (fill character detected before **d** and **l** satisfied)
2 -
3 Result not valid (decimal digit was detected that is not valid).

SCI Error:

Address not valid—part or all of the specified data is at an address above the end of program storage.

Table data not valid—base less than 10.

CVTBL (Convert Base Table)

CVTBL

[Symbol]	CVTBL	p, l, (a, b, c,...)

The **CVTBL** (convert base table) macro creates the base table used with the CV (convert base) macro. The operands specify which base conversion to produce and which digit position(s) to skip.

Note: Table data must start on an even-address boundary.

The parameters for the CVTBL macro have the following meanings:

p An absolute self-defining operand 10-255 specifying the base. Bases 37-255 cannot be used for sort conversion but are available for special processing.

l An absolute self-defining operand specifying the length of the number to be converted, including any imbedded, unused digits or dashes (SS4s).

(a,b,c,...)
Self-defining terms specifying the digit positions to skip. For example, positions three and eight consist of dashes (SS4s) and must be skipped. The coding is:

```
Symbol CVTBL p,l,(3,8)
```

For an example of the combined use of both the CV and CVTBL macros in a sample sort, see “Base-Number Sorting Techniques” on page 2-7.

The base table has the following format:

Byte 0 New base (binary)

Byte 1 Not used

Bytes 2-n Groups (one group per digit to convert) (**n** varies with the number of digits to convert).

Each group in the base table (bytes 2-n) has the following format:

Byte 0 Number of bytes per entry

Byte 1 Total bytes in group (-2)

Bytes 2-n Nine entries—for digits 1-9 (**n** varies with the number of bytes per entry).

DSG (Disengage)

DSG ***(FA 00)***

[Symbol] DSG

The **DSG** (disengage) macro stops the feeding of all documents from the main hopper. Merge documents feed until all merge requests are honored. This instruction ends immediately; it does not wait for feeding to stop. The SCI program handles all in-path documents normally. Restarting the machine requires operator intervention.

Resulting Condition Code: Not changed.

SCI Error: None.

DSGA (Disengage and Set Operator Message Attention)

DSGA ***(FB 00)***

[Symbol] DSGA

The **DSGA** (disengage-and-set-operator-message-attention) macro stops the feeding of all documents from the main hopper, and the operator message appears on
the display. Merge documents feed until all merge requests are honored. This instruction ends immediately; it does not wait for feeding to stop. The SCI program handles all in-path documents normally. Restarting the machine requires operator intervention.

Resulting Condition Code: Not changed.

SCI Error: None.

EXC (Execute)

(F9 d0 aa aa)
EXC (D5 d0 aa aa aa aa)

[Symbol]	EXC	d,label

The work register value at **d** (one byte) is used to change (**OR**) the instruction at **label** (bits 8-15). The resulting instruction is processed as if it were the next sequential instruction in the program.

The ORing does not change the contents of the work register or the actual instruction in storage. It is effective only for interpretation of the processing of the subject instruction.

The EXC macro supports 2-byte addresses or 4-byte addresses. If you use the 4-byte format, set the high-order byte of the address to zero.

Execution and SCI Errors are exactly the same as if the subject instruction were obtained through normal processing, except that the instruction address-and-length code are that of the execute macro.

SCI Error:

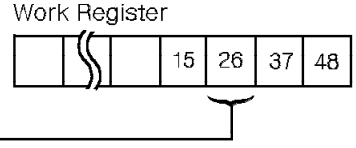
Address not valid—subject instruction is at a hexadecimal address that is above the end of program storage.

Execute—subject instruction is an execute.

Example:

Move the MP code from displacement 2 in the work register to header byte 6.

```
NHI   6,0      Clear Header Byte 6.
EXC   4,MVMP
 .
 .
 .
MVMP  OHI  6,0
```



The instruction at MVMP effectively becomes OHI 6, 26 and is processed as if it were the next sequential instruction.

EXT (Exit (End SCI))

EXT ***(FF 00)***

[Symbol]	EXT

Always specify the **EXT** (exit) macro as the last SCI of any logical selection path.

FM (Feed from Merge)

FM ***(FC 00)***

[Symbol]	FM

The **FM** (feed-from-merge) macro interrupts the main feed to permit one document to feed from the merge hopper. The feed-from-merge SCI instruction ends immediately and does not wait for the merge document to feed. This instruction can be processed once per sort program cycle; later feed-from-merge SCIs set condition code **1** and no merge documents feed.

Resulting Condition Code:

0 Merge document requested
1 Merge request ignored
2 -
3 -

SCI Error: None.

IA (Insert from Save Area into Work Register)

IA ***(4C dl 0s ss)***

[Symbol]	IA	(d, l), s

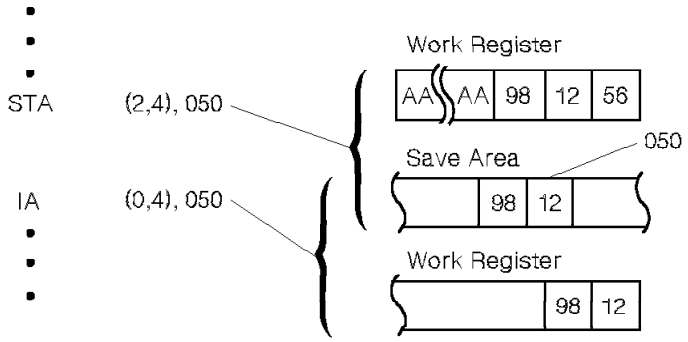
The **IA** (insert-from-save-area) macro moves the contents of the specified location in the save area into the work register positions specified by **d** and **l** (length in digits). Both areas are processed from right to left, and the unused positions of the work register and the complete save area are not changed.

Resulting Condition Code: Not changed.

SCI Error: Save area location not valid—**s** is odd or more than 510.

Example:

This is an example of shifting right in the work register using both the STA and IA macros.



IFD (Initialize Feature Data)

IFD ***(FD i0)***

[Symbol]	IFD	i

The **IFD** (initialize feature data) macro causes one of five functions to take place, depending on the value of **i**. This instruction is processed only in offline mode.

'i'	Function
0	Disengage. Requests feature initialization and stores the low-order byte of the work register in initialization data byte 0.
1	Stores the low-order five bytes of the work register in initialization data bytes 1-5 (feature 1 - item numbering).
2	Stores the low-order five bytes of the work register in initialization data bytes 6-10 (feature 2 - endorsing).
3	Stores the low-order five bytes of the work register in initialization data bytes 11-15 (reserved).
4	Stores the low-order five bytes of the work register in initialization data bytes 16-20 (feature 4 - microfilming).

At feature initialization, initialization data byte 0, bits 0-3 causes the control program to take the following action:

Bit	Action
0	Send initialization data bytes 1 to 5 to feature 1.
1	Send initialization data bytes 6 to 10 to feature 2.
2	Reserved.
3	Send initialization data bytes 16 to 20 to feature 4.

Note: Bits 4-7 of byte 0 are also changed but have no effect. For these bits to become effective, you must enter 00 into the program switches and press Read Disk.

Resulting Condition Code: Not changed.

SCI Error:

Parameter not valid—**i** not 0-4.

Document processor online—the document processor is online when it attempts to process this instruction.

Example:

In the following example, features 1 and 2 are updated:

```
1.  IFD 1
     .
     .
2.  IFD 2
     .
     .
3.  IFD 0
```

1. Sets up initialization data bytes 1 to 5.
2. Sets up initialization data bytes 6 to 10.
3. Sets up initialization data byte 0 and disengages document feeding. After the sort program handles all in-path documents, the system unit initializes the features. At this time, if bits 0 and 1 are *on* in data byte 0, features 1 and 2 are updated.

Note: Restarting the machine requires operator intervention.

II (Insert into Work Register Immediate)

II ***(47 dl ii ii)***

[Symbol]	II	(d, l), i

The **II** (insert-into-work-register-immediate) macro permits the work register digits specified by **d** and **l** to be replaced by immediate data of up to four digits.
Both areas are processed right to left and treat all data as a 4-, 8-, 12-, or 16-bit, binary number with a decimal range from 0 to 65 535 or a hexadecimal range from X'0000-FFFF'.

Resulting Condition Code: Not changed.

SCI Error: Length not valid—length specified is not 1-4.

Example:

Display message 15 to the operator.
(See “SOM (Set Operator Message from Work Register)” on page 2-57.)

```
.
.
II    (0,2),X'0F'
SOM
.
.
```

IMAT (Extended Code Line Data Match)

IMAT **(C3 ii)**
(C5 00 ii ii)

[Symbol]	IMAT	a,b,c,...

The **IMAT** (extended code line data match) macro permits extended code line data matching under sort program control.

a,b,c,...
Parameters separated by commas. They specify the fields participating in the extended code line data match process. The parameters are not positional, so the fields can be in any sequence. This SCI has two formats.

When the format is the 2-byte hexadecimal value X'C3 ii', **ii** represents the field mask designating the fields to process. Each bit of the mask corresponds to a field to match. Bit 0 corresponds to field 1, bit 1 to field 2, and so forth. Acceptable field values are 1 through 8.

When the format is the 4-byte hexadecimal value X'C5 00 ii ii', where **ii ii** represents the field mask designating the fields to process. Each bit of the mask corresponds to a field to match. Bit 0 corresponds to field 1, bit 1 to field 2, and so forth. Acceptable field values are 1 through 15. Note that **bit 14** corresponds to **field 15**; there is not a bit 15.

Resulting Condition Code:

0 Code line data match (process buffer overlayed).
1 Code line data not found (process buffer not changed).
2 Field not valid (not found in the initialization data, process buffer not changed).
3 A previous extended code line data match SCI successfully overlayed the process buffer, or initialization data byte 21, bit 3 equals zero (process buffer not changed).

SCI Error: None.

Example:

IMAT 1,3,5,15 Permits extended code line data matching for fields 1, 3, 5, and 15.

JX (Jump On Table Index)

(BF d0 aa aa)
JX **(D6 d0 aa aa aa aa)**

[Symbol]	JX	d, label

The **JX** (jump-on-table-index) macro gives the means for dynamically changing the SCI processing sequence based on the numeric value of a digit in a table. The work register digit specified by the **d** parameter indexes the table addressed by **label**; that is, if the work register digit is zero, the first byte of the table is read; if the work register digit is one, the second byte is read, and so forth.

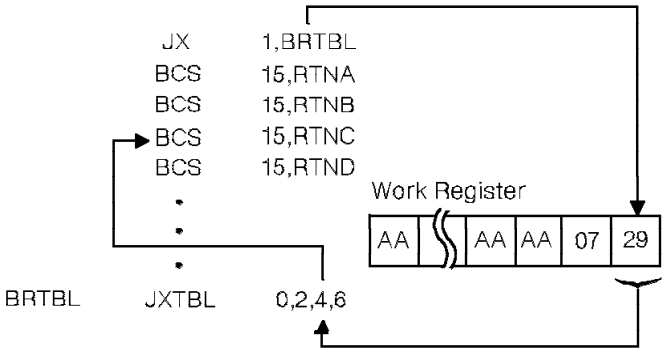
The accessed table byte specifies the number of halfwords to skip. If the accessed table byte is zero, the next sequential SCI is processed; if it is one, then one halfword is skipped, and so forth. The JXTBL (digit-index-table) macro creates the table.

The JX macro supports 2-byte addresses or 4-byte addresses. If you use the 4-byte format, set the high-order byte of the address to zero.

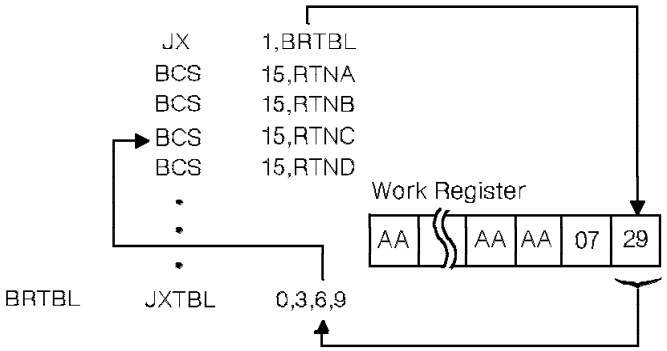
Resulting Condition Code: Not changed.

SCI Error: Address not valid—specified address is odd. An address past the end of program storage results in an address error when the next instruction is processed.

Example 1: Two-byte addressing in the BCSs.



Example 2: Four-byte addressing in the BCSs.



JXTBL (Digit Index Table)

JXTBL

[Symbol]	JXTBL	a, b, c, ...,p

The **JXTBL** (digit index table) macro creates a table for use with the JX (jump-on-table-index) macro. The JX macro indexes into this table and the accessed byte specifies the number of halfwords to skip in the sort program routine. The digit index table has the following format:

Byte 0 Halfwords to skip if digit = 0.

Byte 1 Halfwords to skip if digit = 1.

...

Byte 15 Halfwords to skip if digit = F.

The parameters for the JXTBL macro have the following meanings:

a,b,c,...,p
One to sixteen operands separated by commas. Each operand is a self-defining term from 0-255. Omitted operands are given a value of zero.

For an example of how to use this table-building macro, see “JX (Jump On Table Index)” on page 2-36.

LAS (Load Work Register from Auxiliary Storage)

LAS **(C1 0l aa aa)**
(D7 0l aa aa aa aa)

[Symbol]	LAS	l, a

The **LAS** (load-work-register-from-auxiliary-storage) macro starts with the specified address in auxiliary storage and loads data from right to left into the work register for the length (**l**) of bytes specified. You must specify the auxiliary storage address in absolute hexadecimal form. See the section about an accessible storage map for load/store SCIs in the *3890/XP Series Programming Guide*.

The LAS macro supports 2-byte addresses or 4-byte addresses. If you use the 4-byte format, set the high-order byte of the address to zero.

Resulting Condition Code: Not changed.

SCI Error:

The third hexadecimal digit of the operation code is not 0, 1, 2, or 3.

Address not valid—either the specified address is outside LAS-accessible auxiliary storage, or the specified address, minus the specified byte length, is outside LAS-accessible auxiliary storage.

LCI (Load Counter Immediate)

LCI ***(AB c0 ii ii)***

[Symbol]	LCI	c, i

The **LCI** (load-counter-immediate) macro permits the loading of one of 16 counters, specified by **c**, with two bytes of immediate data with a decimal range from 0 to 65 535 or a hexadecimal range from X'0000' to X'FFFF'.

Resulting Condition Code: Not changed.

SCI Error: None.

Example:

```
Initialize counter to zero.

         TSI     X'20'       First-time switch (user stats).
         BCS     8,INITCTR   When it is 0, go to initialize counter.
          .
          .
          .
INITCTR  LCI     10,0        Initialize counter to 0.
         OSI     X'20'       Sets on first-time switch.
         BCS     15,START    Unconditional branch to START.
```

LF (Load Work Register from Field)

LF ***(3f dl)***

[Symbol]	LF	(d, l), f

The **LF** (load-from-field) macro clears the complete work register to fill characters (X'A's) and then loads the requested length (in digits) of the field, including any SS4s, from right to left directly from the process buffer.

The operation is similar to the LFE (load-first-element) macro until the length is exhausted or a fill character is encountered.

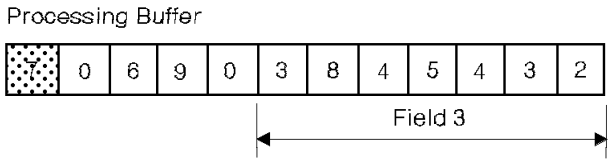
Resulting Condition Code:

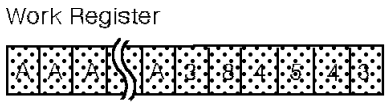
0 Field loaded
1 End-of-field before length exhausted
2 **d** + **l** larger than field specification (specified in the initialization data)
3 Specified field not initialized.

SCI Error: None

Example 1

[Symbol]	LF	(1,6),3





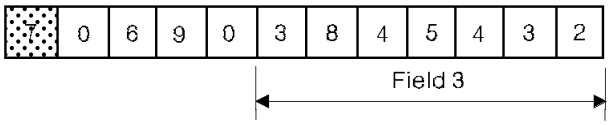
Resulting Condition Code:

0 Field is loaded.

Example 2

```
[Symbol]     LF              (1,10),3
```

Processing Buffer



Work Register

Resulting Condition Code:

2 **d** + **l** larger than field specification.

LFE (Load Work Register from First Element)

LFE ***(1f dl)***

[Symbol]	LFE	(d, l), f

The **LFE** (load-first-element) macro clears the complete work register to fill characters (X'A's) and then loads the work register from right to left for the length (**l**) of digits specified, starting with the first element of the specified field of the process buffer.

The LFE macro moves the first half-byte from the half-byte specified by **d** and **l** to the rightmost digit position of the work register. It moves the remaining half-bytes until one of the following occur:

- The length is exhausted
- The first SS4 (dash) is detected
- A fill character is detected.

The ending SS4 is not loaded into the work register.

Resulting Condition Code:

0 First element loaded as specified
1 End of element before length exhausted (less than **l** digits loaded)
2 **d** + **l** larger than field specification (in the initialization data)
3 Field specified was not initialized.

SCI Error: None.

Note: If **d** is equal to or larger than the number of characters in the field or element specified, no data is loaded (condition code **2**). If **d** is less than the number of characters in the field or element specification, but **d** and **l** extend beyond the field or element specified, the data from **d** to the end of the field or element specification is loaded (condition code **2**).

If the displacement is beyond the first SS4 but within the field specification, condition code **1** is returned.

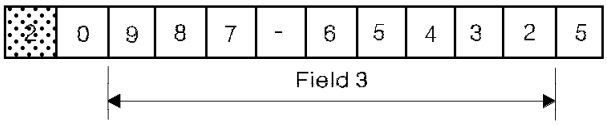
Examples:

In the following examples, field 3 (account number) is specified as a nine-position, variable-length field with dash transmission.

Example 1

```
[Symbol]     LFE          (0,5),3
```

Processing Buffer



Work Register

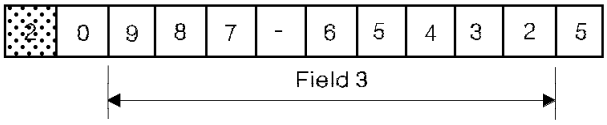
Resulting Condition Code:

0 First element loaded.

Example 2

```
[Symbol]     LFE          (0,6),3
```

Processing Buffer



Work Register

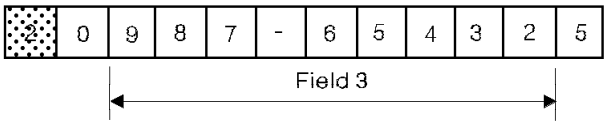
Resulting Condition Code:

1 End of element before length exhausted.

Example 3

```
[Symbol]     LFE          (3,7),3
```

Processing Buffer



Work Register

Resulting Condition Code:

2 **d** + **l** larger than field specification.

LPS (Load Work Register from Program Storage)

		(C1 1l aa aa)
LPS		**(D7 1l aa aa aa aa)**
[Symbol]	LPS	l, label or a

The **LPS** (load-work-register-from-program-storage) macro starts with the specified address in program storage and loads data from right to left into the work register for the length (**l**) of bytes specified.

You must specify with a **label** the storage addresses within the SCI program storage area. You must specify the storage addresses in the initialization data (X'00' to X'7F') in absolute hexadecimal form. If you request 3890/XP emulation and attempt to get access to the process buffer using LPS, the 3890/XP automatically remaps this access to occur at the actual address of the process buffer in 3890/XP-emulated auxiliary storage. These 3890/XP-emulated auxiliary storage addresses also must be in absolute hexadecimal form. See the section about an accessible storage map for load/store SCIs in the *3890/XP Series Programming Guide*.

The LPS macro supports 2-byte addresses or 4-byte addresses.

Resulting Condition Code: Not changed.

SCI Error:

The third hexadecimal digit of the operation code is not 0, 1, 2, or 3.

Address not valid—either the specified address is outside program storage, or the specified address, minus the specified byte-length, is outside program storage.

LSE (Load Work Register from Second Element)

LSE ***(2f dl)***

[Symbol]	LSE	(d, l), f

The **LSE** (load-second-element) macro clears the complete work register to fill characters (X'A's). It then loads the work register from right to left for the length (**l**) of digits specified, starting with the second element of the specified field of the process buffer.

The second element starts with the first digit to the left of the rightmost SS4 and continues to the left through the remainder of the field. The second element can contain one or more SS4s, which have no effect on the operation.

The operation is similar to the load-first-element macro; it loads until the length is exhausted, a fill character is detected, or the end of the field specification is detected.

Resulting Condition Code:

0 Second element loaded
1 End of element before length is exhausted
2 **d** + **l** larger than field specification (specified in the initialization data)
3 Specified field not initialized.

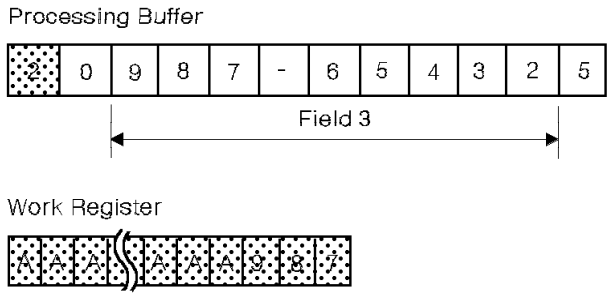
SCI Error: None.

Examples:

In the following examples, field 3 (account number) is specified as a nine-position, variable-length field with dash transmission.

Example 1

[Symbol]	LSE	(0,3),3

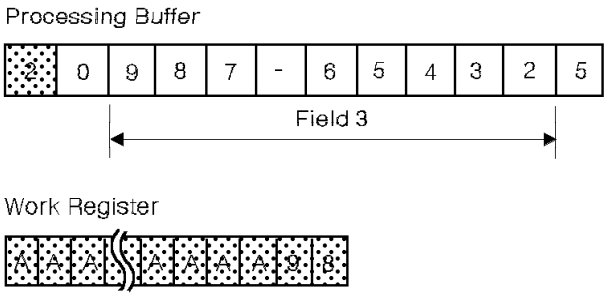


Resulting Condition Code:

0 Second element loaded.

Example 2

```
[Symbol]     LSE          (1,2),3
```

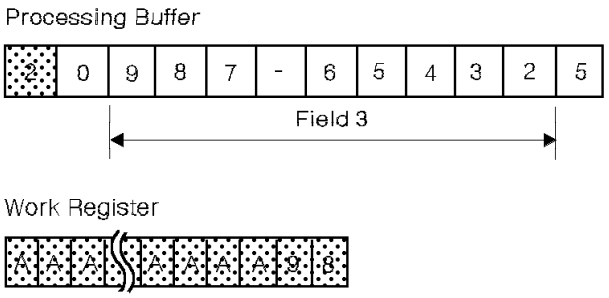


Resulting Condition Code:

0 Second element loaded.

Example 3

```
[Symbol]     LSE          (1,3),3
```



Resulting Condition Code:

2 **d** + **l** larger than field specification.

LSW (Load Work Register from Program Switches)

LSW ***(44 00)***

```
[Symbol]     LSW
```

The **LSW** (load-from-program-switches) macro loads the two half-bytes of the simulated program switches into the two rightmost digit positions of the work register. The remaining digit positions of the work register are cleared to fill characters (X'A's). The program switch values are set by operator command and range in value from X'00' to X'FF' (binary 00000000-11111111).

Resulting Condition Code: Not changed.

SCI Error: None.

NHI (AND Header Immediate)

NHI ***(7h ii)***

[Symbol]	NHI	h, i

The parameters of the **NHI** (AND-header-immediate) macro have the following meanings:

h Specifies the document data record header to read

i Specifies the immediate data to AND to the specified header byte.

The NHI macro *ANDs* the byte of immediate data to the specified byte in the document data header. Treat both values as unsigned, binary data with a decimal range from 0 to 255 or a hexadecimal range from X'00' to X'FF'. Bit 0 of byte 5 (microfilm space) cannot be reset by this instruction.

Resulting Condition Code:

0 Result 0
1 Result not 0
2 -
3 -

SCI Error:

Header byte not valid—header byte specified is not 2, 3, 5, 6, or 9.

Example 1:

This instruction is frequently used to set a specific bit to 0.

```
           NHI    3,X'FE'
 Header Byte 3    01000011
Immediate Data    11111110
        Result    01000010
```

Example 2:

Reset header byte 2 to zero.

```
      NHI  2,X'00'    Any bits on are switched off.
```

NSI (AND User Stats Immediate)

NSI ***(AE ii)***

[Symbol]	NSI	i

The **NSI** (AND-user-stats-immediate) macro permits the byte of immediate data with a value in the decimal range of 0-255 or X'00-FF' to AND to the specified bits of the user stats byte (user program switches).

Resulting Condition Code:

0 Result zero
1 Result not zero
2 -
3 -

SCI Error: None.

Number Self-Checking

Number self-checking on the 3890/XP is under control of the V (verify) and the VTBL (verification table) macros.

Self-checking arithmetic is performed on the contents of the work register as specified in the V macro. A dash or any other special symbol should not participate in the verification; therefore, the corresponding weighting factor should be zero. Weighting factors are a function of the VTBL macro.

The following are descriptions of the more commonly-used modulus 10 and modulus 11 with programming and arithmetic examples. You are not limited to modulus 10 or 11; you can develop formulas to meet your own requirements.

Modulus 10

Operation

1. Each designated digit in the work register is multiplied by its weight factor to get a two-digit product.
2. Add the two digits of each product to get a sum of product digits.
3. Add all the sums of product digits to get a final sum.
4. This final sum minus any remainder (if specified) must be equal to a multiple of 10.

Sample Program

```
        V (0,5),MODTEN
        .
        .
        .
MODTEN VTBL MOD=10,REM=0,WTS=(1,2,1,2,1),
            OPT=1
        .
        .
        .
```

Arithmetic

```
Work Register 5   5   6   7   3
Weight Factor 1   2   1   2   1 (Established modulus 10
                                 weight factors)
Products      05  10  06  14  03
Sum of product  digits 0+5=5  1+0=1  0+6=6  1+4=5  0+3=3
Final Sum 5 + 1 + 6 + 5 + 3 = 20
```

The total of 20 is evenly divisible by 10; therefore, the number 55673 is a legitimate modulus 10 number according to the rules set up in the VTBL macro.

Modulus 11

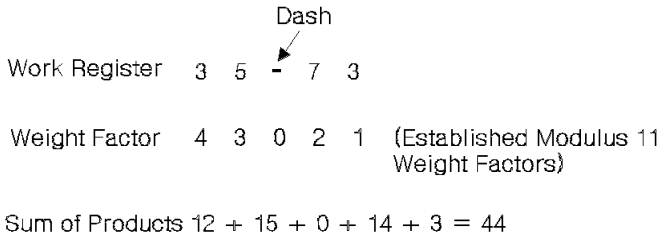
Operation

1. Each designated digit in the work register is multiplied by its weight factor to get a product.
2. Add all the products to get a sum.
3. This sum minus any remainder (if designated) must be equal to a multiple of 11.

Sample Program

```
        V (0,5),MOD11
        .
        .
        .
MOD11 VTBL MOD=11,REM=0,WTS=(4,3,0,2,1),
           OPT=2
        .
        .
        .
```

Arithmetic



The total of 44 is evenly divisible by 11; therefore, the number 3573 is a legitimate modulus 11 number according to the rules set up in the VTBL macro.

OHI (OR Header Immediate)

OHI ***(9h ii)***

[Symbol]	OHI	h, i

The parameters of the **OHI** (OR-header-immediate) macro have the following meanings:

h Specifies the document data record header to read
i Specifies the byte of immediate data to OR to the specified header byte.

The OHI macro *ORs* the byte of immediate data to the specified byte in the document data header. Treat both values as unsigned, binary data with a decimal range from 0 to 255 or a hexadecimal range from X'00' to X'FF'.

Resulting Condition Code:

0 Result 0
1 Result not 0
2 -
3 -

SCI Error:

Header byte not valid—header byte specified is not 2, 3, 5, 6, or 9.

Example:

This instruction is frequently used to set a specific bit to 1.

```
            OHI    9,X'01'

 Header byte 9    01000010
Immediate data    00000001
        Result    01000011
```

OSI (OR User Stats Immediate)

OSI ***(AF ii)***

[Symbol]	OSI	i

The **OSI** (OR-user-stats-immediate) macro permits the byte of immediate data with a value range of decimal 0-255 or X'00-FF' to OR to the specified bits of the user stats byte (user program switches).

Resulting Condition Code:

0 Result zero
1 Result not zero
2 -
3 -

SCI Error: None.

SAS (Store Work Register into Auxiliary Storage)

SAS **(C1 2l aa aa)**
(D7 2l aa aa aa aa)

[Symbol]	SAS	l, a

The **SAS** (store-work-register-into-auxiliary-storage) macro starts at the specified address in auxiliary storage and stores data, from right to left, from the work register into auxiliary storage for the length (**l**) of bytes specified. You must specify the auxiliary storage address in absolute hexadecimal form. See the section about an accessible storage map for load/store SCIs in the *3890/XP Series Programming Guide*.

The SAS macro supports 2-byte addresses or 4-byte addresses. If you use the 4-byte format, set the high-order byte of the address to zero.

Resulting Condition Code: Not changed.

SCI Error:

The third hexadecimal digit of the operation code is not 0, 1, 2, or 3.

Address not valid—either the specified address is outside SAS-accessible auxiliary storage, or the specified address, minus specified byte-length, is outside SAS-accessible auxiliary storage.

SBT (Search Binary)

(5A d0 aa aa)
SBT **(D8 d0 aa aa aa aa)**

[Symbol]	SBT	d, label

The **SBT** (search binary) macro permits the work register value at displacement **d** to be compared to entries in a table at **label** using a binary search technique. The SBTBL macro creates the table. The search stops when there is an equal condition or when the work register value is in a range defined by two consecutive entries in the table.

The matched entry or the lower value of the range is inserted into the work register from right to left, starting at the specified displacement.

If the resulting condition code is 2 or 3, the contents of the work registers are not changed.

The SBT macro supports 2-byte addresses or 4-byte addresses. If you use the 4-byte format, set the high-order byte of the address to zero.

For an example of the combined use of the SBT and SBTBL macros, see page 2-55.

Resulting Condition Code:

0 Equal entry found. (The program storage address of the located equal entry is saved in mapped 3890-emulated auxiliary storage.)
1 Range found.
2 Entry not in list.
3 Displacement plus entry length is outside the work register.

SCI Error:

Address not valid—part or all of the specified data is at an address above the end of program storage.

Displacement not valid—specified displacement is odd.

Table data not valid—entry or comparison length is zero. The entry length is less than the comparison length.

SBTBL (Search Binary Table)

SBTBL

[Symbol]	SBTBL	n, a, b, c,...

The **SBTBL** (search binary table) macro creates two types of tables for use with the SBT (search binary) macro. It produces the search binary table for SCIs with 2-byte addresses and the extended search binary table for SCIs with either 2-byte or 4-byte addresses.

The parameters for the SBTBL macro have the following meanings:

n (must be an even digit other than zero)
Length in digits of the portion of each entry, as specified by **a,b,c,...** , to be used in the comparison against the work register. The value of the portion of each entry to be used in the comparison must be more than that of the previous entry and less than that of the following entry.

a,b,c,...
Each operand is a table entry. The length of each table entry created is equal to the longest operand (rounded up to an even length if necessary). Shorter entries are padded to the left with zeroes. Entries must not be more than 32 digits long. The number of operands is limited only by the amount of storage available.

The tables have the following formats:

Search Binary Table:

Bytes 0-1 Address of last entry

Bytes 2-3 Number of entries

Byte 4 Entry length (bytes)

Byte 5 Compare length (bytes)

Bytes 6-n Table entries. (**n** varies with the number of table entries.)

Extended Search Binary Table:

Bytes 0-3 “SBT4” in EBCDIC

Bytes 4-7 Address of last entry

Bytes 8-11 Number of entries

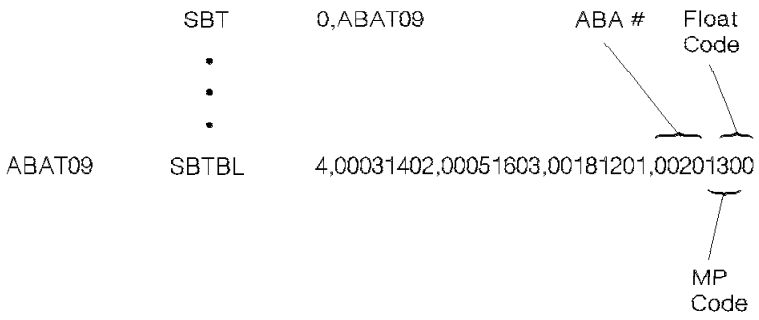
Byte 12 Entry length (bytes)

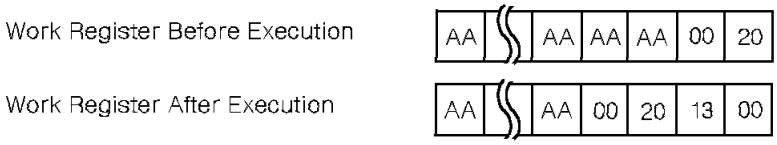
Byte 13 Compare length (bytes)

Bytes 14-n Table entries. (**n** varies with the number of table entries.)

The SBT macro uses a binary search to compare work register values with entries in these tables. Table data must start on an even-address boundary.

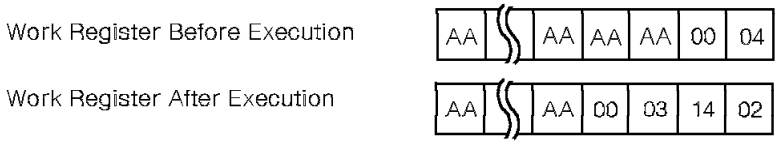
Example:





As a Result of SBT Instruction:

Condition Code Set = 0



As a Result of SBT Instruction:

Condition Code Set = 1

If the condition code is '2' or '3', the contents of the work registers are unchanged.

SFI (Select FC Immediate)

SFI ***(EA ii)***

[Symbol]	SFI	i

The **SFI** (select-feature-immediate) macro permits immediate decimal data from 0 to 255 or hexadecimal data from X'00' to X'FF' to be set into the feature control byte (byte 5) of the document data record header.

Resulting Condition Code: Not changed.

SCI Error: None.

Features can be initiated by either of the following methods:

Initialization data
: Information of a setup nature at the start of a run or when no documents are being processed.

Dynamic control
: Information for each document during normal machine operation.

The SFI macro supplies a method for dynamic feature control. The configuration of the *feature control byte* is as follows:

Bit 0 **Microfilm Space**. This is a privileged bit and can be set, but not reset, by this instruction.

Bit 1 **Microfilm Flash**. Triggers the flashlamp in the microfilm module when the document is in the correct position.

Bit 2 **Increase Microfilm High-Order Segment**. Causes the high-order segment of the index number to increase by one and the low-order segment to reset to zero. This occurs before it puts the code line data on film.

Bit 3 **Increase Microfilm Low-Order Segment**. Causes the low-order segment of the index number to increase by one. This occurs before it puts the code line data on film. When the low-order segment is all 9s and increases by one, it becomes all zeroes. It does not affect the high-order segment.

If bits 2 and 3 are on at the same time, bit 3 is ignored and the action of 2 is taken.

Bit 4 **Inhibit printing of the Endorsement**.

Bit 5 **Inhibit printing of the Item Number**.

Bit 6 **Increase INF High-Order Segment**. Causes the high-order segment to increase by one and the low-order segment to reset to zero. This occurs before the print cycle.

Bit 7 **Increase INF Low-Order Segment**. Causes the low-order segment to increase by one. When the low-order segment is all 9s and increases by one, it becomes all zeroes. It does not affect the high-order segment. This occurs before the print cycle.

If bits 6 and 7 are on at the same time, bit 7 is ignored and the action of bit 6 is taken.

SI (Subtract from Work Register Immediate)

SI ***(49 dl ii ii)***

[Symbol]	SI	(d, l), i

The **SI** (subtract-from-work-register-immediate) macro permits the decreasing of the work register digits specified by **d** and **l** by **l** digits of immediate data. Both areas are processed from right to left and all data is treated as unsigned packed-decimal, although the immediate data must be coded in hexadecimal. The maximum length is four digits with a value range of X'0000' to X'FFFF'.

If the immediate data is more than the specified work register data, the result is in ten's complement form. If the immediate data is less than the work register data, the result is in true form.

Note: Any reference to the terms "plus" and "minus" or the symbols "+" and "-" is for explanation only. No sign is available in any byte in any 3890/XP storage location.

Resulting Condition Code:

0 Result 0 (true form)
1 Result minus (ten's complement form)
2 Result plus (true form)
3 Result not valid (decimal digit that is not valid was detected).

SCI Error: Length not valid—length specified is not 1-4.

SOM (Set Operator Message from Work Register)

SOM ***(45 00)***

[Symbol]	SOM

The **SOM** (set-operator-message) macro requests the display of an operator message from a table indexed by the two rightmost positions of the work register. The contents of the work register are not changed.

When the transport stops, the table text that corresponds to the message number appears on the message line of the display. You can compose your own text messages to display to the operator. If no message is composed for this message number, a default message appears that displays the message number in binary and hexadecimal format.

Resulting Condition Code: Not changed.

SCI Error: None.

SPC (Select MP by Counter)

SPC ***(EC 00)***

[Symbol]	SPC

The **SPC** (select-pocket-by-counter) macro sets the MP code to the MP code of the first pocket counter that reaches or is more than a predetermined value. Your initialization data sets this predetermined value during initialization.

The operation moves from MP 1/1 to 6/6. If no counter reaches, or is more than, the predetermined value, the MP is set to 1/1 (reject).

Resulting Condition Code:

0 MP set for full counter
1 MP set 1/1 (all counters low)
2 -
3 -

SCI Error: None.

Example:

Select a merge document to a full count pocket.

```
         THI  8,X'08'        Is this a merge document?
         BCS  1,MSEL         Yes.  Go to merge select (MSEL).
         BCS  15,PROC1       No.  Continue processing.
          .
          .
          .
MSEL     SPC                 Select merge document to full pocket
                             counter.
```

SPH (Select FC/MP by Header)

SPH ***(EE 00)***

[Symbol]	SPH

The **SPH** (select feature/pocket by header) macro verifies that the code line data match was successful and that the MP code and the feature control (FC) byte (header bytes 6 and 5 respectively) are code line data matched. During successful code line data matching, the host system sends these two bytes.

If code line data matching is not active, or if code line data matching is active and no code line data is found for this document, the bytes are set to zero before SCI processing.

Resulting Condition Code:

0 Code line data found
1 No code line data found
2 Not code line data matching
3 -

SCI Error: None.

SPI (Select MP Immediate)

SPI ***(E9 mp)***

[Symbol]	SPI	mp

The **SPI** (select-pocket-immediate) macro sets the MP code. Acceptable values are 00, 11-16, 21-26, 31-36, ..., and 61-66. For example, an MP code of 5/3 indicates module 5/pocket 3, but it must be coded without the **/** as the value 53. Note that the maximum number of modules or pockets that you can specify is six.

Resulting Condition Code: Not changed.

SCI Error: None.

SPS (Store Work Register into Program Storage)

SPS	(C1 3l aa aa) (D7 3l aa aa aa aa)	
[Symbol]	SPS	l, label or a

The **SPS** (store-work-register-into-program-storage) macro starts at the specified address in program storage and stores data from right to left from the work register into program storage for the length (**l**) of bytes specified.

You must specify with a **label** storage addresses within the SCI program storage. You must specify storage addresses in the initialization data (0000-0128) in absolute hexadecimal form. If you request 3890/XP emulation and attempt to get access to the process buffer using SPS, the 3890/XP automatically remaps this access to occur at the actual address of the process buffer in 3890/XP-emulated auxiliary storage. These 3890/XP-emulated auxiliary storage addresses also must be in absolute hexadecimal form. For more information, see the section about an accessible storage map for load/store SCIs in the *3890/XP Series Programming Guide*.

The SPS macro supports 2-byte addresses or 4-byte addresses.

Resulting Condition Code: Not changed.

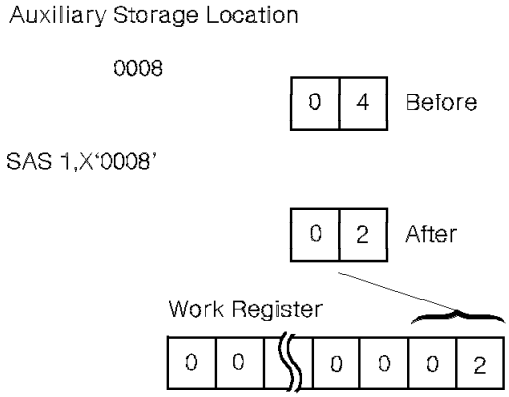
SCI Error:

The third hexadecimal digit of the operation code is not 0, 1, 2, or 3.

Address not valid—either the specified address is outside program storage, or the specified address, minus specified byte-length, is outside program storage.

Examples:

- Redefine the high-order-zero correction parameter of digit errors to be corrected based on control document data.

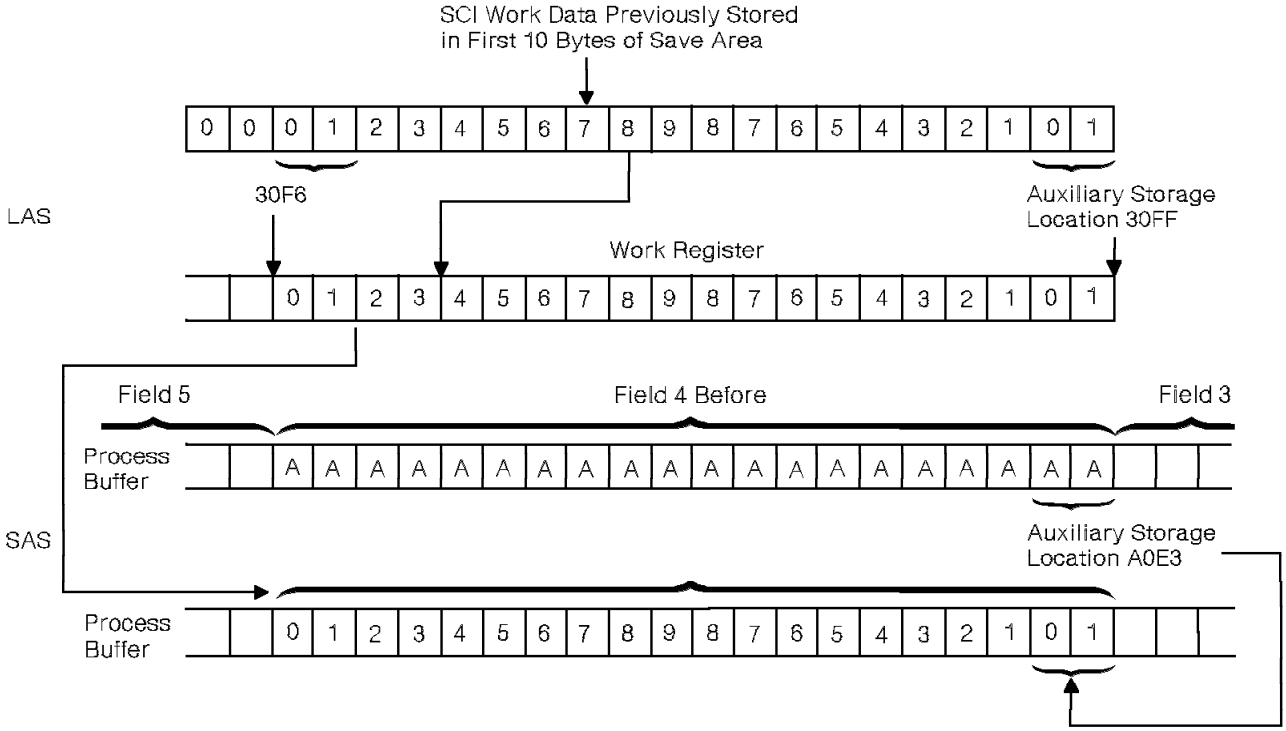


- Store SCI work information into an area of the process buffer that has either no document data or nonessential data (for example, the field-4 area reserved in the initialization data by length specification of 10 bytes, 20 digits).

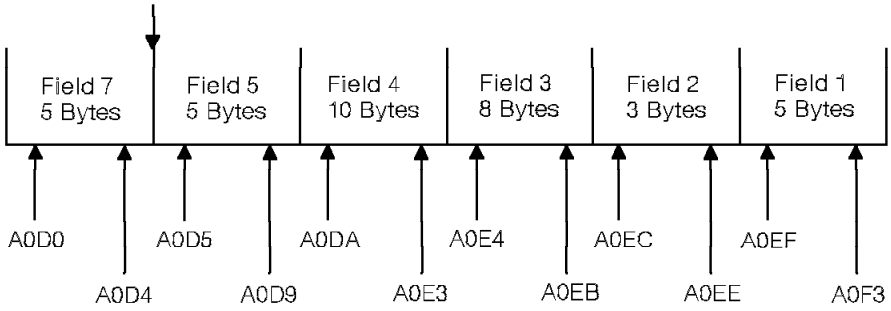
Where:

FLD1 = 5 bytes
FLD2 = 3 bytes
FLD3 = 8 bytes
FLD4 = 10 bytes
FLD5 = 5 bytes
FLD6 = 0 bytes
FLD7 = 5 bytes

```
LAS 10, X'30FF'
SAS 10, X'A0E3'
```

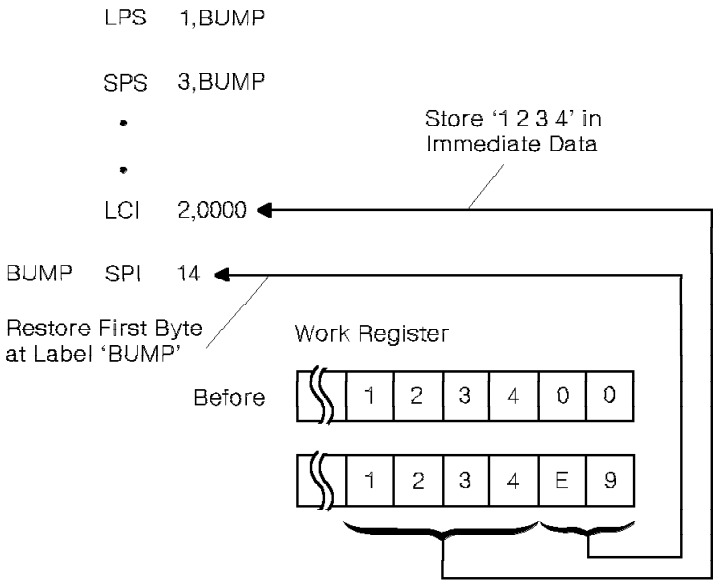


Using field-length specifications given in the preceding example, the process buffer (auxiliary storage locations X'A0D0' through X'A0F3') layout is:



Field data in the process buffer can be replaced with data in the work register using related addresses.

Change immediate data in an existing instruction based on control document data.



SPTX (Select MP by Table Index)

		(EF dl aa aa)
SPTX		**(D9 dl aa aa aa aa)**
[Symbol]	SPTX	(d, l), label

The **SPTX** (select-pocket-by-table-index) macro uses the contents of the work register specified by **d** and **l** (length in digits) as an index into a table to obtain the MP code. The accessed work register digits (half-bytes) are added to the address of the table at **label**. The accessed byte in the table becomes the MP code. The maximum length is four work register positions. The STBL macro creates the table.

The SPTX macro supports 2-byte addresses or 4-byte addresses. If you use the 4-byte format, set the high-order byte of the address to zero.

For an example of both the SPTX and STBL (pocket table) macros in a sample sort, see “Base-Number Sorting Techniques” on page 2-7.

Resulting Condition Code: Not changed.

SCI Error:

Address not valid—created address is above the end of program storage.

Length not valid—length specified is not 1-4.

Note: The data in the work register must be in hexadecimal form. If it is not, you can use the CV (convert-base) macro to convert the data to base 16 before you issue the SPTX macro.

You must guarantee indexing within the table; for example, the accessed work register digits must not point outside the limits of the table.

SS (Subtract Decimal Data)

SS		(5D dl aa aa) (DA dl aa aa aa aa)
[Symbol]	SS	(d, l), label

The **SS** (subtract-decimal-data) macro causes the decimal data at **label** to subtract from the work register digits specified by **d** for a length of **l** digits. The difference replaces those positions specified in the work register. Both areas are processed from right to left and treat all data as unsigned, packed decimal. If the digits at **label** are more in value than those in the specified area of the work register, the result is in ten's complement form.

Note: Any reference to the term "plus" or "minus" or the symbol "+" or "-" is for explanation only. No sign is available in any byte in any 3890/XP storage location.

The SS macro supports 2-byte addresses or 4-byte addresses. If you use the 4-byte format, set the high-order byte of the address to zero.

Resulting Condition Code:

0 Result 0 (true form)
1 Result minus (ten's complement form)
2 Result plus (true form)
3 Result not valid (decimal digit that is not valid was encountered).

SCI Error:

Address not valid—part or all of the specified data is at an address above the end of program storage.

Note: A decimal digit that is not valid does not end the operation. The result of subtractions is predictable, however, if you want to analyze your results. To determine the result when a decimal digit that is not valid is involved in the operation, a description of the 3890/XP system unit's subtract routine follows. For each digit, the one's complement of the second operand **label** (plus any carry from a previous position) is added to the digit in the corresponding position of the work register. If the result of this add is a not a carry, 1010 (X'A') is also added.

Examples:

Work Register Digit = 7 = 0111
Digit at **label** = 4 = 0100 - converted to one's complement = 1011

The carry is always on for the first position of subtraction; therefore, one is added to the converted digit (1011+1=1100).

```
      Add  0111
           1100
    Carry  0011
```

In this example, there is a carry to the next position with no more computation necessary. The answer is 3 (0011).

Work Register Digit = 7 = 0111
Digit at **label** = 9 = 1001 - converted to one's complement = 0110

The carry is always on for the first position of subtraction; therefore, one is added to the converted digit (0110+1=0111).

```
       Add  0111
            0111
 No Carry   1100
```

In this example there is no carry; therefore, 1010(A) is also added.

```
       Add  1110
            1010
            1000
```

The result is 8, but this was both the first and last position of subtraction, and a no-carry indicates and returns a condition code of 1—result minus (ten's complement form). Converting the 8 to its ten's complement form equals -2.

STA (Store into Save Area from Work Register)

STA ***(4A dl 0s ss)***

[Symbol]	STA	(d, l), s

The **STA** (store-into-save-area-from-work-register) macro places the contents of the work register specified by **d** and **l** (length in digits) into the specified location in the save area. Both areas are processed from right to left, and the unused positions of the save area and the complete work register are not changed.

Resulting Condition Code: Not changed.

SCI Error: Save area location not valid—**s** is odd or more than 510.

Note: The complete save area consists of 512 digits, with location 0 being the low-order position and 511 the high-order position. This area can be divided into as many separate save area sections as desired. For example, this area can be divided into thirty-two 16-digit save area sections. However, you have the responsibility to apportion the area and control the use of each section.

STBL (Pocket Table)

STBL

[Symbol]	STBL	a, b, c, ...

The **STBL** (pocket table) macro creates a table for use with the SPTX (select-pocket-by-table-index) macro. The accessed byte in the table becomes the MP code. Acceptable codes are 11 to 66.

The pocket table has the following format:

Byte 0 MP code for index 0

Byte 1 MP code for index 1

...

Byte n MP code for index n.

The parameters for the STBL macro have the following meanings:

a,b,c,...
Parameters separated by commas. Each parameter specifies an MP code.

For an example of both the STBL and SPTX (select-pocket-by-table-index) macros in a sample sort, see “Base-Number Sorting Techniques” on page 2-7.

THI (Test Header Under Immediate Mask)

THI ***(6h ii)***

[Symbol]	THI	h, i

The **THI** (test-header-immediate) macro permits an immediate mask with a decimal range from 0 to 255 or a hexadecimal range from X'00' to X'FF' to test the corresponding bits in the header byte. When the mask bit is one, the corresponding bit in the header byte is tested. **h** can be any one of the header bytes 0-11.

Resulting Condition Code:

0 Selected bits all zeroes (or mask bits all zeroes)
1 Selected bits mixed
2 -
3 Selected bits all ones.

SCI Error: Header byte not valid—header byte specified is not 0-11.

Example:

```
THI  8,X'05"     Test for high-order zero.
BCS  1,HOZRT     If on, branch to high-order-zero correction
                 routine.
THI  8,X'01'     Test for symbol error correction.
BCS  0,SECRT     If on, branch to symbol-error-correction
                 routine.
```

For another example, see the SPC (select-pocket-by-counter) macro on page 2-58.

TI (Test Work Register Immediate)

TI ***(46 dl ii ii)***

[Symbol]	TI	(d, l), i

The **TI** (test-work-register-immediate) macro logically tests the work register digits specified by **d** and **l** against the immediate data of up to four digits. Both areas are processed from right to left and treat all data as a 4-, 8-, 12-, or 16-bit, binary number with a decimal range from 0 to 65 535 or a hexadecimal range from X'0000' to X'FFFF'.

When the mask bit is one, the corresponding bit in the work register is tested.

Resulting Condition Code:

0 Selected bits all zeroes (or mask bits all zeroes)
1 Selected bits mixed
2 -
3 Selected bits all ones.

SCI Error: Length not valid—length specified is not 1-4.

TSI (Test User Stats Immediate)

TSI ***(AD ii)***

[Symbol]	TSI	i

The **TSI** (test-user-stats-immediate) macro permits the byte of immediate data with a value decimal range from 0-255 or X'00-FF' to serve as an immediate mask to test the corresponding bits in the stats byte.

Resulting Condition Code:

0 Selected bits all zeroes (or mask bits all zeroes)
1 Selected bits mixed
2 -
3 Selected bits all ones.

SCI Error: None.

Example:

Using the user stats byte as a first-time switch:

```
      .
      .
     TSI X'20'     See if switch is on.
     BCS 8,INIT    When it is 0, go to INIT.
      .
      .
      .
INIT OSI X'20'     Turn on first-time switch.
```

V (Verify)

V **(5E dl aa aa)**

[Symbol]	V	(d, l), label

The **V** (verify) macro tests the work register digits specified by **d** and **l** (length in digits) for validity as described by the rules set up in the table at **label**.
The VTBL macro creates the table. For an example of the combined use of the V and VTBL (verification table) macros, see “Number Self-Checking” on page 2-49.

Any number of unused digits and/or dashes (SS4s) can be interspersed, but these positions must be assigned a weighting factor of zero in the corresponding positions of the table. These positions must also be included in the **l** (length) parameter.

The V macro supports 2-byte addresses or 4-byte addresses. If you use the 4-byte format, set the high-order byte of the address to zero.

Resulting Condition Code:

0 Verified. (The residual from the verification algorithm is saved in 3890-emulated auxiliary storage.)
1 Verified and short field (fill character encountered before **d** and **l** satisfied).
2 Not verified.
3 Decimal digit not valid.

SCI Error:

Table data not valid—a weighting factor higher than X'0B' (decimal 11).

Address not valid—part or all of the specified data is at an address above the end of program storage.

VTBL (Verification Table)

VTBL

[Symbol]	VTBL	MOD=y, REM=n, WTS=(a,b,c,...p), OPT=1∨2

The **VTBL** (verification table) macro creates a table for use with the V (verify) macro. This table contains the rules by which the self-checking arithmetic is performed.

The parameters for the VTBL macro have the following meanings:

MOD=y
Indicates the modulus to be used, with a range from 00 to 99. The default value is 10.

REM=n
The constant remainder used in calculating the modulus number. REM is an unsigned decimal number in the range 00 to 99.

Note: The remainder specified must always be less than the modulus indicated.

WTS=(a,b,c,...,p)
The weighting factor consists of from 1 to 16 self-defining terms with a value range of 0 to 11. Omitted operands are assigned a value of zero. The **a** applies to the high-order position, **b** applies to the next position to the right, and so forth.

A dash (SS4) or any other special symbol should not participate in the verification; therefore, the corresponding weighting factor should be zero.

OPT=1∨2

1 Each product (field digit times the weighting factor) is treated as two single digits to be added together, as in modulus 10.

2 Each product is treated as a single value, as in modulus 11.

For an example of the combined use of the VTBL and V (verify) macros, see “Number Self-Checking” on page 2-49.

The verification table has the following format:

Byte 0 Option (X'80'=1, X'00'=2)

Byte 1 Modulus (packed decimal)

Byte 2 Remainder (packed decimal)

Byte 3-18 Weighting factors (binary).

Note: Table data must start on an even-address boundary.

SCI Macro Messages

Figure 2-5 lists messages for the MNOTES for the ENTR and SCI macros. The severity codes listed for these MNOTES have the following meanings:

Severity Code	Meaning
* or 0	Information only.
4	Minor errors detected; successful program run is probable.
8	Errors detected; unsuccessful program run is possible.
12	Serious errors detected; unsuccessful program run is probable.

MVS message IDs begin with EFA. The following messages are the same for VSE except that VSE message IDs begin with EFB.

Figure 2-5 (Page 1 of 3). SCI Assembly Time Error Messages

Msg. ID	Message	Recommended Action	Sev. Code
EFA349	DISPLACEMENT NOT 0-{14|15}, 0 ASSUMED	Correct the displacement operand for the SCI and submit the job again.	8
EFA350	LENGTH NOT 1-16, 1 ASSUMED	Correct the length operand for the SCI and submit the job again.	8
EFA351	LENGTH NOT 1-4, 1 ASSUMED	Correct the length operand for the SCI and submit the job again.	8
EFA352	FIELD NOT 0-8, 0 ASSUMED	Specify the correct field (0-8) and submit the job again.	8
EFA353	SAVE AREA ADDRESS IS NOT EVEN OR IS GREATER THAN 510, 0 ASSUMED	Specify the correct save-area storage value and submit the job again.	8
EFA354	HEADER BYTE NOT 0-11, 0 ASSUMED	Correct the header operand for the SCI and submit the job again.	8
EFA355	LINK REGISTER NOT 0-3, 0 ASSUMED	Specify correct link register and submit the job again.	8
EFA356	MODULE POCKET CODE INVALID, 11 ASSUMED	Module pocket code must be 11-16, 21-26, 31-36, 41-46, 51-56, and 61-66, (00 allowed for SPI). Specify the correct module pocket code and submit the job again.	12
EFA357	OPT NOT 1 OR 2, 1 ASSUMED	Specify the correct option and submit the job again.	8
EFA359	MODULUS INVALID OR NOT SPECIFIED, 10 ASSUMED	Specify the correct modulus and submit the job again.	8
EFA360	WEIGHT FACTOR GREATER THAN 11, 0 ASSUMED	Specify a valid weighting factor and submit the job again.	8
EFA361	ERROR IN OPERAND NUMBER N, 0 ASSUMED	Specify the correct operand and submit the job again.	8
EFA362	CONVERSION MUST BE FOR BASES 10-255, MACRO IGNORED	Specify the correct base value and submit the job again.	12
EFA403	HEADER BYTE NOT 2, 3, 5, 6, or 9; 2 ASSUMED	Specify the correct header byte and submit the job again.	8

Figure 2-5 (Page 2 of 3). SCI Assembly Time Error Messages

Msg. ID	Message	Recommended Action	Sev. Code
EFA404	VALUE NOT 0-4, 0 ASSUMED	Specify an immediate value of 0-4 and submit the job again.	8
EFA405	MACRO CONTAINS INCONSISTENT OPERANDS, MACRO IGNORED	Check that the correct ENTR macro operands have been coded for the SCI program or subroutine. For example, the PGMLEN operand should not be coded for an SCI program ENTR macro. Specify the correct operands and submit the job again.	12
EFA412	SCIERR NOT SPECIFIED, 0 ASSUMED	Specify the SCIERR operand and submit the job again.	12
EFA413	PGMLEN OPERAND INVALID, MACRO IGNORED	Specify length in hexadecimal or decimal and submit the job again.	12
EFA416	COUNTER NOT 0-15, 0 ASSUMED	Correct the counter value and submit the job again.	8
EFA421	IMMEDIATE VALUE EXCEEDS ALLOWED MAXIMUM	Correct the immediate value specified and submit the job again.	8
EFA424	240 DWF's ALREADY SPECIFIED	Do not equate displacement and submit the job again.	12
EFA425	D AND L NOT EQUATED BY DWF MACRO, (0,1) ASSUMED	Use DWF macro to equate displacement and length and submit the job again.	8
EFA426	DISPLACEMENT NOT EVEN, ROUNDED TO n	Recode macro with even displacement and submit the job again.	8
EFA427	ENTRY LENGTH GREATER THAN 32 DIGITS, TRUNCATED	Correct entry length and submit the job again.	8
EFA430	ENTRY OUT OF SEQUENCE	Table will be built. If table entry sequence is not desired, change sequence and submit the job again.	8
EFA432	PARAMETERS ARE INVALID, (0,1) ASSUMED	Correct the displacement and length operand and submit the job again.	12
EFA433	INVALID DISPLACEMENT (D,L), (0,1) ASSUMED	Correct the displacement and length operand, and submit the job again.	8
EFA434	INVALID DISPLACEMENT, 0 ASSUMED	Correct the displacement operand and submit the job again.	8
EFA436	BASE BEYOND 36, NOT VALID SORT–SPECIAL USE ONLY	Table will be built. If table is to be used for a base-conversion sort, specify a base of 10-36 and submit the job again.	4
EFA437	SKIP PARM OPERAND n NOT SELF–DEFINING, IGNORED	Correct and submit the job again.	8
EFA438	COMPARE LENGTH NOT EVEN, ROUNDED TO n	Correct compare length if necessary and restart job.	4
EFA439	COMPARE LENGTH NOT 2-32, MACRO IGNORED	Correct compare length and restart.	12
EFA440	NO TABLE ENTRIES	Specify table entries and submit the job again.	0

Figure 2-5 (Page 3 of 3). SCI Assembly Time Error Messages

Msg. ID	Message	Recommended Action	Sev. Code
EFA441	MAX ENTRY LEN LESS THAN COMPARE LEN, ENTRIES PADDED TO COMPARE LENGTH	Correct compare length if necessary.	4
EFA442	ENTRY INVALID, 0 ASSUMED	Correct table entry and submit the job again.	8
EFA455	TYPE=SCI4 INVALID UNLESS EXT=Y, TYPE=SCI2 ASSUMED	Specify EXT=Y or specify TYPE=SCI2.	8
EFA456	FIELD VALUE BEYOND RANGE, IGNORED	Specify field value between 1-8 if EXT=N or 1-15 if EXT=Y.	12
EFA458	NO FIELD VALUES SPECIFIED	Specify the desired field operands on the IMAT macro and submit the job again.	4
EFA459	FIELD NOT 0-15, 0 ASSUMED	Correct the LF, LFE or LSE field operand and submit the job again.	8
EFA460	CALLNATV INVALID UNLESS EXT=Y SPECIFIED ON ENTR MACRO	Specify EXT=Y on the ENTR macro and submit the job again.	8
EFA461	CALLNATV INVALID - NATIVE ROUTINE NOT SPECIFIED	Specify the high-level language routine operand for the CALLNATV macro.	8
EFA483	TYPE VALUE MUST BE SCI2 OR SCI4, SCI4 ASSUMED	Correct the TYPE operand and submit the job again.	8
EFA484	EXT VALUE MUST BE Y OR N, Y ASSUMED	Correct the EXT operand and submit the job again.	8
EFA485	TYPE=SCI4 SPECIFIED, EXT=Y ASSUMED	Informational message.	*

Chapter 3. Stacker Control Instruction Codes

This chapter lists and explains Stacker Control Instructions, SCI condition codes, and SCI execution-time formulas.

SCIs

Figure 3-1 (Page 1 of 2). Stacker Control Instructions

Name	Mnemonic	Operands	Machine Format
Add Counter Immediate	ACI	c,i	AA c0 ii ii
AND Header Immediate	NHI	h,i	7h ii
AND User Stats Immediate	NSI	i	AE ii
Branch and Link	BLS	r,label	BD r0 aa aa D1 r0 aa aa aa aa
Branch on Condition	BCS	m,label	BC m0 aa aa D0 m0 aa aa aa aa
Branch on Register	BUR	r	BE r0
Call Native	CALLNATV	label	F8 00 aa aa aa aa
Compare and Select	CST	(d,l) label	5C dl aa aa D2 dl aa aa aa aa
Compare Counter Immediate	CCI	c,i	AC c0 ii ii
Compare Header Immediate	CHI	h,i	8h ii
Compare Work Register Immediate	CI	(d,l),i	48 dl ii ii
Compare Work Register to Save Area	CA	(d,l),s	4B dl 0s ss
Compare Work Register to Table	CT	(d,l),label	5B dl aa aa D3 dl aa aa aa aa
Convert Base	CV	(d,l),label	5F dl aa aa D4 dl aa aa aa aa
Disengage	DSG		FA 00
Disengage and Set Operator Message Attention	DSGA		FB 00
Execute	EXC	d,label	F9 d0 aa aa D5 d0 aa aa aa aa
Exit	EXT		FF 00
Extended Code Line Data Match	IMAT	a,b,c,...	C3 ii C5 00 ii ii
Feed from Merge	FM		FC 00
Initialize Feature Data	IFD	i	FD i0
Insert from Save Area into Work Register	IA	(d,l),s	4C dl 0s ss
Insert into Work Register Immediate	II	(d,l),i	47 dl ii ii
Jump on Table Index	JX	d,label	BF d0 aa aa D6 d0 aa aa aa aa
Load Counter Immediate	LCI	c,i	AB c0 ii ii

Figure 3-1 (Page 2 of 2). Stacker Control Instructions

Name	Mnemonic	Operands	Machine Format
Load Work Register from Auxiliary Storage	LAS	l,a	C1 0l aa aa D7 0l aa aa aa aa
Load Work Register from Field	LF	(d,l),f	3f dl
Load Work Register from First Element	LFE	(d,l),f	1f dl
Load Work Register from Program Storage	LPS	l,a	C1 1l aa aa D7 1l aa aa aa aa
Load Work Register from Program Switches	LSW		44 00
Load Work Register from Second Element	LSE	(d,l),f	2f dl
OR User Stats Immediate	OSI	i	AF ii
OR Header Immediate	OHI	h,i	9h ii
Search Binary	SBT	d,label	5A d0 aa aa D8 d0 aa aa aa aa
Select FC Immediate	SFI	i	EA ii
Select MP by Counter	SPC		EC 00
Select FC/MP by Header	SPH		EE 00
Select MP by Table Index	SPTX	(d,l),label	EF dl aa aa D9 dl aa aa aa aa
Select MP Immediate	SPI	mp	E9 mp
Set Operator Message from Work Register	SOM		45 00
Store Work Register into Auxiliary Storage	SAS	l,a	C1 2l aa aa D7 2l aa aa aa aa
Store Work Register into Program Storage	SPS	l,a	C1 3l aa aa D7 3l aa aa aa aa
Store into Save Area from Work Register	STA	(d,l),s	4A dl 0s ss
Subtract Decimal Data	SS	(d,l),label	5D dl aa aa DA dl aa aa aa aa
Subtract from Work Register Immediate	SI	(d,l),i	49 dl ii ii
Test Header under Immediate Mask	THI	h,i	6h ii
Test User Stats Immediate	TSI	i	AD ii
Test Work Register Immediate	TI	(d,l),i	46 dl ii ii
Verify	V	(d,l),label	5E dl aa aa DA dl aa aa aa aa

Machine Format Legend

aa aa	Two-byte address in program storage or auxiliary storage
aa aa aa aa	Four-byte address in program storage or auxiliary storage
c	Counter
d	Displacement
f	Field
h	Header
i	Immediate
l	Length in digits or bytes
m	Mask
mp	Module pocket
r	Link register
s ss	Save area
00	Zeroes

SCI Condition Codes

Figure 3-2 (Page 1 of 2). Condition Codes

Condition Code Setting **Mask Bit Position**	**0** **8**	**1** **4**	**2** **2**	**3** **1**
Add Counter Immediate	zero no carry	not zero no carry	zero carry	not zero carry
AND Header Immediate	zero	not zero	--	--
AND User Stats Immediate	zero	not zero	--	--
Call Native	good	--	not zero	--
Compare and Select	equal	not equal	--	--
Compare Counter Immediate	equal	ctr < imm	ctr > imm	--
Compare Header Immediate	equal	hdr < imm	hdr > imm	--
Compare Work Register Immediate	equal	wr < imm	wr > imm	--
Compare Work Register to Save Area	equal	wr < area	wr > area	--
Compare Work Register to Table	equal	not equal	--	--
Convert Base	converted	converted, short field	--	decimal digit not valid
Feed from Merge	merge doc requested	merge request ignored	--	--
Extended Code Line Data Match	code line data match	code line data not found	field not valid	previous match
Load Work Register from Field	field loaded	short length field	d+l > field limit	field not processed
Load Work Register from First Element	element loaded	short length element	d+l > field limit	field not processed
Load Work Register from Second Element	element loaded	short length element	d+l > field limit	field not processed
OR Header Immediate	zero	not zero	--	--
OR User Stats Immediate	zero	not zero	--	--
Search Binary	equal entry found	range found	entry not in list	d+l > work reg
Select MP by Counter	MP set for full counter	MP set to 1/1, counters low	--	--
Select FC/MP by Header	code line data found	no code line data	code line data matching not active	--
Subtract Decimal Data	zero	10’s comp form	true form	decimal digit not valid
Subtract from Work Register Immediate	zero	10’s comp form	true form	decimal digit not valid

Figure 3-2 (Page 2 of 2). Condition Codes

Condition Code Setting **Mask Bit Position**	**0** **8**	**1** **4**	**2** **2**	**3** **1**
Test Header Under Immediate Mask	zero	mixed	--	one
Test User Stats Immediate	zero	mixed	--	one
Test Work Register Immediate	zero	mixed	--	one
Verify	verified	verified, short field	not verified	decimal digit not valid

SCI Execution-Time Formulas

The SCI execution-time formulas are accurate to plus or minus 10%. However, as with any data-dependent instruction, certain special cases might have timings outside of this tolerance.

The following SCI execution-time formulas are for the 3890/XP 16 MH Personal System/2 (PS/2) Model 80. The execution times are calculated in **microseconds**.

Figure 3-3 (Page 1 of 3). SCI Execution-Time Formulas

SCI	Machine Format	Execution-Time Formula	Comments
ACI	AA c0 ii ii	22	
BCS	BC m0 aa aa	22	
	D0 m0 aa aa aa aa	22	
BLS	BD r0 aa aa	23	
	D1 r0 aa aa aa aa	23	
BUR	BE r0	19	
CA	4B dl 0s ss	33 + 5.7*L	L = no. of digits
CALLNATV	F8 00 aa aa aa aa	40 + T	T = Native routine's processing time
CCI	AC c0 ii ii	18	
CHI	8h ii	20	
CI	48 dl ii ii	38	
CST	5C dl aa aa	57 + 2L + 14N	L = no. of digits N = no. of compare loops performed
CT	5B dl aa aa	40 + 2L + 15N	L = no. of digits N = no. of compare loops performed
	D3 dl aa aa aa aa	40 + 2L + 15N	L = no. of digits N = no. of compare loops performed
CV	5F dl aa aa	71 + 2.2L	L = total no. of bytes in CVTBL group-entry line lengths
	D4 dl aa aa aa aa	71 + 2.2N	L = total no. of bytes in CVTBL group-entry line lengths
DSG	FA 00	14	
DSGA	FB 00	12	
EXC	F9 d0 aa aa	35	
	D5 d0 aa aa aa aa	35	
EXT	FF 00	13	
FM	FC 00	14	
IA	4C dl 0s ss	30 + 4.7*L	L = no. of digits
IFD	FD i0	22	
II	47 dl ii ii	35	

Figure 3-3 (Page 2 of 3). SCI Execution-Time Formulas

SCI	Machine Format	Execution-Time Formula	Comments
IMAT	C3 ii	60 + IPRT time	
	C5 00 ii ii	60 + IPRT time	
JX	BF d0 aa aa	28	
	D6 d0 aa aa aa aa	28	
LAS	C1 0l aa aa	38 + 0.5L + 6T	L = no. of bytes T = no. of table entries searched
	D7 0l aa aa aa aa	38 + 0.5L + 6T	L = no. of bytes T = no. of table entries searched
LCI	AB c0 ii ii	15	
LF	3f dl	46 + 7*L	L = no. of digits moved
LFE	1f dl	46 + 7*L	L = no. of digits moved
LPS	C1 1l aa aa	38 + 0.5L	L = no. of bytes
	D7 1l aa aa aa aa	45 + 0.5L	L = no. of bytes
LSE	2f dl	46 + 7*L	L = no. of digits moved
LSW	44 00	36	
NHI	7h ii	20	
NSI	AE ii	16	
OHI	9h ii	18	
OSI	AF ii	16	
SAS	C1 2l aa aa	64 + 0.5L + 6T	L = no. of bytes T = no. of table entries searched
	D7 2l aa aa aa aa	64 + 0.5L + 6T	L = no. of bytes T = no. of table entries searched
SBT	5A d0 aa aa	85 + 32*N 84 if <= first entry 65 if >= last entry	N = no. of compare loops performed
	D8 d0 aa aa aa aa	134 + 56*N 132 if <= first entry 95 if >= last entry	N = no. of compare loops performed
SFI	EA ii	14	
SI	49 dl ii ii	28 + 31.6*L	L = no. of digits
SOM	45 00	35	
SPC	EC 00	18 + 5N	N = no. of pockets scanned
SPH	EE 00	12	
SPI	E9 mp	14	
SPS	C1 3l aa aa	38 + 0.5L	L = no. of bytes
SPS	D7 3l aa aa aa aa	45 + 0.5L	L = no. of bytes
SPTX	EF dl aa aa	36	
SS	5D dl aa aa	28 + 31.6L	L = no. of digits
	DA dl aa aa aa aa	28 + 31.6L	L = no. of digits

Figure 3-3 (Page 3 of 3). SCI Execution-Time Formulas

SCI	Machine Format	Execution-Time Formula	Comments
STA	4A dl 0s ss	27 + 4.6*L	L = no. of digits
THI	6h ii	20	
TI	46 dl ii ii	38	
TSI	AD ii	15	
V	5E dl aa aa	131 + 16L	L = no. of bytes

Glossary

This glossary defines terms that are used in this manual and in other books in the 3890/XP library. If you do not find the term that you want, see the index.

A

alternate INF data. A five-byte field in auxiliary storage that is printed on the back of an item or document instead of the INF number when the alternate INF indicator in auxiliary storage is set to X'01' (on). You must specify ten digits, two for each byte. Each half-byte has a value range of X'0-A'. X'A' prints as a blank.

C

CCW. See *channel command word*.

channel command word. (CCW). A System/370 word interpreted by the channel as an instruction in the channel program. One or more CCWs make up the channel program. That channel program directs channel operations.

code line data matching. The process of matching data in a file to the code line data on a document.

D

default. An alternative value, attribute, or option that is assumed when none has been specified.

disk subsystem. System facilities that are identified by OS/2 drive and path conventions.

display panel. A predefined display image that can contain information such as instructions, prompts, options, and data.

display screen. An illuminated display surface on which display images are presented.

DLL. See *dynamic link library*.

document processor application programming interface. (DPAPI). Macros and modules that supply support for the host application to control the operation of a 3890/XP series document processor and to supply error recovery and recording.

document processor control block. (DPCB). The primary parameter which describes the document processor being used by the application.

DPAPI. See *document processor application programming interface*.

DPCB. See *document processor control block*.

dynamic link library. (DLL). A specially-linked collection of native language subroutines, loaded by OS/2 facilities, that can be called during the running of native modules. The OS/2 loader automatically loads additional DLL files that are needed to resolve labels in the DLL file that is being loaded. The native language file named in the run profile is of this type.

E

ECP. See *endorse control product*.

EIM. See *extended image match*. Also see *extended code line data match*.

endorse control product. (ECP). This product aids the SCI programmer in controlling the endorse feature for each document.

extended code line data match. An operation that permits code line data matching to be done under the control of the SCI program.

extended image match. (EIM). See *extended code line data match*.

I

IBM format. The high-order byte of an integer value is stored in the low address of the field; therefore, the integer value in memory appears the same as its hexadecimal format.

```
Decimal format:      1234
Hexadecimal format:  04D2
IBM format:          04D2
```

image matching. A term used in the past to describe the process of matching data in a file to the code line data on a document. See *code line data matching*.

IML. See *initial microcode load*.

INF number. An eight- or ten-digit number that is printed on the back of each item or document that the 3890/XP processes if the alternate INF indicator in auxiliary storage is set to X'00' (off). The sort program can add one to either the low-order segment or the high-order segment of this number after each document is processed.

initial microcode load. (IML). This is the event that loads the 3890/XP control program. This event must occur after every power transition of the 3890/XP.

initialization data. The first 128 bytes of program storage. The initialization data defines the functions of the 3890/XP. It is extended by the run profile, which can be named in the initialization data.

initialization record macro. (IREC macro). A host system macro that can be used to format the initialization data.

Intel format. The low-order byte of an integer value is stored in the low address of the field. Therefore, the integer value in memory appears to be the reverse of its hexadecimal format.

```
Decimal format:      1234
Hexadecimal format:  04D2
Intel format:        D204
```

IREC macro. See *initialization-record macro*.

item number. See *INF number*.

K

K byte. See *kilobyte*.

kilobyte. (K byte). 1 K byte = 1024 bytes.

L

LAPI. See *LU 6.2 application program interface*.

LCL. See *level control label*.

level control label. (LCL). A 16-character string carried in every run profile element that can be used to check the change level of that element.

LU 6.2 application program interface. (LAPI). Macros and modules that supply support for the host application to communicate with other application programs using LU 6.2 protocol.

M

M byte. See *megabyte*.

machine profile. This file contains parameters describing the specific 3890/XP machine, such as the System/370 channel address. It is loaded with the 3890/XP control program from the disk subsystem at initialization. You can use the customization display panels in the 3890/XP user interface to maintain this file.

megabyte. (M byte). 1 M byte = 1 048 576 bytes.

message line. The area of the display screen where messages appear.

N

native language. Any language that is supported by the OS/2 system.

O

Operating System/2. (OS/2). The operating system for the Personal System/2 computers.

OS/2. See *Operating System/2*.

P

panel. See *display panel*.

program-storage-data file. (PSD). A file on the disk subsystem that is identified in the run profile that is loaded without change into program storage during initialization. PSD also refers to the name and path specification of this type of file. You use a PSD to load a table into the sorter; a run profile might include a PSD entry.

PSD. See *program-storage-data file*.

PSD table. Run-profile-element table. A table that the 3890/XP control program builds for the SCI program and native language sort programs. The table points to locations in storage of data that is loaded from files during initialization. Such files, identified in the run profile, are run profile elements.

PSD tag. A tag that is independent from the file name and that is carried in the PSD keyword of the run profile (PGMSTOREdata) and is available to the SCI and native language sort programs through the PSD table. Sort programs use this tag to identify and find PSD files by type.

Q

QSAM. See *queued sequential access method*.

queued sequential access method. (QSAM). An extended version of the basic sequential access method (BSAM). When this method is used, a queue is formed of input data blocks that are awaiting processing or of output data blocks that have been processed and are awaiting transfer to auxiliary storage or to an output device.

quick stop. The hardware-initiated event that stops the transport immediately when a jam or other problem

is detected. The operator clears the affected module, then uses a runout to permit trapped documents that are still in the transport to be processed. See *runout*.

R

run profile. An ASCII keyword file that controls operation during initialization. This file is available to the 3890/XP control program through the disk subsystem. If no run profile is named in the initialization data, the default run profile is used. The machine profile names the default run profile, the drive, and the path.

runout. The hardware-controlled operation that the operator uses during recovery from a quick stop. It permits items that were trapped by the quick stop but not removed by the operator to be processed. During runout, the SCI program runs and creates read records, even though all documents that are read during runout appear blank. A jam can also occur during runout.

S

SCI. See *Stacker Control Instruction*.

SIO. See *start I/O*.

Stacker Control Instruction. (SCI). SCI is a language that is interpreted by the 3890/XP as the sort program. 3890/XP offers major extensions, including four-byte addressing and native language subroutines.

sort program. All the user-supplied code that an SPX calls. In the run profile, the sort program for each SPX event is specified by an ordered series of SPS keywords. SPS keywords list sort program modules. One special sort program module is SCID, which runs the one SCI program that is stored in the one program store area. Like any sort program module, SCID can be specified in the sort program for any of the SPX events. Once it is called, a sort program module can call other sort program modules.

sort program execution. (SPX). The 3890/XP control program performs SPX by calling the native and SCI routines that are specified by the SPS in the run profile when specific SPX events occur.

sort program sequence. (SPS). Sort program sequence keyword in the run profile. This keyword specifies the components of the sort program for a specific SPX. This includes the native language subroutine (if any),
the SCI program (if used), and the sequence of their processing. A run profile can contain an SPS keyword for each SPX.

special-symbol-sequence table. This table defines the code line for 3890/XP documents.

SPS. See *sort program sequence*.

SPX. See *sort program execution*.

SPX event. An event that causes the sort program to run. SPX events include the following:

- Non-document SPX events:
  - Initialize
  - Motors stopped
  - Operator
- Document SPX events:
  - Document - A document is read.
    The document sort program is specified by the following sequence of SPS keywords:
    Verify
    Correction
    Format
    Code line data match
    Document
    Post sort.

SPXSERV.DLL. See *SPX services dynamic link library*.

SPX services dynamic link library. (SPXSERV.DLL). This library comes with the 3890/XP control program. These subroutines are the lowest-level interface for the native language programmer to 3890/XP services.

SST. See special-symbol-sequence table.

start I/O. (SIO) The System/370 instruction that starts all I/O operations by requesting a channel program.

Index

J

L

M

N

O

P

S

T

U

V

W

Communicating Your Comments to IBM

3890/XP Series
Stacker Control Instructions Reference

Publication No. SC31-2703-00

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:

  United States & Canada: 1-800-955-5259
- If you prefer to send comments electronically, use this network ID:

  IBM Mail Exchange: USIB1362 at IBMMAIL

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies.

Readers' Comments — We'd Like to Hear from You

3890/XP Series
Stacker Control Instructions Reference

Publication No. SC31-2703-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	□	□	□	□	□

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	□	□	□	□	□
Complete	□	□	□	□	□
Easy to find	□	□	□	□	□
Easy to understand	□	□	□	□	□
Well organized	□	□	□	□	□
Applicable to your tasks	□	□	□	□	□

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? □ Yes □ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SC31-2703-00

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Payment Solutions
Department 58G MG34/204
8501 IBM Drive
Charlotte NC 28262-8563

Fold and Tape | **Please do not staple** | Fold and Tape

SC31-2703-00

Cut or Fold
Along Line

File Number: S/370-4300-40

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC31-2703-00