The **Microelectronics Training Center**

The MTC is an initiative within the INVOMEC division

Industrialization & Training in Microelectronics

Lab-exercise

Lab 4:

Transfer assembly code into ram contents

Cluster: Cluster1
Module: Module5c

Target group: Students

Version: 1.1
Date: 9/12/07
Author: Geert Vanwijnsberghe
Modified by:

This material was developed with support of the European Social Fund.
ESF: Prevent and combat unemployment by promoting employability, entrepreneurship, adaptability and equal opportunities between women and men, and by investment in people.
http://www.esf-agentschap.be

Introduction

In the previous module the system (hardware) has been built while in this system the ram contents will be created (Software).

Objectives

After completing this module, you should be able to transfer your own assembly code into machine instructions.

Knowledge background

- Basic knowledge of ModelSim
- Basic knowledge of assembly code

Classification

- Level: 2
- Duration: 1 hour

Input

- VHDL files
- Assembly program written in VHDL
- micro6_v2.pdf : thesis : Design and implementation of a 32-bit RISC microprocessor

The lab

Assembly language

Assembly language or simply assembly is a human-readable notation for the machine language that a specific computer architecture uses. Machine language, a pattern of bits encoding machine operations, is made readable by replacing the raw values with symbols called mnemonics. Mnemonics replace opcodes as well as references to operands, for example, register names and immediate data. An assembly language statement may convey additional information too, for example, addressing modes and cross references to other parts of the program.
Since our microprocessor (called Micro6 hereafter) supports its own instruction set and instruction formats, an assembler was required. The VAS assembler (VHDL assember) was developed for this purpose.
In addition to codes of the machine instructions, Micro6 assembly language provides extra directives for assigning address locations for instructions or code. For simplicity of programming, the layout of the program in memory is transparent to the programmer. However, instructions can be referenced symbolically by *labels*.
Micro6 assembly has a simple symbolic capability for defining immediate data as *constants*. Micro6 does not support immediate addressing mode but this mode is substituted by page-0 addressing.
Like most computer languages, comments can be added to the source code.

VAS assembler

VAS stands for VHDL Assembler because it was written in VHDL, exploiting the programming capabilities of the hardware description language. Moreover, the output of VAS is a VHDL package containing the machine code to be stored in the memory. VAS is not an executable program. It must be invoked, or rather loaded and run, by an HDL simulator.
VAS is a 2-pass cross assembler:
2-pass assembler: VAS goes through the source code (in assembly language) twice. The internal data base is built in the first pass, which is used later in the second pass.
Cross assembler: VAS produces machine code for the Micro6 microprocessor while it runs on a different computer system.

ⓘ It is surely not the standard way to write an assembler program in VHDL. Normal programming languages are more suited for this.

VAS components

VAS is composed of 2 VHDL units as follows:
1. VAS entity and architecture: this is the unit that is loaded by the simulator. (assembler.vhd)
2. Assembler package: a package containing the functions and procedures used by the assembler. (assembler_pk.vhd)

VAS input files

1. Assembly language program (Program segment) (prog.asm)
2. Memory initial data (Data segment): this file is optional (data.asm)
3. Templates of the output files. (micro_ram_pk_top.vhd and micro_ram_pk_bottom.vhd and ram_top.coe)

VAS output files

1. micro_ram_pk.vhd : VHDL package containing the initial memory contents. The memory content is declared as a deferred constant (values in the body of the package) because the use of deferred constants makes recompiling the other design units unnecessary. However, compiling the output file (micro_ram_pk.vhd) is still necessary.
2. ram.coe : Memory Coefficients file (COE). This file is used in module 6 by Xilinx CoreGen to generate the necessary Block RAM modules and initialize their contents. These Block RAM modules are needed to synthesize and implement the memory on the virtexII-pro FPGA.

Exercise: sort

Assume that a list of integer data items is stored in the ram. The first item in the list is the number of items. The sort program has to sort the items and store the again in the ram.

The sort program works by selecting the smallest/largest unsorted item remaining in the list, and then swapping it with the item in the next position to be filled. In other words, the program traverses a list with *n*-elements *n-1* times. Each pass is 1 element shorter than the previous one. Sub lists are sorted by swapping the top element with the smallest/largest one. The assembly code for this algorithm is stored in prog.asm.

You have to complete this code. You find more info on the instruction mnemonics in chapter 3 of the "micro6_v2.pdf" file.

```
.LISTADDR #2999;   -- Address containing the length of the list

LDM LISTADDR R11;
LD  R11 R10;
```



```
INC R11;
DEC R10;

$LOOP:
CPR R11 R13; -- PREVIOUS INDEX OF MIN ELEMENT

PSH R11; -- STARTING ADDRESS
PSH R10; -- LENGTH
JSR MIN;
POP R14; -- INDEX OF MIN ELEMENT
LD  R13 R15;
LD  R14 R16;
CMP R15 R16;
BNQ SWAP1;

$NEXT:
INC R11;
DEC R10;
BEQ END;
BRA LOOP;

$SWAP1:
PSH R13;
PSH R14;
JSR SWAP;
BRA NEXT;

$END:
END;

-----------------------------
-- SUBROUTINE TO SWAP TWO LIST ELEMENTS
-- PARAMETERS:
-- 1: INDEX OF FIRST ELEMENT
-- 2: INDEX OF SECOND ELEMENT
$SWAP:
POP R1; -- INDEX OF SECOND ELEMENT
POP R0; -- INDEX OF FIRST ELEMENT
-- add your code here  <---------------------------------------
-- do the swap of R1 and R0
-- you may use R2 and R3

RTN;
-----------------------------------
-- SUBROUTINE TO FIND THE MIN OF A LIST
-- PARAMETERS:
-- 1: STARTING ADDRESS
-- 2: LENGTH
-- RETURNS:
-- 1: INDEX OF MIN ELEMENT

$MIN:
POP R1; -- LENGTH
POP R0; -- STARTING ADDRESS
```

```
CPR R0 R4;
ZRO R29;
INC R29;

$START:
LD  R4 R2;
LDX R0 I1 R3;
CMP R2 R3;
BGT S1; -- NEXT ELEMENT
ADD R0 R29 R4;
CPR R3 R2;

$S1:
INC R29;
DEC R1;
BEQ FIN; -- FINISH
BRA START;

$FIN:
PSH R4;
RTN;
```

The data to be sorted is stored in the file data.asm.

The above program has to be converted in the memory contents package micro_ram_pk.vhd using the VHDL assembler.
Start modelsim in the directory containing your files and use the commands below.

Modelsim> vlib work
Modelsim> vcom micro_pk.vhd
Modelsim> vcom micro_control_pk.vhd
Modelsim> vcom assembler_pk.vhd
Modelsim> vcom assembler.vhd

These commands create an assembler as a VHDL entity and architecture.
The simulation of this entity/architecture converts the prog.asm and data.asm files into micro_ram_pk.vhd.

Modelsim> vsim -Gusetestdata=true work.assembler
Modelsim> run –all

ⓘ If you modify prog.asm or data.asm you only need to rerun the simulation. You do not have to recompile the vhdl files.

You can verify the syntax correctness of micro_ram_pk.vhd by compiling it.

Modelsim> vcom -quiet micro_ram_pk.vhd

The memory organization you see in the micro_ram_pk.vhd file is shown in figure 2.9 of the micro6_v2.pdf.

In the next lab we will combine the software and the hardware and verify the complete system.