Workshop 04 - Hack CPU

Workshop 04 - Hack CPU

The purpose of this workshop is to show you how the Hack CPU could be built. This is based on **project 05** (https://www.nand2tetris.org/project05) from the Nand2Tetris course. This workshop uses builtin (working) versions of the chips you have already seen.

We have created a **Nand2Tetris Project 05** (https://myuni.adelaide.edu.au/courses/54311/pages/nand2tetris-project-05) assignment in the **Web Submission System** (https://cs.adelaide.edu.au/services/websubmission) so that you can attempt this project independently from this workshop and be awarded up to 5 additional participation marks.

Participation Marks

Up to 5 participation marks are available for every workshop, 2 for preparation, 1 for attendance and 2 for completing an activity, subject to the conditions described on the **Participation Marks** (https://myuni.adelaide.edu.au/courses/54311/pages/participation-marks) page.

Attendance

One participation mark will be awarded if your workshop attendance is recorded and you make a submission to the Workshop 04 assignment in the Web Submission System by **Friday 11.55pm** of week 4. **Do not** leave a workshop until you have checked your attendance mark.

If you are using a CAT suite computer running Linux in a timetabled workshop, your presence should be automatically recorded when you visit our **practical marker** (https://cs.adelaide.edu.au/services/pracmarker/). If your attendance mark is not displayed when you visit our **practical marker** (https://cs.adelaide.edu.au/services/pracmarker/), your presence will need to be manually recorded. **Do not click** on the "flag me for marking" button until a supervisor is standing next to you and is ready to record your presence. The flag is only to speed up the data entry and is only visible for 30 seconds.

Workshop 04 Background Reading

Read the third chapter of the text book and the **project 05** (https://www.nand2tetris.org/project05) description from the Nand2Tetris course.

Workshop 04 Preparation Activity

Two participation marks will be awarded for completion of this preparation activity if it is completed at least 10 minutes before the first timetabled workshop of the week, see **Participation Marks** (https://myuni.adelaide.edu.au/courses/54311/pages/participation-marks) for details.

Note: in example commands % is the shell's prompt, it is not part of the command.

Note: this workshop assumes that you have already created directories for every assignment, workshop project and exam in your svn repository, as described on the **Startup Files for Workshops and Assignments** (https://myuni.adelaide.edu.au/courses/54311/pages/startup-files-for-workshops-and-assignments) page.

Note: the web submission system will record 0 marks for completing this activity, the 2 marks are awarded later if and only if your attendance is recorded.

1. If required, checkout a working copy of the workshop04 directory from your svn repository.
2. Change directory to the working copy of the workshop04 directory.
3. Copy the newest zip file attached below into the updates sub-directory and add it to svn. Do not unzip the file.
4. Run the following command to place the workshop's startup files in the correct locations:

```
% make install
```

5. Add the .hdl files to your svn repository:

```
% svn add *.hdl
% svn commit -m "Workshop 04 Startup Files"
```

6. Goto the Web Submission System and make a submission to the Workshop 04 assignment. A successful submission that passes the preparation tests will complete the preparation activity.

Workshop 04 Activity

Two participation marks will be awarded for completing the following activity. This need not be completed during the workshop but no participation marks will be awarded if the activity is not completed by **Friday 11.55pm** of week 4.

After completing each of the following steps, commit your changes to svn:

```
% svn commit -m workshop04-activity
```

After each commit, go to the Web Submission System and make a submission to the Workshop 04 assignment. A successful submission that passes the **Memoryx** tests will complete the workshop activity.

Step 1

Build and test the Memory from project 05 of the Nand2Tetris course. This requires you to edit the HDL file **Memory.hdl**.

The Memory is made up of 3 parts, 16K words for data, 8K words for the screen memory and 1 word for the keyboard register. You should use the builtin chips **RAM16K**, **Screen** and **Keyboard** to implement these components.

One issue to consider is what to do when a request is made to read / write part of the memory after the keyboard register or to write on the keyboard register. You can simply choose to ignore the write requests but it is good practice to return 0 for the invalid read operations.

You can test your Memory using either the **Memory.tst** or **Memoryx.tst** scripts. The **Memory.tst** script only works if run inside the Hardware Emulator's GUI because if reads from the keyboard device. The **Memoryx.tst** script does not test the keyboard and can be run from the command line:

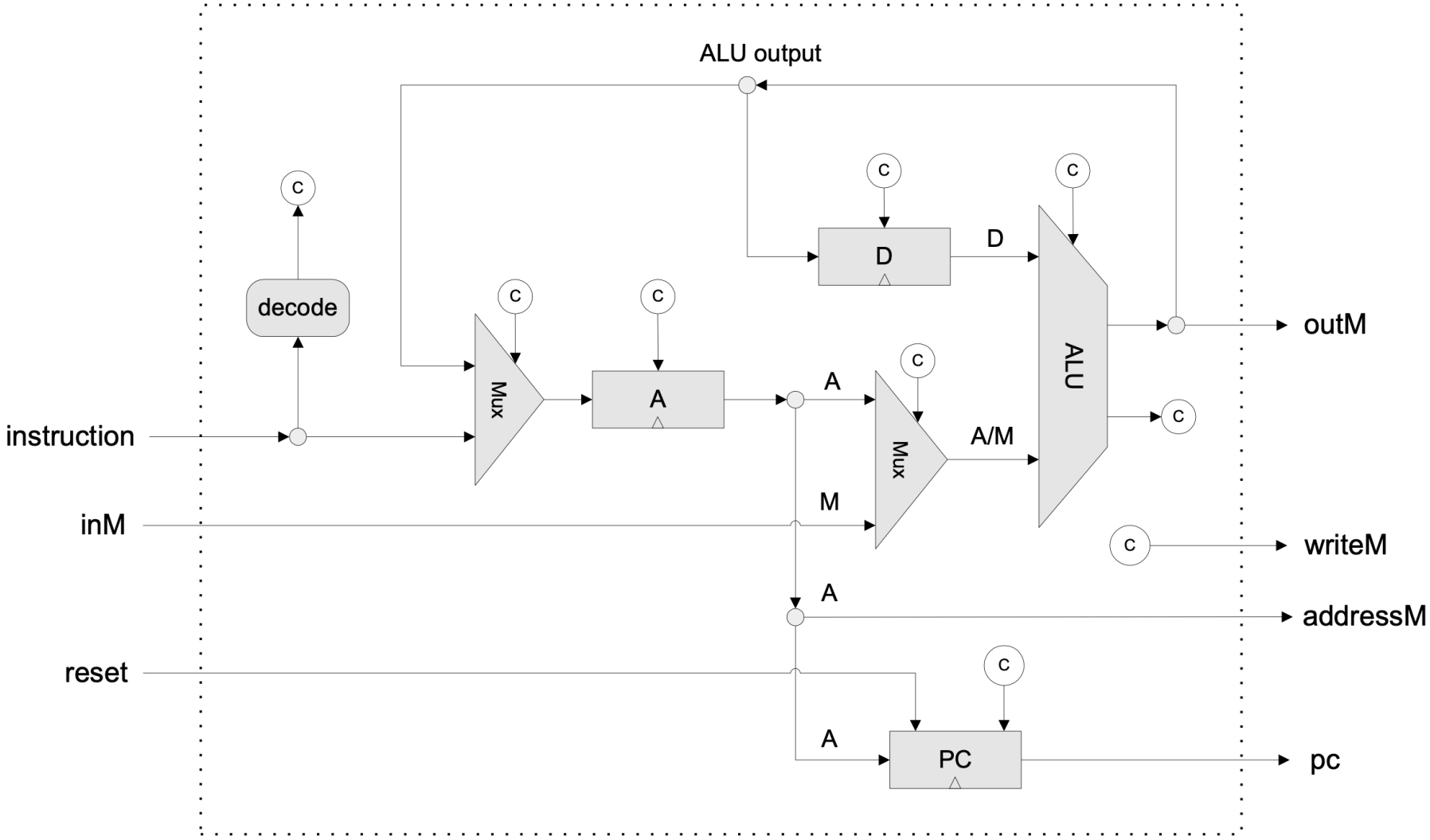
```
% HardwareSimulator.sh Memoryx.tst
End of script - Comparison ended successfully
```

The web submission system does not use the **Memory.tst** script because it cannot run the GUI.

Step 2

Build and test the CPU from project 05 of the Nand2Tetris course. This requires you to edit the HDL file **CPU.hdl**.

The main internal components of the CPU should be implemented using the **PC**, **ARegister**, **DRegister**, **ALU** and **Mux16** builtin chips. The challenge is how to wire up the incoming instruction wires to the inputs of these chips marked with © in the following diagram:



A good starting point is to print a copy of this diagram and write on the names of all the internal wires. This will make it much easier to check that you have wired up the chips correctly. Once you have named the wires, start by connecting up the chips with the named wires.

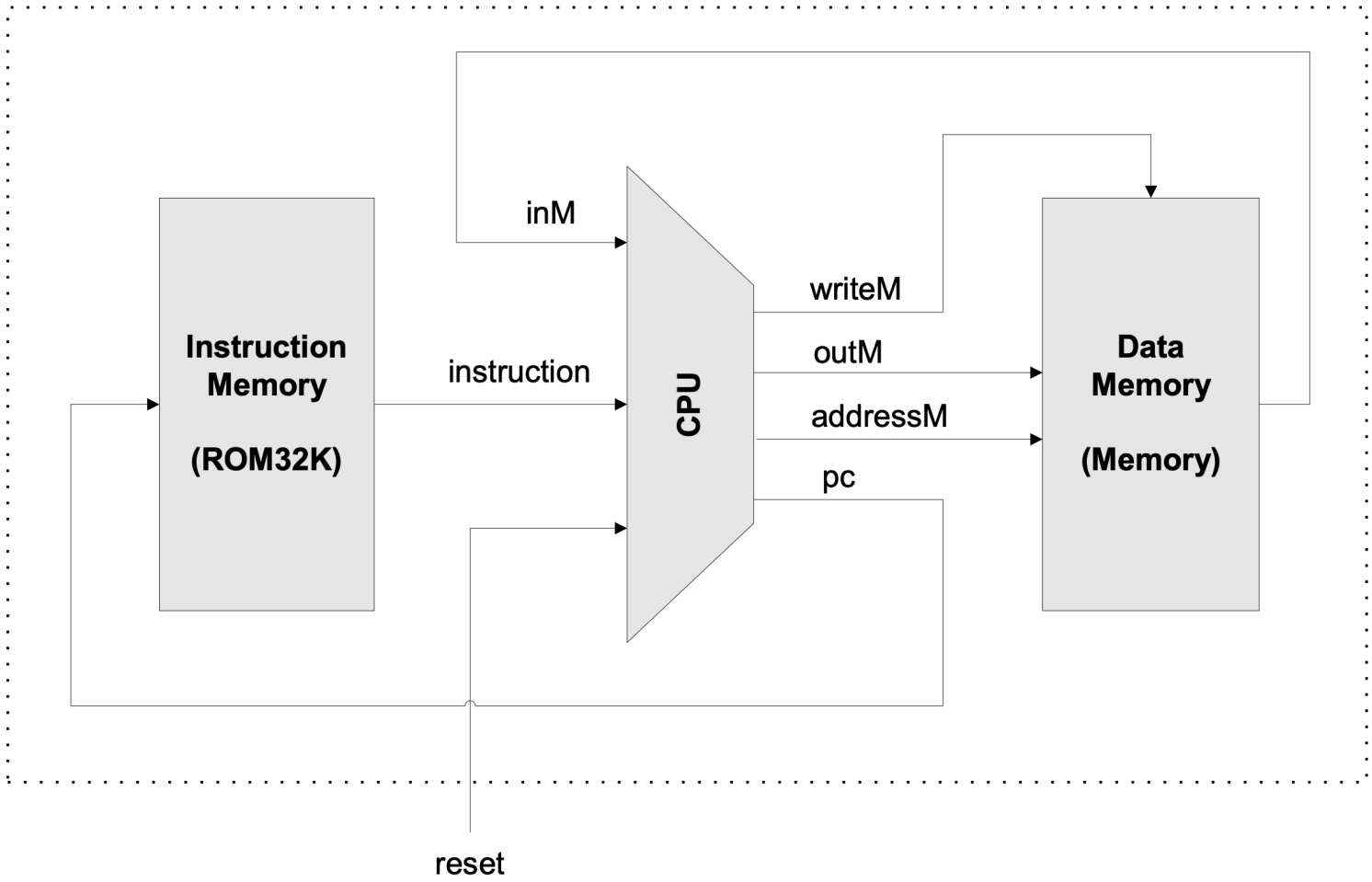
Next you can start working out how to connect the instruction wires to the © inputs. We suggest starting with the two **Mux16** chips first, then the **A** and **D** registers, then the **writeM** output, then the 6 **ALU** control inputs and finally the **PC** inputs.

1. The left **Mux16** either allows the current instruction or the output from the **ALU** to be connected to the input of the **A** register. This choice depends on whether we have an A-instruction or a C-instruction. The most significant bit of an instruction, instruction[15], is 0 for an A-instruction or 1 for a C-instruction. You can connect this to control the *sel* input of the **Mux16** chip.
2. The right **Mux16** either allows the contents of the **A** register or the data from memory, **inM**, to be fed into one input of the **ALU**. Given that we do not care what the **ALU** does in an A-instruction, we can simply use the *a-bit* of a C-instruction, instruction[12] to control the *sel* input of the **Mux16** chip.
3. The **A** register input is its *load* signal which must be set only if we have an A-instruction or a C-instruction that is writing to the **A** register.
4. The **D** register input is its *load* signal which must be set only if we have a C-instruction that is writing to the **D** register.
5. The **writeM** signal must be set only if we have a C-instruction that is writing to memory.
6. We do not care what the **ALU** does in an A-instruction so the control inputs in a C-instruction, wires 6 to 11, can be directly connected to the **ALU** control inputs.
7. The last step, wiring up the **PC** is the most complex. If we have an A-instruction or a C-instruction that does not perform a jump, the **PC** should just increment by 1. If we have a C-instruction that might jump we need to use the zero (**zr**) and negative (**ng**) outputs from the **ALU** combined with instruction wires 0 to 2, to work out if a jump must be taken, that is, do we set the **PC** register's load input?

Step 3

Build and test the Computer from project 05 of the Nand2Tetris course. This requires you to edit the HDL file **Computer.hdl**.

The following diagram shows how the Computer is wired using the **CPU**, **Memory** and the builtin **ROM32K** chips:



Once this chip is complete, you can try testing your computer using the programs in the **Add.hack**, **Max.hack** and **Rect.hack** files.

Startup Files

The newest of the following zip file(s) must be placed in the updates sub-directory and added to svn. When make is run, the newest zip file in the updates directory is used to update the startup files. Any files you are required to edit will not be updated but, a copy of the latest version of those files will be placed in the sub-directory originals.

- **workshop04-20200815-151729.zip (https://myuni.adelaide.edu.au/courses/54311/files/7247294/download?wrap=1)**
- workshop04-20200814-101645.zip

- workshop04-20200724-193248.zip