CocoIDE: Coco de Mer 8 Integrated Development Environment Software Manual

Installation and User Guide for Windows, Linux and Mac

1 Introduction

This manual provides instructions for installing and using the Coco De Mere 8 (CDM8) integrated development environment software CocoIDE, on your own PC or Laptop. CocoIDE is programmed in Python, and allows you to easily edit, develop and test your assembler programs for the CDM8 microprocessor system. You can run the CocoIDE software using your choice of Windows, Linux or Mac operating systems. This manual will guide you through the process of installing the CDM8 development software on your own PC or Laptop, and through the CDM8 program development process by using an example: Writing and testing your first CDM8 assembler program, "HelloWorld.asm". [1]

The Computer Science PC Labs have CocoIDE already installed. You will also be expected to work independently through a tailored set of CDM8 Programming exercises, and make your independent program submissions via email to the main CocoMaRo (CDM8 Marking Robot). More details will be provided in Lectures and Tutorials.

2 Installation:

The general procedure for installation is the same for all platforms. However, configuring the software to run from the command line, and also setting up to it use from the various program menus is different for each platform. Details will be provided for each platform in the sections below

2.1 Pre-Requisites

Before installing the CDM8 software, it is essential that the following programs are correctly installed on your system:

1. **Python**. The software has been tested with both Python 2.7 and 3.5. Older versions (e.g. 2.6, 3.3 etc.) will probably run fine, but be aware there is a small possibility that you might encounter some subtle bugs with older versions. **The recommended Python version is >= 3.5**.

2. **Logisim**. The version used for this course is 2.7.0. You will be using this program in semester B when you will be creating your own useful (simulated) additions to the core CDM8 processor

1 Note, it is traditional for any specific programming language tutorial to start with a simple program that simply displays "Hello World", so this manual takes the same approach for the CDM8 system.

system.

Installation of the Python, and Logisim programs on your particular system are covered in more detail in the following section. Both these programs are Open Source and free to use.

2.1.1 Python Installation

Python may already be installed on your system if your PC runs either Linux or Mac OSX Operating Systems (OS). If it is V2.7 (most probable!) then it is simplest to use the version you have installed already. If you find you do not have Python installed already (most Windows systems), or you wish to install a newer version, the procedure depends on your Operating System. For Windows and Mac OSs, visit www.python.org and follow the instructions for installation for your OS version. When installing the Windows version of Python, please make sure you select the option to "modify environment/path variables", which depending on the installer version may be hidden away under "Advanced" options.

For Linux (and also possibly Mac OS?), Python 2 and 3 are normally available directly from your version's program repositories via your favourite package manager or software installer. This is the preferred method to install Python for Linux based systems.

Python 3.5 (or greater) is the recommended version as it is the version that has been used to develop CocoIDE and therefore testing on other versions necessarily less extensive.

Notes:

Do not download or try to install the source version of Python unless you are happy compiling your own software for your OS!

If your Linux or Mac OS already has Python2.7 installed (almost certainly!), and you have installed Python 3.5 alongside the default version, when running Python from a command line or script etc. you will have to use the command "**python3**" to use the non-default Python 3.5.

2.2 CDM8 Development Software Installation

Once the pre-requisites detailed above have been satisfied, you can download the CocoIDE Software Archive from the 4COM1042 Platforms/Teaching Resources/CocoIDE_Software page on Studynet.

The archive is in .zip (or tar.gz for Linux/Mac) format and contains a compressed folder (or "directory" for non-Windows users) called CocoIDE. You should extract this folder to somewhere convenient (e.g. your normal home or users folder). For Windows, Mac OS and most Linux based systems, you can normally use the default File Manager program to navigate to the CocoIDE.zip (or tar.gz) file and either right click (or double click) on the file to view and/or extract to the location of your choice. For Linux based systems you may need to install an Archive Manager program (such as File-Roller) to view/extract the CocoIDE folder from the CocoIDE.tg.bz archive, or use the appropriate terminal commands if not running an Ubuntu or Debian based system). In all cases make a note of the full directory pathname to where the CocoIDE files are extracted to. For all systems, the recommended extraction location is the users normal "Home" folder or directory. **Note: The CocoIDE program will NOT run correctly unless all the files are EXTRACTED from the supplied archive first!**

For all systems, open your favourite File Manager program (E.g. Windows Explorer, Finder, Nautilus etc.) and navigate to where you extracted CocoIDE directory. In the CocoIDE directory you should see

a number of files including one called cocoide.pyw. This should automatically run the CocoIDE program. For Windows systems, you can either right click on the cocoide program file, and select “Send to Desktop (Create Shortcut)” to setup a desktop icon to easily run the program, or open up the Program Menu “Organise” option and add cocoide.pyw as a Program Menu item. For those with Mac, you simply drag the program icon to the Finder window and it should be appended to the lists of Programs automatically. Linux users can use whichever Program Menu Editor is installed to add cocoide to the main program menus and dock panels as required.

2.3 CDM8 Software Testing

You should now find that you can run cocoide.pyw from the GUI Program Menu. To test if the CocoIDE software runs correctly, open a new file and type the following lines:

```
# CDM8 Software Test Program
asect 0
br 20
end
```

You should find that the text changes colour as shown above as you complete each word. If you select the File Menu, and select “Save As”, save the file as “CDM8Test.asm”. Congratulations, you are now ready to develop programs for the CDM8 microprocessor! If the cocoide program not run, or colour highlighting does not occur – please read the following sub-section.

2.4 Troubleshooting the CocoIDE Software

If you have any problems after you have installed the CDM8 software, please try the following:

2.4.1 Running programs from the Command Line:

First confirm that Python can be run from the command line for your system. For Windows, the “Command Prompt” program can be found in the Accessories Folder of the All Programs option of the Start Menu. For Linux or Mac OS, the command line program may be called “Terminal”, “Terminal Emulator”, “Xterm”, or “Shell”. In any case open the program (you may want to create a Desktop or other short-cut to access this easily in future?). At the command prompt (varies, and can be changed to suit personal preferences – all systems!) type the command to run Python:

> python

(or if you have two Python versions (V3.x is preferred) installed on a Linux or Mac OS system so:

> python3)

If the Python program is correctly configured, you should see some initial text informing you of the Python version, then the prompt will change to “>>>”. Type exit() to quit the Python interpreter back to the command line again.

If you cannot find these in the GUI Program Menus, please (re-)check that they are installed correctly. Manuals and guides for installing and troubleshooting Python can be found on the Python website.

3 Quick Start: Using CocoIDE as an Integrated Development Environment for CDM8 Programming and Testing

Once you have installed CocoIDE (including Python and Logisim) you are now equipped for developing assembler programs for the CDM8 microprocessor system. This part of the manual is a tutorial that guides you step by step through the process of creating, compiling executing and testing your first CDM8 assembler program. Various useful features of CocoIDE are described and illustrated by example at suitable points in this tutorial.

The large left hand panel is a text editor which is oriented towards writing, loading and saving CDM8 Assembly language programs. Unlike a word processor package (such as MSWord etc.) it does not embed formatting, different font, text size and style information in the files which are produced. The text files produced are essentially stripped down versions, containing only bare textual information. These bare text files are therefore known as *"plain text"* files. Most programming languages expect their source code to be written by the human programmer as *plain text* and saved in a file on disk. For a plain text file, which is not meant to be run or compiled as a program, it is customary to give the file a name ending with nnn.txt (or nnn.TXT, where nnn is the name of the file), and the .txt "extension" indicates that the file is plain text (e.g. readme.txt). For specific programming language text files (also known as "source code"), it is usual to use a different extension that indicates what programming language the "source code" file is written in. For example, common program source code file extensions include .py (Python), .bas (Basic), .java (Java), .c (C), .cpp (C++), .lsp (Lisp) and there are many others. CocoIDE recognises any plain text file with an extension of .cdasm as a CDM8 assembly language listing file. Other features also allow it to be used as an IDE for CDM8 Assembler program development.

3.1 Writing and Testing CDM8 Assembly Language Programs

The general procedure for developing a CDM8 program is as follows:

- Start up CocoIDE: using the appropriate Program run method for your OS. Note, CocoIDE has standard drop down Menus and a Button bar (Fig 1, A) to access the most commonly used functions and options. The program also has three other main areas (Panes):
    - An Editor Pane (Fig. 1, B).
    - A Machine Code Listing Pane (Fig. 1, C).
- The CDM9 Emulator panes (Fig. 1, D, E and F) which displays the contents of CDM8 memory in table form (Fig. 1, D), the Watch pane (Fig. 1, E) and the CDM8 Registers (r0 – r3, PC, PS and SP, Fig. 1, F). Compilation error messages will normally appear in the Machine Op-code Listing pane.

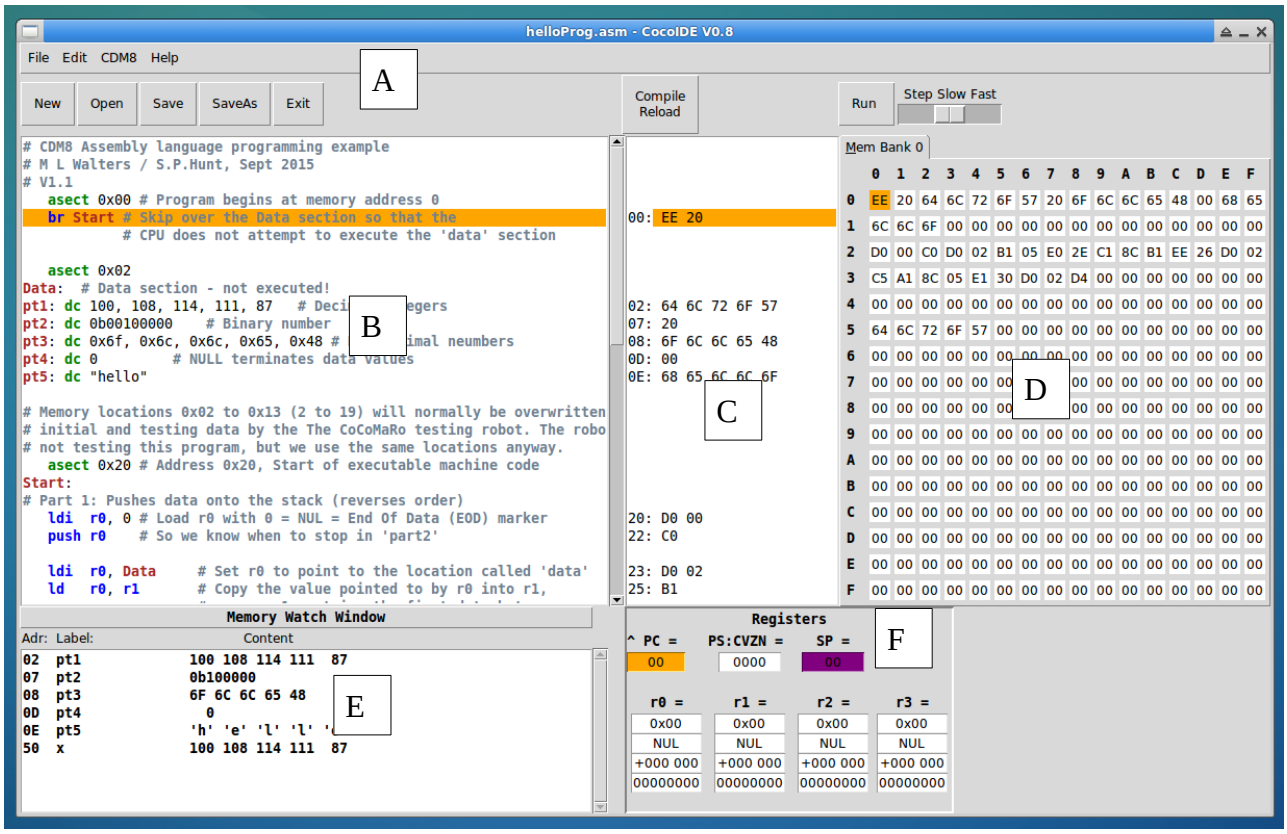

Fig 1: The main CocoIDE program window. The main areas are: A – the Button and Menu areas, B –the Editor Pane, C – the Machine Op-code Pane, D – the CDM8 Memory pane, E – the Watch Pane and F – the Register Pane .

- **Edit:** Write your (CDM8) program code. When opening an existing file with the extension .asm, the comments and assembly code statements will be highlighted in different colours. When saving your file, it will automatically append a file extension of <yourFileName>.asm.
- **Compile:** The Assembly code must then be "Compiled" to create a file which contains the "Machine Coded" version of your assembly code. To do this from CocoIDE, you can simply click the "Compile/Reset" button on the icon bar (Fig2, A). If all goes well, the Machine Code pane will be populated with your Assembly code translated to Machine Code in hexadecimal

form. The Memory Pane should show the same Op-codes loaded into the CDM8 processor memory. If there is an error in compilation, there will be a message which may provide some indication of what the error might be, and the line where the error was detected will be highlighted. Please note, that although an error may be detected on a certain line, the actual error may be found on a different line! You must locate and fix the error(s!) until compilation is successfully achieved before progressing to the next step.

- **Execute:** Once your assembly program has been successfully compiled and loaded, this does not mean that all the errors have been solved. So far, your program is syntactically correct, but there is no guarantee that your program runs correctly. You may find there are "Run-time" errors that cause the CDM8 processor to "Crash". Also the program may run ok, but not achieve the desired result. In order to test your program, the CDM8 Emulator Pane will can run your code and also allow you to inspect and interactively view how the CDM8 memory changes as your code runs. This is a powerful aid to tracking down and rectifying program run-time errors (De-bugging!). To run your program code in the CDM8 emulator, simply click on the "Run" button on the Icon bar. You can vary the speed of executing each program line using the Speed slider from "Fast", to "Slow", or "Step" mode which executes one line of program for each Click on "Run".

Fig 2: Geany IDE Icon Bar: The important Buttons are: A – Menus, B – Editor Shortcut Buttons, C – Compile/Reset (CDM8 Emulator), D – Run CDM8 Program, and E – Speed Slider.

More details of how to use the CocoIDE program to run and test your CDM8 Assembly Language programs are provided in the next section, which walks you through the process of creating, running and testing your first CDM8 program.

4 A First CDM8 Assembly Language Program

Type the program listing below in Fig 3 into the Editor pane of CocoIDE. Remember to save your program often as you type it in,

```
1  # CDM8 Assembly language programming example
2  # M L Walters / S.P.Hunt, Sept 2015
3  # V1.1
4          asect   0x00    # Program begins at memory address 0
5          br      Start   # Skip over the Data section so that the
6                          # CPU does not attempt to execute the 'data' section
7
8          asect   0x02
9  Data:   # Data section - not executed!
10         dc      100, 108, 114, 111, 87          # Decimal integers
11         dc      0b00100000                      # Binary number
12         dc      0x6f, 0x6c, 0x6c, 0x65, 0x48    # Hexadecimal neumbers
13         dc      0                               # NULL terminates data values
14
15 # Memory locations 0x02 to 0x13 (2 to 19) will normally be overwritten with
16 # initial and testing data by the The CoCoMaRo testing robot. The robot is
17 # not testing this program, but we use the same locations anyway.
18         asect   0x14    # Address 20, Start of executable machine code
19 Start:
20 # Part 1: Pushes data onto the stack (reverses order)
21         ldi  r0, 0      # Load r0 with 0 = NUL = End Of Data (EOD) marker
22         push r0         # So we know when to stop in 'part2'
23
24         ldi  r0, Data   # Set r0 to point to the location called 'data'
25         ld   r0, r1     # Copy the value pointed to by r0 into r1,
26                         # so now r1 contains the first data byt
27
28 while                  # Start of iterated (loop) section
29         tst  r1         # Is r1 zero? (= NUL= EOD)
30 stays   ne             # If not zero, keep iterating, else exit loop
31         push r1         # Push the r1 value onto the stack
32         inc  r0         # Add 1 to r0 to point at the next data item
33         ld   r0, r1     # Copy the next data item into r1
34 wend                   # Repeat again from while
35
36 # Part 2: Pops data from stack and overwrites the original data
37         ldi  r0, Data   # load r0 with start address of data
38 do                     # Start of program loop
39         pop  r1         # Get   byte of data from stack to r1
40         st   r0, r1     # Copy byte to data section, last byte first.
41         inc  r0         # Increment data pointer (r0)
42         tst  r1         # Exit loop if r1 is 0 (i.e EOD)
43 until eq               # Otherwise repeat from do
44
45 # Part 3: Tidy up
46         ldi  r0, Data   # Copy the address of the result (i.e stack pointer)
47                         # data into r0 for CocoMaRo / CoCheck to test.
48                         # Note, CocoMaRo not used for this exercise.
49         halt            # Stop the processor
50 end                    # End of program listing
51
52 # Note, when runing in emulator:
53 # Toggle view mode for row 0 of memory to see the message!
54 # To view the stack contents, toggle view mode for row f.
55
```

Fig 3: Test CDM8 Assembly Program Listing: HelloProg.asm

Do not worry if you do not understand all (any!) of the program statements and key words at the moment. Over the next few weeks you will get to know the CDM8 Assembly Language in more detail. Be aware that from any hash character (#) to the end of the line indicates a comment which is greyed out and will be ignored by the CDM8 compiler. It is good programming practice to use comments to explain what your code is doing, make notes about changes, problems etc. and to document your code. The bold words, highlighted in purple, black, green and blue are the actual assembly program statements which will be compiled into executable code. Try compiling your code as you enter every few lines, in order to eliminate any syntax errors as you go. Generally the assembler will indicate the line number and type of any errors detected. When you have typed the whole program in and compiled (Compile/Load icon) it successfully, it is ready to test in the CDM8 emulator pane. Note, handy key combinations are ctrl+z=undo, ctrl+y=redo, ctrl+c= copy, ctrl+v=paste, and ctrl+t=Toggle breakpoint (or double click in the Machine List opcode pane). These can mostly be accessed from the drop down menues, as well as by right clicking the mouse in the appropriate panel.

When you “Compile” your program successfully, the program will automatically load the “Compiled” program into the CDM8 Emulator panel's memory, ready for you to run and test (See Fig 1). For this first test, you need only concern yourself with the Memory View pane (Fig 1, D) and the Control Buttons (Fig 2, C, D and E). The Register pane (Fig 1, D) shows various registers used by the CPU – you will learn more about these over future weeks!

As the memory of the CDM8 is only 256 Bytes (0x00 to 0xff), the current memory is displayed as an indexed table arranged as rows (Most Significant Digit of memory address) and columns (Least Significant Digit memory address). At each memory row/column address location in the table, the current value of that memory location is shown. As you run your program you can see how the contents of the memory cells change. Note any memory cell that is written to by your program will be highlighted in red. If you have used a “**dc**” or “**ds**” instruction in your program, these memory locations will also be displayed in the Watch panel as well. You can either run one “Step” at a time and examine in detail what happens to the CDM8 memory and Registers at each program line, or “Run” your program at full speed, in which case you will just see the final memory contents. It is also possible to run your program in “Slow” mode, in which case you will see how the memory changes in slow motion. When your program has run to a “halt” instruction (D4) the “Compile/Reset” button will re-initialise your program in the CDM8 memory with the original program (memory image). The current machine instruction location to be executed is highlighted in a rather fetching shade of Yellow!.

By moving the mouse over the memory display table, a pop-up window will show the content of any memory cell in the most common data formats: Hexadecimal, ASCII Character, Signed and Unsigned Decimal, and as a Binary bitstring. By double clicking on a line in the Machine op-code display pane you can set “Break Points” where the program will pause execution. This useful for longer running programs, as program execution will pause at these points, allowing you to inspect the memory and register values. At any time you can also “Step” or “Slow” “Run” through following sections of code to the next “break point” or to end of the program (Note, machine instruction code d4 = “halt” instruction).

When you “Run” your program from CocoIDE, the Editor and Machine Code panes will highlight the apropriate lines of the listing of your original assembly code and machine codes. This will provide an indication how and where your program statements and data are actually loaded into the CDM8 processor memory, and allow you to see the current machine code being executed and see what it is in

relation to your original program listing.

Normally you will receive and submit your own personally tailored series of programming exercises by email. More details will be provided in timetabled sessions.