ASIP Meister and ModelSim

Motivation and introduction

In this exercise, you will be introduced to *ASIP Meister*, *ModelSim* and our special directory structure, which makes it easier to combine *ASIP Meister*, *ModelSim*, *dlxsim* and the later introduced tools. First, you have to read chapter 2 from the laboratory script to understand the directory structure. Read this chapter extremely carefully, as this one is very important for every task you have to do in the laboratory. Then you will create your first own *ASIP Meister* project and you will go through the *ASIP Meister* User Manual and the Tutorial to get used to *ASIP Meister*. Afterwards you will simulate a given Assembly Code with *dlxsim* and with *ModelSim*. Therefore, you will have to read the chapters 8.3 (for preparing the simulation files for *dlxsim*) and 5 (for using *ModelSim*) of the laboratory script. For every part, that starts like "a)", "b)", … you have to write an answer and mail it with a CC to your group members to asip00@ira.uka.de and use the topic "asipXX-Session2", with XX replaced by your group number.

For the session in the next week read the chapters 3, this time including 3.2.2 and 3.2.3.
Please **access remotely** i80pc77 or i80pc78 to perform the required simulations, tasks, etc.

Exercises

1) Preparing the Project

After you have read chapter 2 of the laboratory script you can start creating your own directory structure in your home directory. Create a copy of the *TEMPLATE* directory from ~asip00/ASIP-MeisterProjects/ for your first project. Call your project directory "dlx_LaboratoryBasis" and adjust the *env_settings* correspondingly.
Copy the given (see ~asip00/Session02/) *ASIP Meister* file *dlx_LaboratoryWithoutAdd.pdb* into your project directory and rename it to *dlx_LaboratoryBasis.pdb*. Create a directory for the example application in your *Applications* directory, name this directory "LoopExample" and copy the given assembly file into this directory. Now your first project directory is completed and you can start using it in the next exercises.

2) Using ASIP Meister

Start *ASIP Meister* for the given basis CPU: "`ASIPmeister dlx_LaboratoryBasis.pdb &`". Moreover, do not forget to start *ASIP Meister* in your project directory, because it will create the *meister* subdirectory for the created files where it was started and the *meister* subdirectory is expected to be in the project directory.
Read the User Manual and the Tutorial for *ASIP Meister* to get used to the GUI. You can find the related files in the */Software/epp/ASIPmeister/share* directory and in the *tutorial* subdirectory. Read systematically through both files simultaneously and play with the specific parts of the GUI for which you are currently reading the user manual and tutorial. If you anyhow configure something wrong, then just reload the original file. The most important parts for the later work are the *Resource Declaration*, *Instruction Definition*, *Behavior Description* and *MicroOp Description*, so have a detailed look at them.

a) What is the difference between the *MicroOp* implementation for *jr* and *jalr*. What is happening in hardware and when is it happening. Why has *link* to be global?

b) What is the difference between the *MicroOp* implementation for *subi* and *subui*. Why this difference is only needed for the immediate instructions?

3) Adding an Instruction

The *add* commands (*add*, *addu*, *addi*, *addui*) have been removed from the example CPU which we gave you. Go ahead and insert them into the CPU. You will have to work with the *Instruction Definition*, the *Behavior*- and *MicroOp*-Description. Use the following *opcode/func*-values: *add*:000000/00000100000, *addu*:000000/00000100001, *addi*:001000/-, *addui*: 001001/-. If you are unsure about a detail, then have a look at the corresponding *sub*-commands. **However, if you only copy-and-paste everything without learning anything, then you will pay for this in later sessions** where you will not have a template to copy-and-paste from. Create the hardware and the software description for the modified CPU. Make sure, that you do not have any mistakes in the *add* instructions, as this will cause problems in the later sessions.

4) Simulating with dlxsim

Now you have a CPU that is able to execute the given example code *6_for.s*. This code is similar to the code you created in the last session for exercise 6, but this version also takes care about removing the data-dependencies for the DLX pipeline with *NOP* commands.
First, simulate the assembly code with *dlxsim*. Therefore, make a copy for the "Makefile" existed in the TestPrint folder to your LoopExample subdirectory. Then, go to the LoopExample subdirectory inside your Applications directory and execute "make sim".

Use the public-key system described as the last point in chapter 2.2 to avoid typing your password multiple times when executing mkimg.

After "make sim" finished, a new subdirectory called "BUILD_SIM" contained the generated files will be created in your current directory. There is a special ".dlxsim" file for dlxsim and there are TestData.IM and TestData.DM for ModelSim. Simulate ".dlxsim" with dlxsim by typing: "make dlxsim"

a) How many *NOP's* have been executed regarding to the total number of instructions and regarding to the total number of executed cycles (in percentage)?

b) Remove the *NOP* after the "add r6, r6, r7" instruction and simulate the program again (*mkimg_fromS*, *dlxsim*, …). Understand the warning, that *dlxsim* will print and have a look to the result array in *dlxsim*. What is the difference to the correct result and why has it been changed in such a way?

c) The real CPU does not have a *NOP* instruction. Into which instruction is a *NOP* translated and what parameters (register and immediate values) does this instruction get as input? Have a look into the *TestData.IM* to find out which *opcode* is used to implement a *NOP*.

5) Simulating with *ModelSim*

Now we simulate the assembly program together with the real CPU VHDL files. Read chapter 5 in the laboratory script, and create a *ModelSim* project for your CPU and simulate the program. Compare the created *TestData.OUT* with the results from dlxsim. Try to understand the program flow with looking at the *TestData.IM* and the *register write* part from *ModelSim's waves*.

a) Write a short description what is happening in the first data memory access. Mention all the data bus signals and the final register write back. In the upper part of the *ModelSim wave*, you can see a clock counter. Use this value to describe when something is happening. Sometimes things happen in the middle of a clock cycle (i.e. falling edge); mention this, too.

Note: For the Modelsim Version 6.6d, you have to browse to the following path "/Software/ModelSim/ModelSim_6.6d/modeltech/modelsim.ini" for the "Copy Settings From".