ESE 345 Computer Architecture Fall 2018 Prof. Mikhail Dorojevets

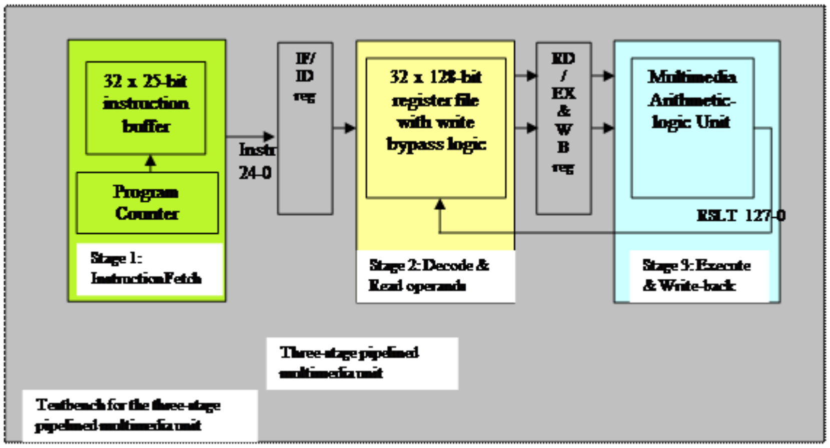
ESE 345 2018 Project:
Pipelined multimedia unit design with the VHDL/Verilog hardware description language

Project Description

Purpose: To learn a use of VHDL/Verilog hardware description language and modern CAD tools for the structural and behavioral design of the triple-stage pipelined multimedia unit with a reduced set of multimedia instructions similar to those in the Sony Cell SPU and Intel SSE architectures.

CAD tools: Mentor Graphics Modelsim at the Undergraduate CAD Lab (room 281 Light Eng. Bldg.) or any other VHDL/Verilog simulator (e.g. Aldec).

It is a **one/two**-student project.



Procedure:

I. It is suggested to read **Chapter 3.6** on subword parallelism and, if necessary, the Intel MMX and Sony Cell SPU[1] papers below and understand the original concept of multimedia processing introduced as MMX architecture for Intel processors in the 1990s.
II. **Refresh your knowledge** of VHDL/Verilog in the HDL design of digital circuits by reading **Chapter 4.13**. �
III. **Develop** a detailed block diagram and the HDL model of the three-stage multimedia unit and its modules.
IV. **Verify** individual modules of your design with their testbenches before instantiating them in higher order modules. Verify the final model with a testbench module and generate file **Results** showing the status of each stage of the unit during execution.

Requirements:

The complete 3-stage pipelined design is to be developed in a structural way with several modules operating simultaneously. Each module represents a pipelined stage with its interstage register. The major units inside those stages modules are described below.

1. **Multimedia ALU**
   The ALU must be implemented as **behavioral model in VHDL or continuous assignment (dataflow) models in Verilog.**

2. **Register file**
   The register file has 32 128-bit registers. On any cycle, there can be 3 reads and 1 write. When executing instructions, each cycle two/three 128-bit register values are read, and one 128-bit result can be written if a write signal is valid. This ***register write*** signal must be explicitly declared so it can be checked during simulation and demonstration of your design. A technique of **data forwarding** is to be used so that a write and read to the same register will return the new value for the read.
   The register module must be implemented as **a behavioral model in VHDL (a (dataflow/RTL model in Verilog).**

3. **Instruction buffer**

The instruction buffer can store 32 25-bit instructions. The contents of the buffer should be loaded by the testbench instructions from a test file at the start of simulations. �Each cycle one instruction specified by the Program Counter (PC) is fetched, and the value of PC is incremented by 1.

The instruction buffer module must be implemented as **a behavioral model in VHDL (a (dataflow/RTL model in Verilog).**

4. **Three-stage pipelined multimedia unit**

Clock edge-sensitive pipeline registers separate the IF, ID, and EXE stages.

The EXE stage of the pipeline is responsible for calculating the result and writing it to the register file.

All instructions (including **li**) take three cycles to complete. All instructions (including **li**) take three cycles to complete. This pipeline must be implemented as structural model with three modules representing corresponding pipeline stages and their interstage registers. Three instructions can be at different stages of the pipeline at every cycle

5. **Testbench**

This module supplies an instruction code to be loaded from a file to the instruction buffer and, when it is finished, **checks the contents of the register file**.

It must be implemented as a **behavioral model.**

It is up to each team to choose how the assembly test code for the unit is converted to binary and saved in a file from which is to be loaded into the instruction buffer at the start of simulation.

6. **Results**

This file must show status of the pipeline with the opcodes, input operands, and results of execution of instructions in the pipeline for each cycle.

7. **Instruction formats and opcode description**

7.1 Load immediate instruction format

24	23 21	20 5	4 0
0	li	16-bit immediate	rd

li: Load a 16-bit immediate value from the [20:5] instruction field into the 16-bit field specified by the li field [23-21] of the 128-bit register rd.

7.2 Multiply-add and multiply-subtract (low/high) R4-instruction format

24	23	22 20	19 15	14 10	9 5	4 0
1	0	MA/MS	rs3	rs2	rs1	rd

In the table below, 32-bit signed integer add and subtract operations are performed with **'saturate to signed word'** rounding that takes a 32-bit signed integer result, and converts it to -2^{31} if it's less than -2^{31}, to $+2^{31}-1$ if it's greater than $+2^{31}-1$, and leaves it unchanged otherwise. Multiply operations on 16-bit signed operands calculate 32-bit integer signed products.

MA/MS/l/h 22-20	**Description of instruction code [22-20]**
x00	**Signed integer multiple-add low with saturation:** Multiply low 16-bit-fields of each 32-bit field of registers **rs3** and **rs2,** then add 32-bit products to 32-bit fields of register **rs1,** and save result in register **rd**.
x01	**Signed integer multiple-add high with saturation:** Multiply high 16-bit-fields of each 32-bit field of registers **rs3** and **rs2,** then add 32-bit products to 32-bit fields of register **rs1,** and save result in register **rd**.
x10	**Signed integer multiple-subtract low with saturation:** Multiply low 16-bit-fields of each 32-bit field of registers **rs3** and **rs2,** then subtract 32-bit products from 32-bit fields of register **rs1,** and save result in register **rd**.
x11	**Signed integer multiple-subtract high with saturation:** Multiply high 16-bit-fields of each 32-bit field of registers **rs3** and **rs2,** then subtract 32-bit products from 32-bit fields of register **rs1,** and save result in register **rd**.

7.3 R3-instruction format

24	23	22 — 15	14 — 10	9 — 5	4 — 0
1	1	opcode	rs2	rs1	rd

In the table below, 16-bit signed integer add (**ahs**) and subtract (**sfhs**) operations are performed with **'saturate to signed word'** rounding that takes a 16-bit signed integer X, and converts it to −32768 if it's less than −32768, to +32767 if it's greater than 32767, and leaves it unchanged otherwise.�

Opcode 22-15	Description of Instruction Opcode
xxxx0000	**Nop**
xxxx0001	**bcw:** broadcast a right 32-bit word of register **rs1** to each of the four 32-bit words of register **rd.**
xxxx0010	**and**: bitwise ***logical and*** of the contents of registers ***rs1*** and ***rs2***
xxxx0011	**or**: bitwise ***logical or*** of the contents of registers ***rs1*** and ***rs2***
xxxx0100	**popcnth**: *count ones in halfwords*: the number of 1s in each of the four halfword-slots in register ***rs1*** is computed. If the halfword slot in register ***rs1*** is zero, the result is 0. Each of the results is placed into corresponding 16-bit slot in register ***rd***. ***(Comments: 8 separate 16-bit halfword values in each 128-bit register)***
xxxx0101	**clz**: *count leading zeroes in words*: for each of the two 32-bit word slots in register ***rs1*** the number of zero bits to the left of the first non-zero bit is computed. If the word slot in register ***rs1*** is zero, the result is 32. The two results are placed into the corresponding 32-bit word slots in register ***rd***. ***(Comments: 4 separate 32-bit values in each 128-bit register)***
xxxx0110	**rot:** *rotate right:* the contents of register ***rs1*** are rotated to the right according to the count in the 7 least significant bits (6 to 0) of the contents of register ***rs2***. The result is placed in register ***rd***. If the count is zero, the contents of register ***rs1*** are copied unchanged into register ***rd***. Bits rotated out of the right end of the 128-bit contents of register ***rs1*** are rotated in at the left end.
xxxx0111	**shlhi**: *shift left halfword immediate*: packed 16-bit halfword shift left logical of the contents of register ***rs1*** by the 4-bit immediate value of instruction field ***rs2***. Each of the results is placed into the corresponding 16-bit slot in register ***rd***. ***(Comments: 8 separate 16-bit values in each 128-bit register)***
xxxx1000	**a**: *add word*: <u>packed</u> 32-bit unsigned add of the contents of registers ***rs1*** and ***rs2*** ***(Comments: 4 separate 32-bit values in each 128-bit register)***
xxxx1001	**sfw**: *subtract from word*: (packed) 32-bit unsigned subtract of the contents of registers ***rs1*** and ***rs2*** ***(Comments: 4 separate 32-bit values in each 128-bit register)***
xxxx1010	**ah**: *add halfword* : (packed) (16-bit) halfword unsigned add of the contents of registers ***rs1*** and ***rs2*** ***(Comments: 8 separate 16-bit values in each128-bit register)***
xxxx1011	**sfh**: *subtract from halfword*: (packed) (16-bit) halfword unsigned subtract of the contents of registers ***rs1*** and ***rs2.*** ***(Comments: 8 separate 16-bit values in each128-bit register)***
xxxx1100	**ahs**: *add halfword saturated*: (packed) (16-bit) halfword signed add **with saturation** of the contents of registers ***rs1*** and ***rs2.*** ***(Comments: 8 separate 16-bit values in each128-bit register)***
xxxx1101	**sfhs**: *subtract from halfword saturated*: (packed) (16-bit) signed subtract **with saturation** of the contents of registers ***rs1*** and ***rs2.*** ***(Comments: 8 separate 16-bit values in each128-bit register)***
xxxx1110	**mpyu**: *multiply unsigned*: the 16 rightmost bits of each of the four 32-bit slots in registers ***rs1*** are multiplied by the 16 rightmost bits of the corresponding 32-bit slots in register ***rs2***, treating both operands as unsigned. The four 32-bit products are placed into the corresponding slots of register ***rd***. ***(Comments: 4 separate 32-bit values in each 128-bit register)***
xxxx1111	**absdb**: *absolute difference of bytes*: the contents of each of the 16 byte slots in register ***rs2*** is subtracted from the contents of the corresponding byte slot in register ***rs1***. The absolute value of each of the results is placed into the corresponding byte slot in register ***rd***. ***(Comments: 16 separate 8-bit values in each 128-bit register)***

Expected Results

A full project report including the goals, multimedia unit block diagram, design procedure, all testbenches, conclusions, **<u>the VHDL/Verilog source code</u>** of the multimedia unit, and simulations results **<u>in printed form</u>** must be presented **<u>at the start</u>** of your 20-minute demonstration during a time slot assigned to your team by TA.

The **electronic version of the report** must be also sent to the TA & Instructor **before the start of the presentation**.

Project presentation will not start without these printed and electronic documents submitted.

Project Demonstration & Submission Period: last week of classes in December.

Here are some links for the Intel MMX technology description that will give you an idea of multimedia processing:

MMX(tm) Technology Architecture Overview

---

[1] Recommended but not required.