Simulation of DLX-Assembly Programs

Motivation and Introduction

The main goal of these exercises is to get used to DLX-assembly programs. You are supposed to have a look at some code examples. Every example shows a basic operation. Examine the code, simulate it with *dlxsim* and answer the corresponding questions. The assembly code is available in the directory *~asip00/Session1/*. *Dlxsim* is available in the directory */Software/epp/dlxsim_Laboratory/*. Before using *dlxsim* you have to read the chapter 3 in the laboratory script, but you can skip the sub-chapters 3.2.2., 3.2.3 and 3.3.2 for now. You can copy *dlxsim* to your directory or you can start it in the */Software/epp/…* directory.
You need to copy a "Makefile" file from the "TestPrint" folder to your applications folder. The Makefile has been prepared to help you in doing the tasks during this Lab. By typing "make", you can see an example of how it can be utilized together with how parameters can be passed to a tool when it is required. For exercises of today, you have to configure the pipeline delay to 1 as it is described in chapter 3.2.1 of the laboratory script.
For instance, to run the first example you can type the following:
"make dlxsim DLXSIM_PARAM="-f1_Arith.s -da0 -pd1"".
The below printed exercise numbers correspond to the numbers from the assembly file names. If you are used to assembly code then don't get bored with the first 3 exercises, the numbers 4 to 6 are trickier. For every part, that starts like "a)", "b)", … you have to write an answer. This answer shall mainly prove that you have understood the problem, so you can make your answers short, but they still have to answer everything, that was asked. Mail your answer with a CC to your group members to asip00@ira.uka.de and use the topic "asipXX-Session1", with XX replaced by your group number.
For the next week read the chapters 2, 5 and 8.3 (without 8.3.x).

Exercises

The exercise numbers correspond to the given assembly files in the ~asip00/Session 1 directory.

1) Basic Assembly Instructions

Understand the functionality of every instruction and understand the values of every target register after the program run has completed (see the dlxsim chapter for reading register values in dlxsim simulation). Check the amount of cycles needed to execute all instructions.

a) What is the reason for this high number of cycles? Which instruction causes that behaviour and why is it doing so?

b) What is the purpose of the "trap #0" instruction?

2) Memory Access

a) Explain the goal of the instruction combination *lhi / ori*. What is generally (so: not only for this specific example) the register value after *lhi*, what is it after the additional *ori*?

b) Why is it in general not possible to omit the *lhi* instruction, although it would be possible in this special example?

3) Branches

a) Which high-level control structure (e.g. 'call subroutine', …) is implemented in this example code?

b) What is computed with this example? r4 = function(r1, r2);

c) Look at the *NOP* instructions and explain why they are placed there.

d) Which meaningful instruction (i.e. an instruction that helps executing the code; not just any dummy instruction) can replace the first *NOP* instruction without changing the final result and why does it not change the final result?

4) Loops

a) What is computed with this example? r3 = *function*(r1, r2). Debug the application step-by-step with the capabilities of dlxsim.

b) The approach to compute the function in the way this example is doing it has two specific names. Do you know those names? (One is founded by the main operations, while the other is founded historically. You either know the names or you don't. If you don't know both names, you may guess)

c) In general, how often is the loop maximally executed? How the input data has to look like to get this maximal number of iterations?

5) A High Level Structure

a) Which high-level control structure do you recognize? Explain the purpose of the instruction-block between "addi r3, r0, 2" and "jr r4".

b) Why do you have to shift by value 3? Explain it with a close view to the body of the control structure and pay attention to the addressing mode of the DLX processor.

c) What are the general differences between branch and jump instructions in the DLX instruction set (also have a look at the different instruction formats to find a part of the answer)?

6) Writing Assembly Code

Consider the following fragment of C code:

```
for (i=0; i<10; i++) {
  A[i] = B[i] + 5 + C;
}
```

Assume that A and B are arrays of 32-bit integers in the memory and that C is a 32-bit integer, that is also placed in memory (currently not available in a register). Write the corresponding assembly code for the DLX processor. Your code should be as short as possible (in #instructions) and it should compute the same result.

a) Attach the assembly file to the mail.

b) How many assembly instructions do you need altogether?

c) How many clock cycles does your code need to execute?

d) How many memory accesses are performed altogether?