Computer Architecture LU
Lab Notes/Assignments

Wolfgang Puffitsch
wpuffits@mail.tuwien.ac.at

September 28, 2010

Contents

1 General

The goal of the Computer Architecture lab course is to design a microprocessor. This includes the design and simulation of an instruction set, the implementation of a processor and its evaluation. The only requirement for the processor is that is must be pipelined. Everything else is up to you. Feel free to be creative!

You may retrieve information from any source, as long as proper acknowledgement is given. You are especially encouraged to discuss ideas with your colleagues. There is also a wiki for this course; contributions to that wiki are welcome.

Computer Organization and Design: The Hardware/Software Interface by David A. Patterson and John L. Henessy and *Computer Architecture: A Quantitative Approach* by John L. Hennessy and David A. Patterson are recommended for this course. Keep in mind that especially the latter book also covers advanced techniques whose implementation is beyond the scope of this course. A book that might be interesting for those who want to learn more about computer architecture for embedded systems is *Embedded Computing: A VLIW Approach to Architecture, Compilers, and Tools* by Joseph A. Fisher, Paolo Faraboschi, and Cliff Young. However, you should have understood the basic concepts of computer architecture before turning to that book.

2 Instruction Set Comparison

Goal Overview of different instruction set architectures (ISAs).

Points 10

Deliverables

- Description and comparison of the ISAs

2.1 Description

Compare different instruction set architectures. Describe one ISA for each group member, and especially answer the following questions:

- Does the ISA describe an accumulator, a register, or a stack machine?
- Does it describe a CISC or RISC architecture?
- Are latencies handled by the hardware or are they visible at the ISA level? For example, branch delay slots expose the latencies of branches at the ISA level.
- Are conditional branches (compute condition and jump) performed in a single step, or are testing and branching unbundled?
- Where are processors that implement the ISA deployed? In embedded systems (micro-controllers, communication, multimedia), in servers, in desktop computers?
- What are the goals of the architecture? Performance, die area, energy efficiency, code size, ...?
- How are these goals reflected in the ISA?
- Which features of the ISA do you like, which features would you change?

2.2 Code

Translate the following C-code to the assembly language described by the ISA (you can use a compiler, as long as you understand the result):

Listing 1: Compute Sum

```c
int sum(int len, int arr[]) {
    int i;
    int sum = 0;
    for (i = 0; i < len; i++) {
        sum += arr[i];
    }
    return sum;
}
```

How many instructions are executed in each loop iteration? Also try to estimate the number of cycles for a loop iteration and the code size of the loop.

2.3 Comparison

Explain the differences between the ISAs described by your group.

3 Instruction Set Design

Goal Understanding the trade-offs for the high-level design of an ISA.

Points 10

Deliverables

- Description of the ISA
- Documentation of design decisions
- Programs in assembler and C

3.1 Description

Design an ISA and the respective encoding. Answer the questions from Section 2.1 and 2.2 for your ISA.[1]

Write simple programs (at least one program per group member), to detect flaws and inconsistencies. Please also submit C code for these programs.

[1]Use your imagination where facts are missing.

4 Assembler and Simulator

Goal First evaluation of the ISA.

Points 10

Deliverables

- Demonstration of the assembler and the simulator
- Assembly code for the simulated programs
- Source code for assembler and simulator
- Changes to the ISA if applicable

4.1 Description

Write a simple assembler and high-level simulator for your ISA. Simulate various simple programs with these tools. Deficiencies of the original ISA can and should be fixed.

4.2 Assembler

Keep the assembler as simple as possible. It is not necessary to support macros or linking several files. You can use the C preprocessor for macros.

You can find example assemblers in the course resources. Of course, these examples can be reused for your own assembler.

4.3 Simulator

Write a high-level simulator for the ISA. You can find example simulators in the course resources. Keep the simulator simple and concentrate on the core functionality.

4.4 Simulation

Simulate the programs from Section 3 with these tools. You will also be assigned C programs from other groups to enlarge the scope of the evaluation.

5 Block Diagram

Goal Basic design of the processor

Points 10

Deliverables

- Block diagram with description
- Documentation of design decisions
- Changes to the ISA if applicable

5.1 Description

Design a block diagram for a processor that implements your ISA. The minimum requirement is that the processor is pipelined. Take care to make the different stages of the pipeline clearly visible. Try to trace the execution of several instructions in the pipeline. How does the pipeline behave for consecutive instructions with dependencies (e.g., `r1 = r2 + r3; r4 = r1 & r5`)? What happens when the pipeline executes a jump?

Decide whether the processor implements a Harvard or Von Neumann architecture, and whether you use internal or external memory for the memory areas. Also consider how memory and I/O accesses are treated in the pipeline.

On-chip memories in modern FPGAs have registered inputs, which should be taken into account in the block diagram. A register file that is implemented with on-chip memory and accessed in the decode stage of the pipeline is shown in Figure 1.

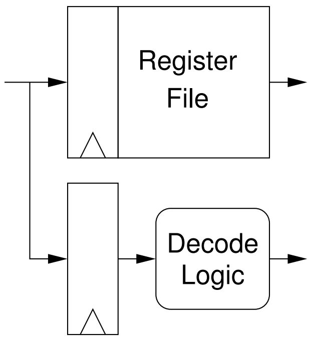

Figure 1: Decode stage with parallel register access

6 Simulation I

Goal Unit test of hardware modules

Points 5

Deliverables

- Demonstration of the simulation
- Implementation documentation

6.1 Description

Implement the individual pipeline stages of the design from Section 5 in VHDL or Verilog. Simulate the individual pipeline stages in Modelsim or some other hardware simulation environment. Write test benches to catch implementation errors as early as possible in the design flow.

The register file and the ALU are relatively simple units and therefore lend themselves as starting points of the implementation. You can leave optimizations to later design steps. For example, write `result <= a*b` instead of using a more efficient (in terms of area and maximum frequency) multi-cycle multiplication routine.

7 Simulation II

Goal Resolving data dependencies

Points 5

Deliverables

- Demonstration of the simulation
- Documentation how data dependencies are resolved
- Change log of the implementation

7.1 Description

Extend the design from Section 6 such that sequences of instructions can be executed. Simulate at least the execution of the following instruction sequence (of course, appropriately modified for your ISA):

Listing 2: Code with data dependencies

```
r1 = 7;
r2 = 5;
r3 = 4;
r4 = r2 + r3;
r5 = r4 & r1;
```

8 Simulation III

Goal Resolving control hazards

Points 10

Deliverables

- Demonstration of the simulation
- Documentation how control hazards are resolved
- Change log of the implementation

8.1 Description

Extend the design from Section 7 such that jumps (and function calls) are handled correctly. Simulate the following program[2]:

Listing 3: Code with control dependencies

```
int fib(int a) {
   if (a < 2) {
       return 1;
   } else {
       return fib(a-1)+fib(a-2);
   }
}

int main() {
   return fib(9);
}
```

[2]The function fib() is off-by-one compared to the canonical definition of the Fibonacci sequence.

9 Presentation of the Hardware

Goal Evaluation of the processor design

Points 20

Deliverables

- Demonstration of the processor running on an FPGA
- Documentation of the final ISA
- Documentation of the implementation
- Results of the evaluation
- Source code of the processor and the related tools

9.1 Description

Complete your hardware design and let it run on an FPGA. The provided development board comprises a Cycore board and a dspio board. The Cycore board contains an Altera Cyclone FPGA (EP1C12Q240) and 1 MB SRAM. The dspio board provides I/O connections, most importantly an RS232 interface.

Include at least a UART module in your processor to enable reasonable debugging, and a cycle counter for the performance evaluation. You find appropriate hardware modules in the course resources. There, you also find the pin definitions for the dspio board. Most I/O modules are interfaced through the SimpCon bus, a simple bus protocol for system-on-a-chip components.

Measure the performance of your processor with the programs from Sections 4, 7, and 8. In the evaluation results please provide the execution time of the program fragments in cycles. Furthermore, provide the following figures for your processor: maximum clock frequency f_{max}, real clock frequency f_{real}, occupied logic cells r_{LC}, logic cells that are used as registers r_{reg}, and memory bits r_{mem}. Please also indicate the factors α, β, γ, and δ, as described in Section A.1. These factors depend on whether you optimized your design for execution time, code size, resource consumption, or power consumption.

A Grading

The maximum number of points is 100. For the assignments in Sections 2 to 9, at most 80 points can be attained. The missing 20 points result on the one hand from the procedure described in Section A.1, on the other hand from special features of the processor. The grading scheme is shown in Table 1. For a positive grade, it is necessary to achieve at least 50 points and to present the final hardware as described in Section 9.

Feature Points

- Interrupts: 5 points
- Program download: 5 points
- External memory: 5 points
- Special feature X: 5-20 points

Points	Grade
100 - 90	S1
89 - 80	U2
79 - 65	B3
64 - 50	G4
49 - 0	N5

Table 1: Grading scheme

A.1 Relative Evaluation

Four parameters are considered for the evaluation of the design: execution time T, code size S, cost C, and power consumption P. The formulas 1 to 4 show how these parameters are computed. B is the set of benchmarks to be evaluated.

$$T = \frac{1}{f_{real}} \prod_{b \in B} cycles(b)^{\frac{1}{|B|}} \qquad \text{geometric mean, in } \mu\text{s} \tag{1}$$

$$S = \prod_{b \in B} size(b)^{\frac{1}{|B|}} \qquad \text{geometric mean, in bit} \tag{2}$$

$$C = r_{LC} + \frac{r_{mem}}{16} \tag{3}$$

$$P = 10 + f_{real} \left(\frac{{r_{regs}}^{0.79}}{600} + \frac{r_{LC}}{7000} + \frac{r_{mem}}{50000} \right) \tag{4}$$

The benchmarks for the evaluation of the performance and the code size of the processor can be found in the course resources. The rules for benchmarking are detailed in Section A.2.

The cost estimation is linear in terms of logic cell and memory consumption. When designing ASICs, it is usually assumed that costs increase more that linear with the die area. The formula however reflects the fact that the price within an FPGA family is roughly linear to the number of logic cells.

The formula for the power consumption is a very rough estimate. It takes into account the static power, the consumption for the clock network and the dynamic power for logic cells and memories. The computed value is only intended for the relative evaluation and should not be confused with the real power consumption of the processor.

Please provide factors for weighting execution time, code size, cost and power consumption such that the following formula is maximized:

$$V = \alpha \frac{T_{avg}}{T} + \beta \frac{S_{avg}}{S} + \gamma \frac{C_{avg}}{C} + \delta \frac{P_{avg}}{P} \tag{5}$$

The following constraints must be fulfilled: $\alpha + \beta + \gamma + \delta = 1$ and $0.1 \leq \alpha, \beta, \gamma, \delta$. T_{avg}, S_{avg}, C_{avg}, and P_{avg} are mean values that are computed from the designs of the other groups and a reference design. The design with the highest value V receives 20 points; other designs receive a proportionate number of points.

A.2 Benchmarking Rules

A.2.1 Optimizations

Optimizations are allowed for the C programs, but the code between BENCHMARK START and BENCHMARK END must not be optimized away (i.e., you must not replace that code with the final result).

You may apply any optimization a reasonable compiler might apply, such as common subexpression elimination, software pipelining, if-conversion, etc. You must not change the general algorithm, such as replacing a recursive algorithm with an iterative one; elimination of tail recursions is allowed though.

As a special case, *any* optimizations are allowed for functions marked with FREESTYLE OPTIMIZATIONS, as long as they still work with general input. An example would be replacing an interpreter with just-in-time compilation.

Keep in mind that int is defined as "natural size suggested by the architecture", but must be at least 16 bits wide. Typically, int has the same size as a general purpose register. A char must be large enough to store members of the basic character set and it must also be individually addressable. For example, if your architecture can only address 16-bit values, a char would be 16 bits wide. Types such as int32_t must be implemented using the obvious bit width.

A.2.2 Timing

The timing result is the number of cycles your processor consumes between the the BENCHMARK START and the BENCHMARK END marks. Place your timing instrumentation there. Cycles spent for data initializations may be excluded from the timing measurement.

A.2.3 Size

The size result is the number of *bits* the whole .text-segment of the program consumes, excluding instrumentation code, but including code for data initialization.

A.2.4 Data Initialization

You are free to choose how constant data is initialized. During execution, the constant data must reside in data memory; the benchmark functions must also work with data that is constructed at run-time. Constant data may reside in read-only data memory. Please note that the size result includes data initialization, but the timing result does not.

A.2.5 Refereeing

The course organizers will decide on the legality of optimizations in cases that are not fully covered by the above rules. The basic principle for decisions will be that a fair comparison with other groups must be possible (also taking into account the impact on timing, cost, and code size). You may support your case by presenting real-world examples of similar optimizations.

B Links

Homepage
http://ti.tuwien.ac.at/rts/teaching/courses/calu

Resources
http://ti.tuwien.ac.at/rts/teaching/courses/calu/files

Wiki
http://www.soc.tuwien.ac.at/Computer_Architecture_Lab

David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface, 4th ed.* Morgan Kaufmann Publishers, 2009.
http://www.elsevierdirect.com/product.jsp?isbn=9780123744937

John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach, 4th ed.* Morgan Kaufmann Publishers, 2006.
http://www.elsevierdirect.com/product.jsp?isbn=9780123704900

Joseph A. Fisher, Paolo Faraboschi, and Cliff Young. *Embedded Computing: A VLIW Approach to Architecture, Compilers and Tools* Morgan Kaufmann Publishers, 2005.
http://www.elsevierdirect.com/product.jsp?isbn=1558607668

Peter Ashenden. *The VHDL Cookbook.* University of Adelaide, 1990.
http://www.google.com/search?q=VHDL+cookbook+Ashenden&btnG=Google+Search

Quartus Web Edition and Modelsim Altera Edition
http://www.altera.com/support/software/sof-quartus.html

SPEAR2
https://trac.ecs.tuwien.ac.at/Spear2

Java Optimized Processor (JOP)
http://www.jopdesign.com

McAdam's RISC Computer Architecture (marca)
http://www.soc.tuwien.ac.at/files/marca