Technische Universitaet
Chemnitz
Fakultaet fuer Elektrotechnik
und Informationstechnik
Professur Schaltkreis- und
Systementwurf
Prof. Dr.-Ing. Heinkel

Introduction to DLX processor

Manual

Practical course for the lecture "Components and Architectures"
Date: May 23, 2014

1 Introduction

1.1 Aim of the practical lesson

Within the first practical lesson the students should get familiar with the DLX processor and its simulator dlxview. The DLX processor executes Assembly programs, a very low-level programming language. This lesson is the base for the other practice lessons on pipelining and code optimization.
Without passing this practice it is not possible to continue in this course!

1.2 Literature

- Exercise slides chapter 1 and 2
- DLX instruction list (available in OPAL)
- John Hennessy, David Patterson: Computer Architecture. *A Quantitative Approach.* Morgan Kaufmann Publishers (available in the library)

2 Theoretical background

2.1 DLX processor

The DLX processor is a theoretical processor designed by Hennessy and Patterson for education purposes at Berkeley University. It is closely related to the commercially used MIPS architecture. It is a 32 bit RISC architecture with a very limited instruction set. The register file contains 32 general purpose registers (r0 to r31), 32 single-precision floating point registers (f0 to f31) and some special purpose registers. The single precision FP-registers can be combined to 16 double precision registers (f0, f2, f4, f6, ...). GP-register r0 is always zero.
Instruction execution is separated in 5 stages:

- IF - Instruction Fetch
- ID - Instruction Decode
- EX - EXecute
- MEM - MEMory access
- WB - Write Back result

These stages will be discussed in detail in the pipelining practice (lesson 2) but are already visible in the simulator.

2.2 Assembly language

The main task of the whole practice course is to understand the principles of pipelining. So we need a detailed idea what is happening inside the control unit of a processor. CPUs can only deal with machine code - a series of 0 and 1 representing the operations and operands. Every program (like C/C++, Pascal, ...) running on a processor need to be compiled to that machine level. This machine code is barely readable for humans, so the Assembly language was created. It represents the machine code instruction by instruction. So the language highly depends on the processor for which it was intended.
An Assembly code for x86 will not work on a DLX or on an ARM core.
For editing Assembly files you can use any native editor that does not add built-in information to the files. So WYSIWYG editors like Word or OpenOffice are not usable. For this practice it is suggested to use *gedit* or *xemacs*.

3 Software Environment

At first you should have received the login information from your advisor. This information is valid for all SSE courses (EDA-Tools, Components & Architectures, Design of heterogeneous systems, Rapid Prototyping, Software Environments for Smartphone Applications) so keep it with you at every practice session.
You should be familiar with the most common Linux commands. These are:

- cd [name]- change directory
- mv [name] [target] - move a file or directory to the target
- cp [name] [target]- copy a file or directory

The default directory after login from a labpool-PC is

```
/home/praktikum/prXX/.SLinux6
```

If you do a remote login on hokus.etit.tu-chemnitz.de the shell will point to

```
/home/praktikum/prXX/.SLinux5
```

so you have to change directory to get your data.

4 Preparation of practice lesson

The following questions shall be solved before coming to the lab! Students that cannot present their preparation tasks will be refused from continuing the lesson.

- Read this document from the beginning to the end carefully!
- Review the exercise slides from chapter 1 and 2!
- Please answer the following questions on the solution sheet (last page).
  1. What is the comment sign in DLX Assembly language?
  2. What is the value of register r0?
  3. What is the mathematic formula for the instruction *addi r1, r2, 3*?
  4. What is the size of a variable with datatype "double"?
  5. What is the difference of the simulator behavior between "next cycle" and "step forward"?

5 Tasks for Practice lesson

Please answer the questions in the sections "task to solve" on the solution sheet (from your preparation) during the practice. Please submit your solution sheet to your advisor at the end of the practice session.

5.1 Setting up your environment

At first please start the login process. If your username is not in the list select "Andere" (others) and type your login name. *In the next screen (where you are asked for your password) you can select your language at the bottom of the screen.* After entering your password you should see a desktop. Open a terminal (black rectangle in the menu bar on top).

Please copy some example files to your directory:

```
cp -r /home/4all/packages/dlxview0.9/examples  ./dlx
```

Then go into this directory

```
cd dlx
```

and have a look in the file *integer_test.s*

```
gedit integer_test.s &
```

Keep the editor open. It is not possible to start the editor gedit from within the simulator! The simulator is called by typing (without &!)

```
dlxview
```

The dlxview simulator has a straightforward setup. At first you need to configure the processor. You can choose between three pipeline types. Which you should select depends on the practice task to solve.

5.2 Introduction to integer unit

For this practice please go to configure, choose *Basic Pipeline* and leave all parameters as they are. Then load the file *integer_test.s* to the simulator. You have now three options to simulate:

- Step Forward (to next instruction fetch)
- Next Cycle (one clock cycle)
- Run (forever)

Select Step Forward. Chose *Default* as entry point.
The code is a quite simple introduction example:

Listing 1: "integer_test.s"

```
1 .text
2         addi r3, r0, 8
3         addi r4, r0, 1
4 loop: add r4, r4, r4
5         subi r3, r3, 1
6         bnez r3, loop
7         nop
8         trap #0
```

Let's walk through the lines:

- Line 1 states that the following code should be in program memory
- Line 2 sets register3 to 8 (remember r0 is always zero but there is no set instruction for numbers in registers) - this is our loop counting variable
- Line 3 sets register4 to 1
- Line 4 is the start of the loop body (label "loop") and adds r4 to itself
- Line 5 decrements the loop variable r3
- Line 6 is a branch where the loop is repeated if r3 is not zero
- Line 7 is a No Operation instruction in the "delayed slot" - we will deal with this in the next practice
- Line 8 is a call to the operating system and terminates the program

So in C language this is just a for-loop

```
for (r3 = 8; r3 > 0; r3--)
       r4 = r4 + r4;
```

In Assembly language it is in most cases better to use a decrementing loop variable as comparisons to zero are much easier to realize in hardware than comparisons to other values.

5.2.1 Task to solve

Simulate the program *integer_test.s* using dlxview. Check the phases of instruction execution in the integer datapath window.
Register content can be viewed using the terminal. For general purpose registers type:

```
get XX d
```

where X shall be replaced by either the register (like r1) or the memory address/variable (like 0x8) and d is the argument for decimal return value.

- How many cycles does the program need to finish when configured as default pipeline?
- What are the final decimal values of the registers r3 and r4?

5.3 Introduction to Floating point unit (FPU)

This is also a simple program to get familiar with the FPU consisiting of two files.
The .d file provides initial memory data. The .s file is pure Assembly code. The third format for dlx simulation is .i to initialize registers. The .s file is always loaded as last file.
At first switch to the editor window and open the new files float_test.s and float_test.d there.
Next please:

- Reset the simulator with "Same Config, New Program".
- Select the initial memory data file "float_test.d" and press Load
- Select the program file "float_test.s" and press Load
- press *Done*

Listing 2: "float_test.d"

```
        .data           0
        .global  a
a:      .double  3.14159265358979
        .global  x
x:      .double  1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
        .double  17,18,19,20,21,22,23,24,25,26,27
        .global  xtop
xtop:   .double  28
```

Listing 3: "float_test.s"

```
        ld      f2,a
        add     r1,r0,xtop
loop:   ld      f0,0(r1)
        multd   f4,f0,f2
        sd      0(r1),f4
        sub     r1,r1,#8
        bnez    r1,loop
        nop
        trap    #0      ; terminate simulation
```

5.3.1 Task to solve

Simulate the program *float_test.s* using dlxview. Check the phases of instruction execution in the general datapath window.

- How many cycles does the program need to finish when configured as default pipeline?
- How many loop runs are executed?
- What is the value of r1 after executing the line $add\ r1, r0, xtop$?
- What is the value of xtop after the first loop execution? For checking floating point registers and memory contents use

  ```
  fget XX [d]
  ```

  where XX is either one of the FP registers f0 to f31 or a memory address/variable (e.g. fget xtop d). The option d is used for double precision data.
- What is the mathematic formula of the program? This should be given in the form
  $OutputToMemory = WhatHappensToInputFromMemory$

Solution sheet for: Introduction to DLX processor

Name:

Matrikel number:

Number of lab group:

Preparation Tasks:

Assembly comment sign:

Value for register r0:

Mathematic formula:

Size of a double:

Difference between NextCycle and StepForward:

Practice Tasks:

integer_test.s:

Total number of cycles for simulation:

Final decimal value of registers r3 and r4:

float_test.s:

Total number of cycles for simulation:

Total number of loops executed:

Value of r1 after second instruction:

Value of xtop after first loop execution:

Mathematic formula of program:
