

COMP2300

Tutorial / Laboratory 04 - Introduction to PeANUt

Semester 1, 2007

Week 5 (21 - 24 March)

Preparation

It is expected that you would have looked over your lecture notes of the PeANUt module as well as read the relevant parts of the *Specification of the PeANUt Computer* before doing this session.

Don't worry if you have just a partial understanding of PeANUt. It is the purpose of this session to help you acquire familiarity with the fundamentals of the PeANUt machine.

You should bring the COMP2300 reading brick *Specification of the PeANUt Computer* to this and all subsequent tutorial / laboratory sessions (as well as all the PeANUt lectures).

Tutorial Exercises

In the following box you can see a simple PeANUt machine language initialisation file (this file is also available in the directory /dept/dcs/comp2300/public/lab4/ on the student machines under the name lab4a.mli).

```

; lab4a.mli - Simple PeANUt machine language initialisation file
;
START a10           ; initialise the PC to a10.

AT a10             ; from address a10 on, initialise with the following:
000 001 0 000 000 001 ; a10: this goes in a10
000 011 0 000 000 010 ; a11: this goes in a11
110101 0 000 000 001 ; a12: this goes in a12 (this instruction must always
;                   be at the end of an instruction sequence)

```

1. Examining a .mli file

All text to the right of the ;'s are comments. The PeANUt conversion to image file process ignores these comments, they are only there to help humans to read the file. The remaining information (the text to the left of the ;'s) will be used to initialise the PeANUt.

- o The first line (START a10) will result in the PeANUt initialising its **PC** (program counter) with a10 (memory address 10₈, 10 octal).
- o The **PC** holds the location (in memory) of the next instruction to be executed, so initialising the PeANUt's **PC** to a10 will result in the instruction at location a10 in memory being executed first when the machine is started.
- o The second line (AT a10) will result in the PeANUt to initialise its memory cells from a10 onwards with the data in the lines that follow.
- o The next three lines are the information (16 bits each) that the PeANUt will initialise the memory cells a10, a11 and a12 with.

2. Understanding PeANUt instructions

Write down the answer to the following questions.

In the program lab4a.mli, the information to be stored at memory locations a10 onwards are binary numbers (with 16 digits, i.e. 16 bits) that the PeANUt will interpret as instructions.

1. Open your PeANUt manual on page 3.
2. Since the **PC** (program counter) will be initialised to a10, when the PeANUt begins executing it will fetch the first instruction from a10. The data at a10 should thus be viewed as a PeANUt instruction. Examine the data that is to be used to initialise memory a10:

```
000 001 0 000 000 001
```

Note that the PeANUt ignores spaces, it treats "000 001 0 000 000 001" as "0000010000000001". The spaces are only in the instruction to help make it easier for human readers to understand.

- What **format** is this instruction? (Section 2.4)
- What is the **mode** of this instruction? (Section 2.4)
- What is the **opcode** of this instruction? (Section 2.4)
- What is the **opspec** of this instruction? (Section 2.4)
- What is the **operand** (OP) for this instruction? (Sections 2.4, 2.5)

3. What will be the effect of this instruction, when executed?
4. Write an appropriate comment for this instruction (similar to the comments given in the programs presented in the lectures).

5. Repeat this process (steps 2, 3 and 4) for the data that will be used to initialise `a11` and `a12`.
 6. What is the effect of executing the three instructions in this machine language initialisation file?
3. **Writing a simple PeANUt machine language program**

Write - on paper - a PeANUt machine language program that does the following:

- o The memory cell `a0` should be initialised to contain the decimal value `42`.
 - o The memory cell `a1` should be initialised to contain the decimal value `3`.
 - o Your program should start at address `a20`.
 - o Your program should multiply the number stored in memory cell `a0` with the number stored in memory cell `a1`, and then store the result into memory cell `a3`.
 - o Your program should be correctly terminated using the appropriate `trap` instruction.
 - o Write appropriate comments for each instruction of your program.
4. (referring to [Lecture P1](#) (p. 9) and [Lecture P2](#) (e.g. p 9)), write down the sequence of internal steps in the PeANUt instruction cycle for the multiply instruction in the above program. Assume the address of the first instruction is at `a10`. Make sure you include the instruction fetch stage. Give the contents of the **CI** (in binary), and the **MAR** and **MDR** each time these change.
 5. Suppose it is desired to increase the PeANUt memory size to 4096 cells. What problems would this create in the instruction set? Discuss ways in which this problem could be mitigated.

Part 2 - Laboratory Exercises (using the PeANUt simulator)

The PeANUt simulator is a tool that allows detailed viewing of the operation of the PeANUt computer. The objective of this laboratory is to introduce you to the PeANUt simulator and the basic operation of the PeANUt computer. Try to answer all questions requiring explanation yourself first, but don't spend long on any one - rather ask your tutor for help.

Twenty minutes or so before the end of your session, if you have not done so already, get to the [Your multiplication program](#) part and complete that before your session finishes. Any remaining parts can be completed in your own time.

1. Preliminaries

1. In your `comp2300` folder create a folder called `lab4`.
2. Copy the file `/dept/dcs/comp2300/public/lab4/lab4a.mli` into your `lab4` folder.

`lab4a.mli` is a text file that will be used to initialise the PeANUt machine, however, the text (ASCII characters) in this file cannot be used *directly* to initialise the PeANUt. We have to convert this file into a **binary file**. The file created by this process is called an image file (`lab4a.img`), which can be used directly to initialise the PeANUt.

Open your copy of `lab4a.mli` in your favourite text editor and you should see the file as shown in the box in Part 1 above.

2. Starting the PeANUt

1. Start a terminal window as you have done in the previous laboratories.
2. You have to log onto the machine `iwaki` in order to be able to start PeANUt.

Important:

```
Peanut is currently only available on the student Solaris
server - please ssh logon to 'iwaki' and start Peanut
in a terminal window with:
```

```
ssh -X iwaki
Peanut &
```

3. Once you have logged onto `iwaki` you will have to change into your `comp2300` folder and then into the folder `lab4`.
4. Note first that whenever you pause the mouse cursor over one the **buttons** on the PeANUt simulator (such as the one labelled **QUIT**), you get some *balloon help* as a little yellow tag.
5. Note also that there is a tab called **Help**. Play with it, if you must, but make sure you go back to the **Memory** display before proceeding.
6. Convert your file `lab4a.mli` from MLI format to image format (as `lab4a.img`). Do this by left-clicking of your mouse over the **MLI-IMG** button and then select `lab4a.mli` from the list in the **File Selection** window. (You may have to find the correct folder first.) Left-click over the **Mli2img** button.
The PeANUt simulator has now registered that you are interested in `lab4a.mli` and has translated this into an appropriate image file called `lab4a.img` for the PeANUt machine. The next step is to ask the PeANUt simulator to initialise the PeANUt using this file.
7. Initialise the PeANUt (left-click over the **load IMG** button). A **dialog box** should pop up. This box (with a banner saying **Load IMG**) will present you with a list of files from which you select `lab4a.img` by using left-click on the name. Note that the Selection window gives the full path name of the `lab4a.img` file. Now use left-click on the button labelled **Load IMG** to load the file. The PeANUt machine then initialises itself according to the initialisation specified in the binary file `lab4a.img` (which is a binary representation of the initialisation specified in the text file `lab4a.mli`).

3. Experimenting with the PeANUt

1. Examine the *state* of the PeANUt. Make sure that the **Memory** display tag is selected so that you have a view of memory for the following tasks.
 - What are the values of memory locations `a10` to `a12`?
 - What is the value of **PC** (the program counter)? This is indicated in several ways. There is the **PC** register shown in the part of the machine called **Console**. Then there is the bottom 10 bits of the **PSW** shown in the part of the machine called **Registers**. Finally, the largest of the three memory displays starts out tracing the program counter and its value is in a box at the top of the memory display.
 - What is the current instruction? The **CI** (current instruction) register is set during every instruction execution cycle to the instruction that is executed in that cycle. Between instruction execution cycles it will therefore refer to the last instruction to be executed.
 - What is the next instruction to be executed? Get this from looking in memory at the location where the **PC** points.
 - What is the value of the **AC** (accumulator)?
 2. Execute the next instruction (left-click on the **Step** button). Balloon help will enable you to identify the **Step** button.
 3. How has this changed the *state* of the PeANUt?
 - What is the current instruction now (it is in the **CI** register)? What is the address of the next instruction (the value in **PC**)? What is the next instruction?
 - What is the value of the **AC** (accumulator)? Is this what you expected to happen?
 4. Execute the next instruction (left-click on the **Step** button again).
 5. How has this changed the *state* of the PeANUt?
 - What is the next instruction now?
 - What will happen when the next instruction is executed? (hint: page 8 of the PeANUt specification might help...)
 - What is the value of the **AC** (accumulator)? Is this what you expected to happen?
 6. Execute the next instruction.
 7. What has happened?
 - There should be a status indication in the bottom right corner of the PeANUt window. What does it say?
 - Has anything happened to any of the buttons on the PeANUt simulator? If so, why do you think this has happened? Try balloon help on the buttons that have changed colour.
 - What values are in the **AC**, **PC**, and memory locations `a10` to `a12`?
 8. Is this what you expected to happen? If not, look carefully at the instruction in memory location `a12` and its definition on pages 5 and 8 of the PeANUt specification.
4. **Other ways of executing on the PeANUt**

In the previous section, you used the **Step** button on the PeANUt simulator to make the PeANUt execute the instructions in its memory one at a time. When you work with larger programs, you will find that this method is often too tedious. There are other ways of making the PeANUt execute the instructions in its memory.

1. Re-initialise the PeANUt (left-click on the **Reset** button). This will place the PeANUt in exactly the same state as the initialisation file specifies (don't forget that the exercises that you did previously changed a few things, including the **PC** and **AC**). Note that you can also re-initialise PeANUt by pressing the **Load IMG** button and re-loading an image file.
2. Left-click on the **Run** button. What happened?
3. Compare the state of the PeANUt now with its state after you stepped through the instructions (check the contents of **CI**, **AC**, **PC** and memory locations `a10` to `a12`).
4. Re-initialise the PeANUt (**Reset**).
5. Left-click on the **Animate** button. What happened?

Further experiments

Now that you know how to use the PeANUt simulator to initialise and execute instructions on the PeANUt, you can start experimenting with initialisation files.

1. Experimenting with the PC

In this section, you will experiment with the **PC**. This should help you understand the important role of the **PC** and why initialising the **PC** is a crucial part of the initialisation process.

1. Open your `lab4a.mli` file in a text editor.
2. Change the `START` point to `a11`. (Don't change the argument of the `AT` clause, however.)
3. Store as a new file, `lab4b.mli`.
4. Convert `lab4b.mli` into an equivalent `lab4b.img` file using the PeANUt as you have done before.
5. Initialise the PeANUt (left-click the **load IMG** button).
6. What is the value of the **PC**? What is the *next instruction*?
7. What do you think the state of the **AC** will be after the PeANUt has completed execution?
8. Step through the execution by clicking on the **Step** button.
9. Is the final state what you expected? What has happened?

10. Modify the **START** point of `lab4b.mli` to `a12`. Save the file.
11. Re-initialise the PeANUt. You will need to create a new image file as done previously.
12. What do you think will happen when you execute?
13. Step through the instructions. What happened?
14. Modify the **START** point of `lab4b.mli` to `a13`. Save the file, then convert it into an image file.
15. Re-initialise the PeANUt.
16. Step through the instructions. What happened?
17. Modify the start location of the data (`AT. .`) in `lab4b.mli` to `a13`. Save the file.
18. Re-initialise the PeANUt.
19. Step through the instructions. What happened?

2. Other instructions

The PeANUt is capable of executing all the instructions that are listed in the table on page 5 of the PeANUt manual. For each of the following exercises create a new `.mli` file (`lab4c.mli`, `lab4d.mli`, `lab4e.mli`...). Once you have created the appropriate initialisation file (`.mli` file) test it by converting it into an image file; and then loading, initialising and executing it on the PeANUt.

1. Create a file (based on `lab4a.mli`) that subtracts 4 from 7.
2. Create another file (based on the above) that subtracts 4 from 7 and then stores the result in `a0`.
3. Create a file that adds the value in `a2` to the number 5 and stores the result in `a0`.
4. Create an initialisation file that instructs the PeANUt to print out the letter 'A'. You will need to use the `trap #3` instruction, see pages 5 and 8. The ASCII code for 'A' is 101_2 ; you will need to convert this to binary.
5. Create another file (based on the above) that prints out your initials. You will need to find the ASCII codes for your initials (see the [Relevant Links](#) section of the *COMP2300* web site).
6. Create a file that instructs the PeANUt to print out the character whose ASCII code is found at memory address `a20`. You might like to initialise `a20` with an appropriate value.

3. Data and instructions

In the PeANUt, there is little distinction between data and instructions. In this experiment you will see how fine that distinction can become.

1. Create a new file (based on one created earlier) that subtracts 4 from 7 and then stores the result in `a13`. As before the program should start at address `a10`.
2. Load and execute the instruction sequence.
3. What happened?
You should have found that execution stopped when `a13` was reached. This was because the data in `a13` is not a legal instruction. You could have done some other arithmetic which resulted in a legal instruction appearing in `a13`. In that case, that instruction would have been executed.
4. Create another file that writes a *legal* instruction into `a13` prior to `a13` being executed.

4. Your multiplication program

Open a text editor and type in your tutorial program that multiplies 42 with 3, then save it under the file name `lab4m.mli`. Convert it into an image file, then load it into PeANUt and run it. If it is not correct (i.e. if the result of 42×3 is not written into memory cell `a3`), debug your program.

5. Addressing modes and repetition

Have a look at the program `printword.mli` from [lecture P2](#) and available from the COMP2300 [lecture P2 programs page](#).

1. Write a program `printname.mli` that prints your name.
2. Now modify your program so that the characters of your name are stored in memory, and your program uses *direct mode* (instead of immediate mode) to load the characters from memory and print them.
3. Think about how you can use a loop (as discussed in the lectures) to print out your name. Modify your program accordingly.
4. Finally, think about how you can use *indirect addressing* to load the characters from memory in a smart way.