Canola

Reference Manual

Peter Miller
pmiller@opensource.org.au

.

This document describes Canola version 0.8.D001
and was prepared 20 May 2012.

Table of Contents(Canola)

Table of Contents(Canola)

DESCRIPTION

The *canola*(1) command is used emulate a Canon Canola 1614P, a programmable desktop calculator from 1972. It had 14 floating point memories, and 240 instructions of program. A few still survive today in working condition.

DEPENDENCIES

The following libraries are required to build the Canola project:

g++ You will need a C++ compiler. The GNU C++ compiler is what the author uses.

groff The documentation is formatted using GNU Groff.

Boost Library
: This is used for several useful C++ classes and templates. This is usually called "`libboost-dev`" or similar on package based systems.

libexplain
: This is used for all error reporting, when operating system errors occur. This is usually called "`libexplain-dev`" or similar on package based systems.

Gtkmm-3.0
: The C++ binding of the Gnome-3 widget library is used. Note that this library tends to have numerous dependencies. This is usually called "`libgtkmm-3.0-dev`" or similar on package based systems.

libjpg This is used to read the JPEG images that are used to build the various custom button and widgets. This is usually called "`libjpg62-dev`" or similar on package based systems.

libpng This is used to read the PNG images that are used to build the various custom button and widgets. This is usually called "`libpng12-dev`" or similar on package based systems.

sharutils This provides the *uudecode*(1) command, used to turn `.uue` source files into binary files during the build.

BUILDING

This is traditional open source software, and the usual build method applies. After fetching the tarball, do the following:

```
$ tar xzf canola-0.8.D001.tar.gz
$ cd canola-0.8.D001
$ ./configure --prefix=/usr
$ make
$
```

The software includes a test suite:

```
$ make sure
Passed all tests.
$
```

If you wish to install the software, only one more command is required:

```
$ make install
$
```

All done. See the *canola*(1) man page for more how to use it.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

RELEASE NOTES

This section details the various features and bug fixes of the various releases.

Version 0.8 (2012-May-20)

- The emulator now implements more of the conditions described on page 21 of the 1614P Instruction Manual, concerning how and when floating truncation takes place, and what is displayed for overflow.
- The emulator now as an Edit−>Preferences dialog box, for changing some aspects of how the emulator is presented.

Version 0.7 (2012-Jan-10)

- Antonio Arias <antonio.arias99999@gmail.com> discovered that the Makefile.in file was missing all of the entries needed to install the PNG files needed by the GUI. This has been fixed.

Version 0.6 (2012-Jan-09)

- Antonio Arias <antonio.arias99999@gmail.com> discovered that one too many "gui" path components were added when searching for `.png` files for the custom widgets.
- A bug has been fixed in the management of transparency of the card reader window. The use of Gtk::Window::set_opacity was wrong, it makes the whole window, including the contents, transparent. Instead, the window needs to (a) have an RGBA visual, and (b) set the background to transparent.

Version 0.5 (2012-Jan-07)

- The *canola*(1) program now includes a card reader animation, with an overlayed undecorated window. If a compositing window manager is used, it will be transparent as well.
- The dot down the left hand side of the program listing has been removed, because the real Canola 1614P doesn’t print one.
- The *canola*(1) command is now able to save the data memories as an assembler program file. This can be useful when producing cards sets of constants, as seen in the GF series of cards in the Program Card Package.
- The *canola*(1) command now has a more faithful replicas of the Power Button, the Program Selector Switch Program Mode Switch, and the Rounding Switch.
- The *canola-card-scanner*(1) command is now able to look for the rows and columns of holes by itself, making it more able to cope with slight differences with image scanner DPI settings and alignment.
- The address field in the Debugger pane can now be modified by the user. This is helpful when debugging jumps.
- The *canola-asm*(5) man page now has details of each opcode.
- The Program Card Package (included with each 1614P) is now complete, the final MS-18 to MS-25 subroutines have been added.

Version 0.4 (2011-Dec-30)

- There is now a PDF of the Canola project’s Reference Manual on the web site.
- The Program Card Package now includes MS-7 a**n, MS-8 a**x, MS-9 10**x, MS-10 log10(x), MS-11 exp(x), MS-12 ln(x), MS-13 sinh(x), MS-14 cosh(x), MS-15 tanh(x), MS-16 asinh(x), and MS-17 acosh(x).
- The Program Card Package is now available on the web site, as example software, linked from the Documentation page.
- The *canola-card-scanner*(1) program now also understands JPEG image files, in addition to the PNG files it understood previously.
- Hans Dorn <hans.dorn@gmail.com> reports that the START keys is ignored in LRN mode, and now the emulator conforms.
- Hans Dorn <hans.dorn@gmail.com> reports that the $710..$715 opcodes cause an overflow and display

9999999999999999, and now the emulator conforms.

Version 0.3 (2011-Dec-24)

- The Test Run Program still does not perform correctly, so there are still some instruction set subtleties that have yet to be discovered and addressed. There are many new questions in the questions.html file on the web site.
- The MS-3 to MS-6 card sets are now included, in addition to MS-1 and MS-2 previously.
- Thanks to scans provided by Markus Brenner <hyperion2@utanet.at>, the Statistics ST-1 to ST-4 card sets are now included.
- A bug has been fixed in the handling of the EJ instruction. It was not correctly detecting whether or not the user had entered data.
- Thanks to scans provided by Markus Brenner <hyperion2@utanet.at>, the Group Function GF-1 to GF-4 card sets are now included.
- The program load and program save dialog boxes now start from the current directory, rather than the Recently Used list.
- The emulator now include support for using the [UJ] key when in [OPE] mode, to cause execution to start immediately from the indicated flag jump.
- Chris Baird <cjb@brushtail.apana.org.au> contributed his prime number program, and his quadratic equation solver.
- The error dialog is no longer modal. If more than one error happens before you can click "OK" then all of them will accummulate in the secondary text area.
- Hans Dorn <hans.dorn@gmail.com> discovered that the real 1614P isn't very fussy about the key following RM3˜ (et al). It simply uses the lower 4 bits of whatever key you press.
- Markus Brenner <hyperion2@utanet.at> responded to the "Questions" page on the web site with answers to many of the questions, and those answers have been incorporated.
- The file menu entry Program Card System now throws up a dialog if you don't have it in LRN mode. You can still choose OPE, but the default is LRN.
- Chris Baird <cjb@brushtail.apana.org.au> discovered some build problems with the latest version of libpng, as they phased out some long-deprecated struct accesses.
- There is a new *canola-card-printer*(1) program for drawing Canon Canola 1614P card images. It can draw them as PDF, PNG or SVG. It can draw the classic "blue" cards, or the blank green ones, *etc*.

Version 0.2 (2011-Nov-27)

- The state machine controlling the calculator's arithmetic and operator precedence has been extensively tested and improved.
- A unit test for MS-1 (sin) has been added, and another bug in it has been foxed.
- There is a new debugger pane. It displays the register values, the memory values, and when a program is executing, it shows the source code with the nect opcode to execute highlighted.
- The test suite has been supplemented with examples from the 1614P Calculator Instruction Manual.
- The [K] key (constant mode) is now implemented, for both multiplication and division.
- A bug in the SM<n> opcodes has been fixed, they were storing into the wrong location.
-

The Program Print key on the printer is now permitted in Program Mode OPE.

Version 0.1 (2011-Oct-27)

- Program Mode Switch now supported for all modes.
- Retro widgets now implemented for all keyboard buttons.
- Most opcodes are now implemented, including jumps and subroutines. Some opcodes still have subtle problems.

Version 0.0 (2011-Oct-01)

NAME

canola − Canon Canola 1614P programmable calculator emulator

SYNOPSIS

canola [*option*...]

canola −V

DESCRIPTION

The canola command is used to emulate a 1972 Canon Canola 1614P programmable desktop calculator.

OPTIONS

The canola command understands the following options:

−V Print the version of *canola* and exit.

EXIT STATUS

The *canola* command terminates with an exit status of 1 for all errors. It will only have an exit status of 0 if there are no errors.

EXIT STATUS

The *canola* command terminates with an exit status of 1 for all errors. It will only have an exit status of 0 if there are no errors.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

canola−asm − assemble and disassemble code files

SYNOPSIS

canola-asm *source-file binary-file*

canola-asm −d *binary-file source-file*

canola-asm −V

DESCRIPTION

The canola-asm command is used to assemble and disassemble binary Canon Canola 1614P programs, and program card images.

OPTIONS

The canola-asm command understands the following options:

−d The normal mode of operation is to assemble (convert text to binary). This option requests the opposite, to disassemble (convert binary to text).

−I *directory*
This option may be used to specify a directory to look in for include files. This option may be given more than once, directories will be searched in the order specified.

−V Print the version of *canola-asm* and exit.

EXIT STATUS

The *canola-asm* command terminates with an exit status of 1 for all errors. It will only have an exit status of 0 if there are no errors.

EXIT STATUS

The *canola-asm* command terminates with an exit status of 1 for all errors. It will only have an exit status of 0 if there are no errors.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

canola-card-printer − draw images of Canola 1614P program cards

SYNOPSIS

canola-card [*option...*][*infile* [*outfile*]]

canola-card −V

DESCRIPTION

The canola-card command is used to read a Canola 1614P program, in assembler, and draw card images for that program.

The kind of output to write depends on the file extension of the output file. Supported formats include “pdf”, “png”, “ps” and “svg”.

The PDF format includes an envelope that can be cut out and glued together to hold the set of cards.

OPTIONS

The canola-card command understands the following options:

−I *directory*

This option may be used to specify a directory to look in for include files. This option may be given more than once, directories will be searched in the order specified.

−V Print the version of *canola-card* and exit.

EXIT STATUS

The *canola-card* command terminates with an exit status of 1 for all errors. It will only have an exit status of 0 if there are no errors.

EXIT STATUS

The *canola-card* command terminates with an exit status of 1 for all errors. It will only have an exit status of 0 if there are no errors.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

canola-card-scanner − scan card images to extract the code

SYNOPSIS

canola-card-scanner *image-file*.png *binary-code-file*

canola-card-scanner −V

DESCRIPTION

The canola-card-scanner command is used to scan images of Canon Canola 1614P program cards, to extract the instructions into a binary card image.

If you want to turn the binary code into text, to read or to edit, used the *canola-asm*(1) command, with the **−d** option to disassemble the binary file into text.

OPTIONS

The canola-card-scanner command understands the following options:

−b Usually, a text disassemble of the program card is produced. This option asks that a 40-byte binary output file be written instead.

−i Normally the canola-card-scanner program expects white holes in a blue card, when the white underside of the lid of the scanner shows through the holes. This option may be used to process black holes instead, by inverting the colors.

−V Print the version of *canola-card-scanner* and exit.

EXIT STATUS

The *canola-card-scanner* command terminates with an exit status of 1 for all errors. It will only have an exit status of 0 if there are no errors.

EXIT STATUS

The *canola-card-scanner* command terminates with an exit status of 1 for all errors. It will only have an exit status of 0 if there are no errors.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

canola-asm − assembler language description

DESCRIPTION

The *canola-asm*(1) command accepts a language that strongly resembles the way Canon Canola 1614P programs are written in the various books. You are referred to the Canon Canola 1614P Instruction Manual for authoritative descriptions of the opcodes.

Source File Format

In general, white space is ignored, except as it serves to separate tokens.

Comments start with a semicolon (“;”) or hash (“#”) character and extend to the end of the line.

You don’t have to put just one opcode per line, even though this is conventionally done in the Instruction Manual. This is quite convenient when writing numbers. It also makes jump opcodes clearer, to just place a space between the opcode and the label, making the association between them clearer.

OPCODES

This is a brief description of each of the valid opcodes. For an authoritative source, consult the Canon Canola 1614P Instruction Manual.

Registers

The following opcode descriptions describe the operations of the calculator using the terms *x-register* and *y-register*. The x-register is the one that can be seen on the nixie tube display. The y-register is the one “behind” the display, holding the other operand for arithmetic operations.

Include Files

You can use include files. This is very handy for including subroutines from the Program Card Package, which is automatically on the include file search path.

`include` *file-name*

The file name is *not* quoted, and extends to the end of the line. It is probably not a good idea to have white space or shell meta characters in the file-name.

Raw Opcodes

From time-to-time it is necessary to represent a value that is not a valid opcode. The convention used here is to use a dollar sign (“$”) followed by three decimal digits. For example, the zero “0” key can also be represented as “$700”. This number is the one you see in the right hand opcode column in a program listing, or on the display in LRN mode and PRO CHE mode.

```
SUJ $014
```

The disassembler facilities in the software provided by the Canola project choose to always display jump labels and subroutine jump labels in this way. Despite this, there are many opcodes that may not be used as a jump label, including the `$000` opcode, all the `$`*n*`15` opcodes, and all of the `$1`*nn* opcodes.

$001 Plus Equals (+=)

If there is no deferred operation, this opcode results in the x-register being replaced by the sum of the x-register and the y-register. The y-register is unchanged.

If the deferred operation is multiplication (see the $003 multiply opcode) the x-register is replaced by the product of the x-register and the y-register. Note: if the result of the multiplication is zero, the y-register assumes the previous value of the x-register, otherwise the y-register is unchanged. The deferred operation is cleared.

If the deferred operation is division (see the $004 divide opcode) the x-register is replaced by the y-register divided by the y-register. The y-register is unchanged. The deferred operation is cleared.

The rounding switch and the decimal selector dial affect the result, except when preceded by one of the $606 round up, $606 round off, or $607 round down opcodes.

$002 Minus Equals (-=)

If there is no deferred operation, this opcode results in the x-register being replaced by the y-register minus the y-register. The y-register is unchanged.

If the deferred operation is multiplication (see the $003 multiply opcode) the x-register is replaced by the negative product of the x-register and the y-register. Note: if the result of the multiplication is zero, the y-register assumes the previous value of the x-register, otherwise the y-register is unchanged.

If the deferred operation is division (see the $004 divide opcode) the x-register is replaced by the negative of the y-register divided by the y-register. The y-register is unchanged.

The rounding switch and the decimal selector dial affect the result, except when preceded by one of the $605 round up, $606 round off, or $607 round down opcodes.

$003 Multiply (*)

The deferred operation is set to multiplication.

If there was no previous deferred operation, the x-register is copied into the y-register.

$004 Divide (/)

The deferred operation is set to division.

If there was no previous deferred operation, the x-register is copied into the y-register.

$005 Square Root (SQRT)

The x-register is replaced by the square root of the x-register. The sign of the x-register is ignored. They y-register is set to zero.

$101 Entry (ENT)

This opcode causes execution to stop, with the ENTRY lamp lit. When the user presses the START key, execution will resume at the following opcode.

If the user presses SJ instead of START, execution will resume from the first program step (1 or 121, depending on the bank selector switch).

$102 Sense Jump (SJ)

The SJ opcode is followed by jump destination label. See the $107 Flag Jump opcode for an explanation of jump destinations.

This opcode causes execution to stop, with the ENTRY lamp lit. When the user presses the SJ key, the program will jump to the specified program step. If the START key is pressed instead of the SJ key, the program will proceed to the following instruction.

$103 Entry Jump (EJ)

The EJ opcode is followed by jump destination label. See the $107 Flag Jump opcode for an explanation of jump destinations.

This opcode causes execution to stop, with the ENTRY lamp lit. If the user enters a number (or even performs some calculations) and then presses the START key, the program will jump to the specified program step. If the START key is presses *without* a number being entered, the program will proceed to the following instruction.

$105 Minus Jump (MJ)

The MJ opcode is followed by jump destination label. See the $107 Flag Jump opcode for an explanation of jump destinations.

If the x-register is negative, the program will jump to the specified program step. If the x-register is zero or positive, the program will proceed to the following instruction.

$106 Unconditional Jump (UJ)

The UJ opcode is followed by jump destination label. See the $107 Flag Jump opcode for an explanation of jump destinations.

Execution will unconditionally jump to the specified program step.

$107 Flag Jump (`FJ`)

Where you would expect a label in a more modern assembler language, the Canola 1614P has Flag Jumps. The FJ opcode is followed by almost any 7-bit value, the “label”. When a jump opcode (SJ, EJ, MJ or UJ) needs to know where to jump to, it looks for a matching FJ opcode.

When execution flows past an FJ opcode, it is a two instruction no-op.

You may not use $000, $*n*15, or $1*nn* as a Flag Jump label.

The advantage of this system is that code is relocatable without modification, called *position independent code* in modern terms. Also, the instructions are only 7 bits wide, so and absolute address scheme would be limited to a 7-bit address space, and a relative addressing scheme would be limited to ±63 steps.

The disadvantage of this system is that Flag Jump opcodes take two instruction slots to perform a no-op. Also, the FJ search is performed from step 1, which makes jumps take longer when the FJ is further from 1.

The Flag Jump label space and the Subroutine Flag Jump label space are independent. A search for one cannot be satisfied by the other.

$108 Subroutine Jump (`SUJ`)

The SUJ opcode is followed by subroutine jump destination label. The execution step of the following opcode is saved, and then execution jumps to the specified program step.

$109 Subroutine Flag Jump (`SFJ`)

When a subroutine jump opcode (SUJ) needs to know where to jump to, it looks for a matching SFJ opcode.

Where you would expect a label in a more modern assembler language, the Canola 1614P has Subroutine Flag Jumps. The SFJ opcode is followed by almost any 7-bit value, the “label”. When a subroutine jump opcode (SUJ) needs to know where to jump to, it looks for a matching SFJ opcode.

When execution flows past an SFJ opcode, it is a two instruction no-op.

The SFJ search is performed from step 1, which makes subroutine jumps take longer when the SFJ is further from 1. Subroutines are used to factor out common code, so it make sense to try to make them as fast as possible to find.

The Subroutine Flag Jump label space and the Flag Jump label space are independent. A search for one cannot be satisfied by the other.

$110 Subroutine Return Jump (`SRJ`)

Execution jumps to the address saved by the SUJ opcode.

$201..$214 Add Memory (`M1` .. `M14`)

This opcode adds the x-register to the given floating point data memory.

$301,$302 Minus Memory (`MM1`, `MM2`)

This opcode subtracts the x-register from the given floating point data memory.

Note: only M1 and M2 are able to be used in this way.

$303..$314 Store Memory (`SM3` .. `SM14`)

This opcode saves the x-register into the given floating point data memory, discarding any previous value.

Note: only M3..M14 are able to be used in this way.

$401..$414 Recall Memory (`RM1` .. `RM14`)

The y-register is set to the value of the x-register, and then the x-register is set to the value of the given floating point data memory.

$500 Clear Indicator (`CI`)

The x-register is set to zero.

$501..$514 Clear Memory (`CM1` .. `CM14`)

This opcode saves zero in the given floating point data memory.

$600 Decimal Point (.)
This opcode inserts a decimal point in the number being constructed.

If we have already added some digits to the right of the decimal point, the digits stay the same, but the decimal point moves back to the right hand side of the display.

$601 Reverse (RV)
This opcode exchanges the values of the x-register and the y-register.

$602 Right Shift (->)
This opcode removes the right most digit, usually resulting in the display shifting to the right (hence the name, and the direction of the arrow).

When entering a number, if the value includes digits to the right of the decimal point, and the right shift clears all decimals, the next number key will be to the right of the decimal point. That is, the "decimalness" is not cleared by the right shift key.

A right shift that results in zero also clears the negative sign

A right shift clears any overflow of the value.

$603 Change Sign (CS)
This opcode changes the sign of the x-register.

$605 Round Down (FLOOR)
This opcode is followed by a $700..$714 opcode to indicate the desired number of decimal places.

The next += or −= opcode will ignore the Rounding Switch and the Decimal Point Selector Dial, and will instead round down to the specified number of decimal places.

By "round down" we actually mean "round towards zero". This is uninteresting for positive numbers, but significant for negative numbers.

$606 Round Off (ROUND)
This opcode is followed by a $700..$714 opcode to indicate the desired number of decimal places.

The next += or −= opcode will ignore the Rounding Switch and the Decimal Point Selector Dial, and will instead round off to the specified number of decimal places.

$607 Round Up (CEIL)
This opcode is followed by a $700..$714 opcode to indicate the desired number of decimal places.

The next += or −= opcode will ignore the Rounding Switch and the Decimal Point Selector Dial, and will instead round up to the specified number of decimal places.

By "round up" we actually mean "round away from zero". This is uninteresting for positive numbers, but significant for negative numbers.

$608 PRINT
This opcode causes the value of the x-register to be printed; this takes 420ms.

If the printers is not connected, or is turned off, *it does something else*.

$609 Paper Feed (FD)
This opcode causes a blank line to be printed; this takes 290ms. If the printers is not connected, this is no-op.

$700..$709 Digits 0..9
These opcodes are used to enter digits of a number being constructed. If to the left of the decimal point, the x-register is multiplied by 10, and the digit added; the decimal point does not move. If to the right of the decimal point, almost the same thing happens, but the decimal point moves one place to the left.

$710..$714 Numbers 10..14
These opcodes cannot be used to enter digits, or the calculator will immediately produce an overflow error. Additionally, it is ambiguous to write these opcodes as "10" (*etc*) as this will appear to the lexer as two $701 $700 opcodes. When using it as a label or as a rounding length, use $710 instead.

Program Card Package

The Canon Canola 6114P came with a Program Card package. These included several useful subroutines. the first column is the SUJ label.

$201	MS-1	sin(x)
$202	MS-2	cos(x)
$203	MS-3	tan(x)
$204	MS-4	asin(x)
$205	MS-5	acos(x)
$206	MS-6	atan(x)
$207	MS-7	a**n
$208	MS-8	a**x
$209	MS-9	10**x
$210	MS-10	log10(x)
$211	MS-11	exp(x)
$212	MS-12	ln(x)
$213	MS-13	sinh(x)
$214	MS-14	cosh(x)
$301	MS-15	tanh(x)
$302	MS-16	asinh(x)
$303	MS-17	acosh(x)
$304	MS-18	atanh(x)
$305	MS-19	Polar to Rectangular
$306	MS-20	Rectangular to Polar
$307	MS-21	Degree,Minute,Second to Degrees
$308	MS-22	Degrees to Degree,Minute,Second
$309	MS-23	Degrees to Radians
$310	MS-24	Radians to Degrees
$311	MS-25	Factorial N

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/