
Appendix B: Simple I/O Macros

```
IIIIIIIII // 000000000000
IIIIIIIII // 000000000000
      II // 00      00
      II // 00      00
      II // 00      00
      II // 00      00
      II // 00      00
      II // 00      00
      II // 00      00
      II // 00      00
      II // 00      00
IIIIIIIII // 000000000000
IIIIIIIII // 000000000000
```

Simple I/O Macros

1

- We use some simple macro instructions for input, output, conversion, and display
 - CONVERTI converts decimal characters in memory to 32- or 64-bit binary integers in a general register
 - CONVERTO converts 32- or 64-bit binary integers in a general register to decimal characters, or contents of a floating-point register to hexadecimal characters, in memory
 - DUMPOUT displays the contents of storage in hexadecimal and character formats
 - PRINTLIN sends a string of characters to a printer file
 - PRINTOUT displays the contents of registers and of named areas of memory, and/or terminates execution
 - READCARD reads an 80-byte record from an input file to a specified area of memory
- Each macro calls an entry point in an automatically generated control section

The forms that I/O takes within and across z/Architecture operating systems are varied enough that it takes many books to describe them. So that you won't need to understand the I/O rules associated with a particular Operating System, the simple needs of small programs like the Programming Problems here can be satisfied by the following facilities:

1. An instruction (CONVERTI) to convert decimal characters in memory to a signed binary value in a General Purpose register.
2. An instruction (CONVERTO) to convert the contents of a General Purpose register to a string of decimal characters in memory.
3. An instruction (DUMPOUT) to generate a formatted hexadecimal dump of specified areas of memory.
4. An instruction (PRINTLIN) to print line images on a printer file, with carriage control characters and optional specification of the length of the character string to be printed.
5. An instruction (PRINTOUT) to:
 - print the value of the contents of an area of memory, giving its name as well as the value of the contents in an easily-readable format.
 - print the contents of the General Purpose and Floating-Point registers.
 - return control to the Supervisor when program execution has been successfully completed.
6. An instruction (READCARD) to read 80-character records into a named area in your program, with provision for optionally transferring control to some out-of-line location if no further records are available.

These macro instructions do not change the Condition Code.

Notation and Terminology Conventions

2

- The macro descriptions use these terms:

<name>	a symbol naming an area of memory addressable from the macro
<number>	a self-defining term (or a predefined absolute symbol) with value limits specified by the macro
<d(b)>	specifies an addressable base-displacement operand
<address>	specifies a <name> or <d(b)>
<nfs>	an optional name-field symbol on a macro
[item]	[] indicates an optional item
...	indicates that the preceding item may be repeated
- Referring to registers:
 - Numbers 0-15 refer to 32-bit general registers 0-15
 - Numbers 16-31 refer to 64-bit general registers 0-15
 - Numbers 32-47 refer to Floating-Point registers 0-15

B.1. Macro Facilities

The arguments of the CONVERTI, CONVERTO, DUMPOUT, PRINTLIN, PRINTOUT, and READCARD macro-instructions use these notational abbreviations in their descriptions.

<name>	any valid symbol naming an area of memory which is addressable from the point where it is used in a macro-instruction.
<number>	any valid self-defining term; limits on the size of the term are described for each macro-instruction. In most cases, a predefined absolute symbol may be used.
<d(b)>	any valid operand providing an <i>addressable</i> displacement and base.
<address>	specifies a <name> or <d(b)>
<nfs>	a valid (and optional) name-field symbol (label) naming the macro-instruction in whose name field it appears.
[optional]	square brackets around a term means that it is optional.
...	an ellipsis means that the preceding item may be repeated any number of times.

Examples of these macro instructions are given below.

You may want to precede the first call to any of these macros with a

PRINT NOGEN

statement; otherwise your listing will include many generated statements that won't have much meaning to you until you've had more experience with Assembler Language and System z.

The CONVERTI Macro Instruction

3

- CONVERTI is written

```
<nfs>    CONVERTI  <number>,<address>[,ERR=<address>][,STOP=<address>]
```

- The digits starting at the second operand <address> are converted to binary in the general register designated by <number>
 - The first non-blank character must be +, -, or a decimal digit; if not, GR1 is set to the address of the invalid character
- ERR= specifies an address to receive control for an invalid <number> or a too-large converted value
- STOP= specifies an address to receive control for an invalid character in the input
- If either condition occurs and neither ERR= or STOP= is specified, the program terminates with an error message.

- Example:

```
          CONVERTI  3,Data    c(GR3) = X'00000013', c(GR1) = A(Data+3)  ('?')
          ---
Data      DC      C'+019?'    Input string
```

B.1.1. The CONVERTI Macro Instruction

CONVERTI is generally used to scan a data record received by the READCARD macro instruction. It converts a string of optionally signed decimal characters to binary into a specified 32-bit or 64-bit general register, and sets GR1 to the address of the non-digit character at which scanning was stopped. This means that you can convert multiple values from the same character string.

The CONVERTI macro instruction is written in the form

```
<nfs>    CONVERTI  <number>,<address>[,ERR=<address>][,STOP=<address>]
```

where <number> specifies a general register, and <address> is the starting address of a string of bytes in storage of characters to be converted to binary. The <address> operand may also be written as <d(b)>.

The first non-blank character must be a plus sign, a minus sign, or a decimal digit; otherwise, GR1 is set to the address of that character and the register specified by <number> is unchanged. (If you expect unusual characters to be scanned, you should specify the STOP= operand on the CONVERTI statement.)

Be Careful!

Don't specify either 1 or 17 for the <number> operand, because any converted value in GR1 or GG1 will be replaced by the address of the "stop" character.

The optional keyword operands ERR= and STOP= specify locations in your program where the CONVERTI will transfer control if certain conditions occur:

- ERR=** If the value of the <number> operand is greater than 31, or if the value of the significant decimal digits at the <address> operand is too large to be converted correctly to the general register specified by the <number> operand, control will be transferred to the <address> given by the ERR= operand.
- STOP=** If an invalid character is found in the string of characters starting at the <address> operand, GR1 is set to the address of that character and control will be transferred to the <address> given by the STOP= operand. (See example 3 below.)

If either of these conditions occurs and the needed ERR= or STOP= <address> is not specified, CONVERTI will print a message and terminate the program.

32-bit general register

A <number> between 0 and 15 specifies the corresponding 32-bit general register. The number to be converted must have no more than 10 significant digits. Insignificant leading zero digits are ignored.

64-bit general register

A <number> between 16 and 31 specifies the corresponding (less 16) 64-bit general register. The number to be converted must have no more than 19 significant digits. Insignificant leading zero digits are ignored.

Execution on z/Architecture is required.

For example:

1. Convert the digits at **D1** to binary in 32-bit GR3:

```
CONVERTI 3,D1
- - -
D1      DC    C' +019 '
```

The contents of GR3 will be X'00000013' and GR1 will contain the address of the blank character following the digit 9.

2. Convert the digits at **D2** to binary in 64-bit GG5:

```
CONVERTI 21,D2
- - -
D2      DC    C' -9223372036854775808 '
```

The contents of GG5 will be X'8000000 00000000' and GR1 will contain the address of the blank character following the final digit 8.

3. The character string at **D3** contains several values to be stored at **Table**. The scan is terminated by the character *.

	LA	3,Table	
	LA	2,D3	Start of string
CvtLoop	CONVERTI	0,0(2),STOP=Check	Invalid character? Test at Check
	ST	0,0(,3)	Store an entry at Table
	LA	3,4(,3)	Point to next Table entry
	B	CvtLoop	Resume converting
	-	-	
Check	CLI	0(1),C'*	Is the invalid character ours?
	JNE	BadChar	No, a bad character appeared
	-	-	
D3	DC	C' +1 -2+3 -0000000000000004 *	

and the four words starting at **Table** will contain the values 1, -2, 3, and -4.

Providing a known “stop” character lets you scan an input string for all the values provided. For example, a record stored at **InRec** by the READCARD macro could be followed by a stop character:

InRec	DS	CL80	Buffer area
StopChar	DC	C'*	Stop character

The CONVERTO Macro Instruction

4

- CONVERTO is written
`<nfs> CONVERTO <number>,<address>`
- The contents of the register specified by <number> (*not* the <number> itself!) are converted to N characters in memory starting at <address>
 $0 \leq \text{<number>} \leq 15: N=12$
 $16 \leq \text{<number>} \leq 31: N=21$
 $32 \leq \text{<number>} \leq 47: N=20$
- The first character of the result is always a blank
- If the value of <number> is not between 0 and 47, the macro is ignored
- Converted negative binary values are preceded with a – sign
- Examples (where • represents a blank character):

CONVERTO	4,GR4Va1	c(GR4Va1) = •-2147483648
CONVERTO	22,GG6Va1	c(GG6Va1) = •-9223372036854775808
CONVERTO	34,FR2Va1	c(FR2Va1) = •X'FEDCBA9876543210'

B.1.2. The CONVERTO Macro Instruction

The CONVERTO macro instruction is written in the form

```
<nfs>    CONVERTO  <number>,<address>
```

where the <number> is the number of a general or floating-point register, and <address> points to a string of bytes in storage where the converted result is stored. The <address> operand may also be written as <d(b)>.

CONVERTO converts the contents of the designated register from binary to decimal characters (for general registers) or to hexadecimal characters (for floating-point registers). The first character of the converted result is always blank, so it may be printed immediately using the PRINTLIN macro.

If the contents of a general register argument is negative, a minus sign is placed before the first decimal digit.

Note that the length of the generated character string is always as specified below; be careful to allocate enough space for the result so that you won't overwrite other data or instructions.

If the value of the <number> operand does not lie in the range $0 \leq \text{<number>} \leq 47$, the macro call is ignored.

32-bit general register

A <number> between 0 and 15 specifies the corresponding 32-bit general register. For example, if the operand is 9, the contents of GR9 will be converted to a string of 12 characters. For example, if `c(GR9)=X'80000000'`, the formatted string will be the 12 characters (where • is our representation for a blank character):

•-2147483648

64-bit general register

A <number> between 16 and 31 specifies the corresponding (less 16) 64-bit general register. For example, if the operand is 16, the contents of GG0 will be converted to a string of 21 characters. For example, if `c(GG9)=X'80000000 00000000'`, the formatted string will be the 21 characters (where • is our representation for a blank character):

•-9223372036854775808

Execution on z/Architecture is required.

Floating-Point Register

A <number> between 32 and 47 specifies the corresponding (less 32) Floating-Point Register. For example, if the operand is 36, the contents of FPR4 will be printed as a string of 20 characters (where • is our representation for a blank character):

•X'FEDCBA9876543210'

Because there are three floating-point representations, and because accurate conversion from hexadecimal and binary floating-point formats is quite difficult, only hexadecimal values are displayed.

Be Careful!

If your system does not support z/Architecture instructions or the full set of 16 Floating-Point registers, trying to display their contents may cause a program interruption for an invalid operation code or a specification exception.

- DUMPOUT prints a formatted display of memory (a “dump”)
 - `<nfs> DUMPOUT <address>[,<address>]`
 - If only one operand is present, only one line is dumped
 - If both operands are present, the dump is from the lower address to the higher
- Each line starts on a word boundary and displays 32 bytes
 - The first line contains the byte at the lower address
 - The last line contains the byte at the higher address
- Example:

`Dumpout A,B Dump,including bytes from A to B`

- produces something like this:

```
*** DUMPOUT requested at Address 01A102, Statement    797, CC=0
01A000  181190EF F00C58F0 F01405EF 00F12802 0001A000 0001A204 F001A002 00000006  *...0..00....1.....s.0.....*
01A020  98EFE000 070090EF F03058F0 F03805EF 00F12802 0001A000 8001A22C 0001A026  *q.....0..00....1.....s.....*
```

B.1.3. The DUMPOUT Macro Instruction

The DUMPOUT macro instruction is written in the form

`<nfs> DUMPOUT <address>[,<address>]`

where either operand can be written as `<name>` or `<d(b)>`.

DUMPOUT prints a formatted hexadecimal dump of the area of memory starting with the fullword containing the first operand, 32 bytes to a line. If the second operand is omitted, one line will be printed. If the second operand is given, all of memory between the two addresses will be dumped. The dump starts from the lower of the two addresses and proceeds toward the higher. The lower address is “rounded down” to a fullword boundary, and 32 bytes are displayed on each line even if some bytes are at addresses greater than the higher address.¹

No checks are made to avoid possible storage access violations. For example,

`DumpA DUMPOUT A`

will cause the 32-byte area of memory starting at (or very near) A to be dumped. Similarly,

`DumpAB DUMPOUT A,B`

would print a dump of the area of memory starting with a line containing the byte at A and ending with a line which includes the byte named B.

¹ That is, the dump is from the smaller to the larger of $(\text{LowAddr}/4) \times 4$ and $((\text{HighAddr}+31)/32) \times 32$.

- PRINTLIN sends up to 121 characters to a print file


```
<nfs>    PRINTLIN  <address>[,<number>]
```

 - The character string starts at <address>
 - <number> is the number of characters (at most 121)
 - If <number> is omitted, it is assumed to be 121
- The first character is used for vertical spacing (“carriage control”) and is not printed:
 - EBCDIC ' ' (blank) means single space
 - EBCDIC '0' (zero) means double space
 - EBCDIC '-' (minus) means triple space
 - EBCDIC '1' (one) means start at the top of a new page
 - EBCDIC '+' (plus) means *no* spacing
- Example:


```
PrTt1    PRINTLIN  Title
      --
Title     DC    CL121'1Title for Top Line of a Page'
```

B.1.4. The PRINTLIN Macro Instruction

The PRINTLIN macro instruction is written in the form

```
<nfs>    PRINTLIN  <address>[,<number>]
```

where the <address> and (optional) <number> operands may also be written as <d(b)>.

PRINTLIN causes the character string beginning at the location defined by the first operand to be printed; the number of characters is specified by the second operand (which may be a predefined absolute symbol). The print-line length is limited to 121 characters.

The *first* character of the string will be *detached* and used for spacing control. The ANSI Standard carriage control characters are:

- an EBCDIC ' ' (blank) means single space,
- an EBCDIC '0' (zero) means double space,
- an EBCDIC '-' (minus) means triple space,
- an EBCDIC '1' (one) means start at the top of a new page, and
- an EBCDIC '+' (plus) means *no* spacing (the new line will be printed over the previous one).

If the second operand is omitted, the length of the character string is assumed to be 121 bytes, which means that 120 characters will be printed after the first is detached.² If the second operand is present, it is taken to be the length of the string; the number of characters specified will be placed at the left end of an internal buffer, extended to 121 bytes with blanks if necessary, and then sent to the printer file. For example, we could write

² See the discussion at Section B.3.1 for information on setting the default print-line length.


```
PrTtl    PRINTLIN  Title
```

```
- - -
```

```
Title    DC    CL121'1Title for Top Line of a Page'
```

to print the indicated title at the top of a new page. If we wrote

```
PRINTLIN  Title,1
```

then the printer would skip to the top of a new page and print a blank line there, because only the spacing control character (the “1”) is transmitted from the program to the print file.

The PRINTOUT Macro Instruction

7

- PRINTOUT supports 3 types of operand: <name>, <number>, and *

```
<nfs>    PRINTOUT  [<name>,...][<number>,...]
```



```
<nfs>    PRINTOUT  *
```

 - * terminates execution; it is treated as the last operand
 - a <number> operand between 0 and 47 refers to a register; other values are treated as an address, or ignored
 - a <name> operand causes the named area to be printed; format and length depend on operand attributes

- Examples:

```
PrintOut 1,19,32,*      Print GR1, GG3, FPR0, terminate
```

- Produces output like this:

```
*** PRINTOUT requested at Address 01A132, Statement 808, CC=0
GPR  1  = X'0001A197' =      106903
GGR  3  = X'FFFFFFFFFFFFFF' =      -1
FPR  0  = X'0000000000000000'
*** Execution terminated by PRINTOUT * at Address 01A132
```

B.1.5. The PRINTOUT Macro Instruction

The PRINTOUT macro-instruction lets you print the value of the contents of named areas of memory, the contents of registers, and to terminate execution.

The operand field of the PRINTOUT macro-instruction may contain any number of <name>s or <number>s separated by commas, with no intervening blanks. An operand consisting of a single asterisk will terminate execution. The basic forms of the PRINTOUT macro instruction are written

```
<nfs>    PRINTOUT  [<name>,...][,<number>,...]
```

```
<nfs>    PRINTOUT  *
```

where any combination of the <name> and <number> operands may be used in an operand list. Either may be written in the form <d(b)>. If the asterisk operand is used, it is treated as the last operand in the list. For example,

```
AllDone PRINTOUT 0,*
```

will display the contents of GR0 and then terminate execution.

A <number> operand with value between 0 and 47 causes the contents of a register to be printed in hex and decimal; larger values are treated as addresses. The <number> may in most cases be a predefined absolute symbol.

32-bit general register

a <number> between 0 and 15 specifies the corresponding 32-bit general register. For example, if the operand is 12, the contents of GR12 will be printed:

```
GPR 12 = X'FFFFFFFF3' = -13
```

64-bit general register

a <number> between 16 and 31 specifies the corresponding (less 16) 64-bit general register. For example, if the operand is 16, the contents of GGR0 will be printed:

```
GGR 0 = X'1234567890ABCDEF' = 1311768467294899695
```

Execution on z/Architecture is required.

Floating-Point Register

a <number> between 32 and 47 specifies the corresponding (less 32) Floating-Point Register. For example, if the operand is 36, the contents of FPR4 will be printed:

```
FPR 4 = X'FEDCBA9876543210'
```

Because there are three floating-point representations, and because accurate conversion from hexadecimal and binary floating-point formats is quite difficult, only hexadecimal values are displayed.³

To print the contents of the “original” four System/360 floating-point registers F0, F2, F4, and F6, we could write

```
FourFPRs PRINTOUT 32,34,36,38
```

Printing the contents of any other Floating-Point Register requires those registers to be installed and available on your machine.

To print the contents of R14 and then terminate execution, we could write

```
PRINTOUT X'E',*
```

To print the contents of memory areas named A, B, and C, we could write

```
PRINTOUT A,B,C
```

The format of the output depends on the type attribute of the symbol naming the memory area:

- Type attribute C (character) data is shown as strings of at most 100 characters.
- Type attribute F or H data are shown as signed decimal numbers.
- If you use forms like PRINTOUT d(b) and d has attributes C, F, or H, the result will be formatted as above; otherwise, the data is displayed as 16 hexadecimal digits.
- Other type attributes cause data to be displayed as 2 to 100 hexadecimal digits, depending on the length attribute of the operand.

Specifying PRINTOUT with no operand is useful for flow tracing; only a header line is printed. An example is shown below.

```
PRINTOUT
```

If you want a comment field on the statement, put a single comma as the operand:

```
PRINTOUT , Your comments here
```

Finally, if you want to terminate execution with no message:

```
PRINTOUT *,Header=NO Terminate quietly
```

Any value other than “No” (in any mixture of upper and lower case) is treated as “Yes”.

³ If your system does not support z/Architecture instructions or the full set of 16 Floating-Point registers, trying to display their contents may cause a program interruption for an invalid operation code or a specification exception.

- READCARD reads 80-byte records into your program

```
<nfs>    READCARD  <address>[,<address>]
```

- The first operand specifies the location in your program for the record
- If no records remain ("end of file", EOF)
 1. If the second operand is present, control is returned to that location
 2. If the second operand is omitted, the program is terminated with a message

*** Execution terminated by Reader EOF

- Example:

```
GetARec  READCARD  MyRecord,EndFile
        - - -
EndFile  - - -      Do something about no more records
```

B.1.6. The READCARD Macro Instruction

READCARD reads records into an 80-byte buffer in the program. This macro-instruction is written

```
<nfs>    READCARD  <address>[,<address>]
```

where either <address> operand may also be written as <name> or <d(b)>.

It reads a record from the input file into the 80-byte area beginning at the first operand address. If no record is available, then (1) control is returned to the instruction specified by the second operand if it is present, or (2) if no second operand is present, execution is terminated with the message

*** Execution terminated by Reader EOF

For example,

```
        READCARD  MyRecord
```

will read the next record and place it as an 80-byte EBCDIC character string at the location named MyRecord; if no record is present, execution will be terminated. The instruction

```
GetARec  READCARD  MyRecord,EndFile
        - - -
EndFile  - - -      Do something about no more records
```

does the same as the previous example, except that if no record is available, control will be transferred to the instruction named EndFile.

- Previous examples (slides 5 and 7) have illustrated DUMPOUT/PRINTOUT header lines:

```
*** PRINTOUT requested at Address xxxxxx, Statement sssss, CC=n
or
*** DUMPOUT requested at Address xxxxxx, Statement sssss, CC=n
```

- where sssss is the statement number of the macro
- where CC=n is the Condition Code at that point

- The header line can be suppressed by specifying an operand

Header=no

at any position in the operand list; mixed case is OK

- Examples:

```
DUMPOUT  A,B,Header=NO
PRINTOUT 0,19,header=no,34,*
```

B.1.7. PRINTOUT and DUMPOUT Headers

Normally, the output produced by the PRINTOUT and DUMPOUT macros will be preceded by a “header” line:

```
*** PRINTOUT requested at Address xxxxxx, Statement sssss, CC=n
or
*** DUMPOUT requested at Address xxxxxx, Statement sssss, CC=n
```

where sssss is the statement number of the macro, and CC=n shows the current Condition Code setting.

To suppress this header line, you can specify an additional operand HEADER=NO on the PRINTOUT or DUMPOUT macro. For example:

```
PRINTOUT  A,B,C,Header=No
ABDMP    DUMPOUT  A,B,header=no
```

The default is HEADER=YES.

- All the macros must execute in 24-bit addressing mode, AMODE(24), and reside below the 16MB “line”, RMODE(24)
- The instructions generated by the macros are self-modifying, as is the generated “service” control section; programs using these macros are not reenerable
- Most operands of the form <address>, <name>, and <number> are resolved in S-type address constants, so addressability is required when a macro is invoked
- Be careful not to reference areas outside your program
- At most 8 characters of <name> and <d(b)> operands are displayed by PRINTOUT

B.1.8. Usage Notes

1. All the macros require residence in RMODE(24) storage below the 16MB “line”, and execute in AMODE(24). The generated code is frequently self-modifying, and is not re-enterable.
2. Most operands of the form <name>, <d(b)>, and <number> are resolved in S-type address constants, so addressability is required when all macros except \$GENIO are invoked.
3. When you execute a macro, be sure that the base register used at assembly time to resolve the S-type constants has the correct address at execution time.
4. Be careful not to reference areas outside your program, as you may risk interruptions for memory protection violations.
5. If you use PRINTOUT to display named areas of memory, it uses the name's attributes for formatting the result.
6. At most eight characters of the <name> and <d(b)> operands are displayed. If you write

```
PRINTOUT 00000000(3),00000000(7)
```

the eight bytes addressed by registers GR3 and GR7 will be displayed in hexadecimal, but both will have the “name” 00000000.

- This sample program and its listing and output are shown in the text

	Print Nogen	Suppress expansions
IOSamp	Csect ,	Sample Program
	Using *,15	Local base register
	SR 1,1	Clear card counter
*		Next statement for flow tracing
	PrintOut	
Read	ReadCard CardOut,EOF	Read card until endfile
	LA 1,1(0,1)	Increment card counter
	PrintOut 1	Print the count register
	PrintLin Out,LineLen	Print a line
	ConvertI 2,CardOut	Convert a number into GR2
	ConvertO 2,OutData	Put it in printable form
	PrintLin OutData,L'OutData	Print the value
	B Read	Go back and read again
EOF	DumpOut IOSamp,Last	Dump everything
	XGR 3,3	Set GG3 to 0
	BCTGR 3,0	Now set GG3 to -1
	PrintOut 1,19,32,*	Print GR1, GG3, FPR0, terminate
Out	DC C'0Input Record = ''	First part of line
CardOut	DC CL80' ',C' ''	Card image here
LineLen	Equ *-Out	Define line length
OutData	DS CL12	Converted characters
Last	Equ *	Last byte of program
	End	

B.2. Sample Program

Here is a small sample program that uses all of these macros. The assembly listing and its output are in the following figures.

	Print Nogen	Suppress expansions
IOSamp	CSECT ,	Sample Program
	Using *,15	Local base register
	SR 1,1	Clear card counter
*		Next statement for flow tracing
	PrintOut	
Read	ReadCard CardOut,EOF	Read card until endfile
	LA 1,1(0,1)	Increment card counter
	PrintOut 1	Print the count register
	PrintLin Out,LineLen	Print a line
	ConvertI 2,CardOut	Convert a number into GR2
	ConvertO 2,OutData	Put it in printable form
	PrintLin OutData,L'OutData	Print the value
	B Read	Go back and read again
EOF	DumpOut IOSamp,Last	Dump everything
	XGR 3,3	Set GG3 to 0
	BCTGR 3,0	Now set GG3 to -1
	PrintOut 1,19,32,*	Print GR1, GG3, FPR0, terminate
Out	DC C'OInput Record = ''	First part of line
CardOut	DC CL80' ',C'''	Card image here
LineLen	Equ *-Out	Define line length
OutData	DS CL12	Converted characters
Last	Equ *	Last byte of program
	End	

The program uses READCARD to read two card images, keeps a count in GR1 which is displayed with PRINTOUT. The card image just read is shown with a prefix using PRINTLIN. Then, one number from each record is converted to binary using CONVERTI, and then to printable format using CONVERTO. The result of each conversion is printed using PRINTLIN.

When there are no more input records, DUMPOUT shows the entire program. Finally PRINTOUT * also displays the contents of GR1, GG3, and FPR0 before terminating the program.

The two 80-byte card-image input records look like this:

+123456	* First record
-000034567890	* Second and last record

Here is a portion of the assembly listing, including a carriage control character on the first line:

0		1	Print Nogen	Suppress expansions
000000	00000 001E4	2	IOSamp CSECT ,	Sample Program
	R:F 00000	3	Using *,15	Local base register
000000 1B11		4	SR 1,1	Clear card counter
		5	*	Next statement for flow tracing
000002 90EF F00C	0000C	6	PrintOut	
000024 0700		743	Read ReadCard CardOut,EOF	Read card until endfile
00004A 4110 1001	00001	752	LA 1,1(0,1)	Increment card counter
00004E 90EF F058	00058	753	PrintOut 1	Print the count register
00007C 0700		764	PrintLin Out,LineLen	Print a line
00009C 0700		772	ConvertI 2,CardOut	Convert a number into GR2
0000BC 0700		780	ConvertO 2,OutData	Put it in printable form
0000DC 0700		788	PrintLin OutData,L'OutData	Print the value
0000FC 47F0 F026	00026	796	B Read	Go back and read again
000100 0700		797	EOF DumpOut IOSamp,Last	Dump everything
000128 B982 0033		806	XGR 3,3	Set GG3 to 0
00012C B946 0030		807	BCTGR 3,0	Now set GG3 to -1
000130 0700		808	PrintOut 1,19,32,*	Print GR1, GG3, FPR0, terminate
000176 F0C99597A4A340D9		823	Out DC C'OInput Record = ''	First part of line
000187 4040404040404040		824	CardOut DC CL80' ',C'''	Card image here
	00062	825	LineLen Equ *-Out	Define line length
0001D8		826	OutData DS CL12	Converted characters
	001E4	827	Last Equ *	Last byte of program
		828	End	

The output from this sample program is shown below. The listing shows the carriage control characters.

```

*** PRINTOUT requested at Address 01A002, Statement 6, CC=0
*** PRINTOUT requested at Address 01A04E, Statement 753, CC=0
GPR 1 = X'00000001' = 1
0Input Record = " +123456 * First record "
123456
*** PRINTOUT requested at Address 01A04E, Statement 753, CC=0
GPR 1 = X'0001A192' = 106898
0Input Record = " -000034567890 * Second and last record "
-34567890
*** DUMPOUT requested at Address 01A102, Statement 797, CC=0
01A000 1B1190EF F00C58F0 F01405EF 00F12802 0001A000 0001A204 F001A002 00000006 *....0..00....1.....s.0.....*
01A020 98EFE000 070090EF F03058F0 F03805EF 00F12802 0001A000 8001A22C 0001A026 *q.....0..00....1.....s.....*
01A040 F18798EF E00047F0 F1024110 100190EF F05858F0 F06005EF 00F12802 0001A000 *1gq....01.....0..00-...1.....*
01A060 0001A204 0001A04E 000002F1 A0070001 F1404040 40404040 98EFE000 070090EF *..s....+...1....1 q.....*
01A080 F08858F0 F09005EF 00F12802 0001A000 0001A1F8 F1760062 98EFE000 070090EF *0h.00....1.....81...q.....*
01A0A0 F0A858F0 F0B005EF 00F12802 0001A000 0001A220 0002F187 98EFE000 070090EF *0y.00....1.....s...1gq.....*
01A0C0 F0C858F0 F0D005EF 00F12802 0001A000 0001A214 0002F1D8 98EFE000 070090EF *0H.00....1.....s...1Qq.....*
01A0E0 F0E858F0 F0F005EF 00F12802 0001A000 0001A1F8 F1D8000C 98EFE000 47F0F026 *0Y.000...1.....81Q..q....00.*
01A100 070090EF F10C58F0 F11405EF 00F12802 0001A000 0001A1E8 0001A102 0000031D *...1..01....1.....Y.....*
01A120 F000F1E4 98EFE000 B9820033 B9460030 070090EF F13C58F0 F14405EF 00000000 *0.1Uq....b.....1..01.....*
01A140 00000000 0001A204 0001A132 00000328 20070001 F1404040 40404040 20070013 *.....s.....1 .....*
01A160 F1F94040 40404040 20070020 F3F24040 40404040 0800F0C9 9597A4A3 40D98583 *19 .....32 ..0Input Rec*
01A180 96998440 7E407F40 404060F0 F0F0F0F3 F4F5F6F7 F8F9F040 40404040 40404040 *ord = " -000034567890 *
01A1A0 40404040 5C40E285 83969584 40819584 409381A2 A3409985 83969984 40404040 * * Second and last record *
01A1C0 40404040 40404040 40404040 40404040 4040407F 40404060 F3F4F5F6 * " -3456*
01A1E0 F7F8F9F0 00000000 900FF098 9201F136 41C00008 A7F40020 900FF088 41C00004 *7890.....0qk.1.....x4....0h....*
*** PRINTOUT requested at Address 01A132, Statement 808, CC=0
GPR 1 = X'0001A197' = 106903
GGR 3 = X'FFFFFFFFFFFFFFFF' = -1
FPR 0 = X'0000000000000000'
*** Execution terminated by PRINTOUT * at Address 01A132

```