UiT

THE ARCTIC
UNIVERSITY
OF NORWAY

# Mandatory Assignment 3

INF-2200 (fall 2018)
Department of Computer Science
University of Tromsø

UNIVERSITY · OF TROMSØ ·

# In a sentence

Implement a hierarchical memory sub-system with two cache levels, used by a CPU simulator.

# Details

- Two cache levels
  - Level 1
    - Read-only instruction cache
    - Data cache
  - Level 2
    - Unified cache
- Precode provides CPU simulator that will perform memory accesses against your memory subsystem.
  - You will implement memory.c
  - No changes to precode necessary!

# Goal

- Best cache design
  - Use benchmark from assignment 1 (or other benchmark).
  - Measure cache hit and miss ratio.
  - Experiment with different parameters.

# Precode

- Small and straight forward.
- Implements API and starting point for memory subsystem.
- Implements CPU simulator that does
  - Instruction fetch
  - Load data
  - Store data
- Memory trace stored in binary format
  - P2addrTr struct (byurt.h)
- Use valgrind to generate memory trace logfile.
- Precode provides script for converting logfile to binary trace file read by CPU simulator.

# Generate trace

- Step 1
  - Run the following command:

    ```
    valgrind --log-file=logfile --tool=lackey --trace-mem=yes [your-program-name]
    ```

  - This will create a file `trace.tr`that contains the memory trace of your program.
- Step 2
  - Parse the trace file by running:

    ```
    python traceconverter.py
    ```

  - This will produce a file *logfile* that can be used as input to the cache simulator.
- Step 3
  - Run the cache simulator:

    ```
    ./cachesim logfile
    ```

  - The precode will initialize your memory subsystem by calling *memory_init()* and will then, for each memory access in the *logfile* , call one of the functions:
    - memory_fetch() – if the memory access is an instruction fetch.
    - memory_read() – if the memory access is a data read.
    - memory_write() – if the memory access is a data write.

# Requirements

- Implementable in real hardware
  - Parameters realistic according to book
- Parameters
  - Easily changeable
  - Start with parameters given in assignment text
- L1 data cache and L2 unified cache should support both reads and writes.
  - L1 instruction cache should be read only.

# Requirements

- Simplifications
  - Assume each data access is within boundary of one cache line.
  - Assume all instructions of fixed size, and aligned.
    - Neither is actually true on x86…
  - Not necessary to implement reads and writes that actually *transfer data*, just count cache hits and misses.
- Select a replacement policy and implement it
  - Random
  - LRU
  - Temporal/Spatial
- Write policy
  - Write-back
  - Write-through

# Count

- Hits and misses
  - Differentiate
    - Layers
    - Reads
    - Writes

# Method of approach

- Evaluate cache by creating memory trace
  - Use valgrind and convert with python script
- Two traces
  - Correctness (hits and misses known in advance)
    - May have do be created manually
  - Trace from benchmark

# Report

- Cache performance tweaks
  - How?
  - Why? / Why not?
  - Temporal/Spatial
- Correctness test
- Reductions and simplifications

# Deliverables

- Code
- Written report
  - **Maximum 6 pages**
  - One report per group (if working in groups)
  - Goal: expert reader should be able to redo your work by reading only the report.
- The repository must contain:
  - A directory named “doc”, containing the report.**pdf**
  - A directory named “src” containing code, Makefiles, READMEs
    - **NO compiled files.** Delete executables etc before you hand in
    - README must contain how to compile and run the code
  - A file named after your UiT username
    - Or all usernames of the group seperated by hyphen ( - )
    - E.g. abc001-cde002
  - Groups only need one repository
  - Maximum two per group

# Deadline

- November 2nd @ 12:00 PM (noon)
- Hard deadline (no extensions possible)!