

CSEN 702: Microprocessors Winter 2014

Project First Deliverable

Deliverable Name: Memory Hierarchy Simulator

Grading weight: The full project accounts for 20% of the course mark (15% for the implementation and 5% for the report). The first project deliverable accounts for 50% of the project mark (i.e., 10% of the course mark: 7.5% for the implementation and 2.5% for the report). A bonus up to 5% of the deliverable mark will be given for implementing at least **two** of the *bonus* features suggested below. Please do not be tempted to implement more than three bonus features, as this will cost you too much time.

Deliverable Overview: The goal of this deliverable is to develop a simulator capable of assessing the performance of the memory hierarchy of a simplified 16-bit RISC processor. This document details the instruction set to be supported, the inputs to the simulator, and the expected outputs.

Implementation language: Any general purpose programming language (preferably an object-oriented language like C++, Java, or C#.NET). The resulting application can either be a console application or a graphical user interface (GUI) application as a *bonus* feature (as explained at the end of the document).

Team Size: 1 to 5 students (preferably 2 or more). You should send your teams to the TA by email on Thursday, November 6th 2014 or earlier.

Important Plagiarism notice: You have to write your own code from scratch. Deliverables based on others code will receive a grade of **zero** in the entire deliverable (even if the code is heavily re-factored/modified, etc...). Examples of such sources include (but is not limited to) code coming from the following sources: other teams, previous year projects, open-source software, tutors, etc...

Deliverable Deadline: Thursday, November 20th, 2014. Evaluations will take place in the following week according to a schedule that will be posted on the MET website.

Instruction set architecture (ISA): The simulator assumes a simplified RISC ISA inspired by the ISA of the Ridiculously Simple Computer (RiSC-16) proposed by Bruce Jacob. As implied by its name, the word size of this computer is 16-bit. The processor has 8 general-purpose registers R0 to R7 (16-bit each). The register R0 always contains the value 0 and cannot be changed. The instruction format itself is not very important to the simulation and therefore is not described here. However, the simulator should support the following set of instructions (16-bit each):

1. Load/store

- Load word: Loads value from memory into regA. Memory address is formed by adding imm with contents of regB, where imm is a 7-bit signed immediate value (ranging from -64 to 63).
 - LW regA, regB, imm
- Store word: Stores value from regA into memory. Memory address is computed as in the case of the load word instruction
 - SW regA, regB, imm

2. Unconditional branch

- Jump: branches to the address PC+1+regA+imm
 - JMP regA, imm

3. Conditional branch

- Branch if equal: branches to the address PC+1+imm if regA=regB
 - BEQ regA, regB, imm

4. Call/Return

- Jump and link register: Stores the value of PC+1 in regA and branches (unconditionally) to the address in regB.
 - JALR regA, regB
- Return: branches (unconditionally) to the address stored in regA
 - RET regA

5. Arithmetic

- Add: Adds the value of regB and regC storing the result in regA
 - ADD regA, regB, regC
- Subtract: Subtracts the value of regC from regB storing the result in regA
 - SUB regA, regB, regC
- Add immediate: Adds the value of regB to imm storing the result in regA
 - ADDI regA, regB, imm
- Nand: Performs a bitwise NAND operation between the values of regB and regC storing the result in regA
 - NAND regA, regB, regC
- Multiply: Multiplies the value of regB and regC storing the result in regA
 - MUL regA, regB, regC

Simulator inputs:

1. *Memory hierarchy*: The system being simulated must have separate L1 data and instruction caches. However, the user of the simulator may wish to use additional levels of cache. For this reason, the user should enter the number of cache levels he wants to simulate. For each cache level, the user should specify: 1) the full cache geometry (S, L, m), 2) writing policies (both in case of hit and miss), and 3) the number of cycles required to access data. The user should also specify main memory access time (in cycles). Memory capacity is assumed to be 64 Kbytes (to be addressable using 16-bits).
2. *Assembly program*: After specifying all the required information above, the user should be able to input an assembly program to be simulated on the specified

- machine. He should also specify its starting address (where the program's first instruction should be loaded in the memory).
3. *Program data:* Finally, the user should specify any data required by the program to be initially loaded in the memory. For each data item both its value and memory address should be specified.

Simulation: The simulator should simulate the memory hierarchy behavior. The program should record the number of instructions executed, the number of cycles spent to access memories, and the number of cache accesses and misses in each cache level.

Simulator output: At the end the simulator should display the following performance metrics:

1. The hit ratio of each cache level
2. The global AMAT of the memory hierarchy (in cycles)

Simplifying assumptions:

1. No virtual memory
2. No floating point instructions, registers, or functional units
3. No input/output instructions are supported
4. No interrupts or exceptions are to be handled
5. For each program being executed, assume that the program and its data are fully loaded in the main memory and that the cache is initially cold (nothing is cached at the beginning of a program execution)

Project Report: In addition to your team member names, the report should include:

1. A brief description of your implementation including any bonus features included
2. A summary of how the work was split among your team members (who did what exactly)
3. A user guide including a full simulation example step-by-step with snapshots.
4. A list of programs (and associated data if any) you simulated. You should at least provide 3 programs. The programs must cover all instructions supported and one of them at least must have a loop.
5. The memory hierarchy you used to simulate each program and the results obtained from each simulation. You must simulate each program with at least two different memory hierarchies.
6. A brief discussion of the obtained results.

Bonus features:

1. Building the application as a GUI application
2. Building the application as an educational GUI application. In this case the user should not only be allowed to simulate entire programs at once and get performance metrics but also to step through the program cycle-by-cycle or instruction-by-instruction while monitoring the internals of the processor. This feature is also helpful for debugging purposes and will be counted as two features as

- far as grading is concerned (since it includes and extends the previous bonus feature).
3. Implementing and integrating a simple assembler to allow the user to supply programs in a suitable assembly language. The assembly can be entered directly in the application or in a text file read by the application.
 4. Provide a relatively large set of benchmark programs (not less than 12 programs) illustrating the effect of various architectural design choices.
 5. Provide a comparative study illustrating the effect of each of the cache geometry parameters on the overall AMAT for a given benchmark.