# ΕΠΛ 605
# Εργαστήριο 5

Παναγιώτα Νικολάου

11/10/18

Slides from: Rajagopalan Desikan, Doug Burger, Stephen Keckler, Todd Austin

# Simulators

***"Simulation is the process of designing a model of a real system and conducting experiments with this model for the purpose of either understanding the behavior of the system and/or evaluating various strategies for the operation of the system."***

*Introduction to Simulation Using SIMAN (2nd Edition)*

- An architectural simulator is:
  - a tool that reproduces the behavior of a computing device
- Why we use a simulator?
  - Leverage a faster, more flexible software development cycle

**Functional simulators** implement the architecture.

- Perform real execution
- Implement what programmers see

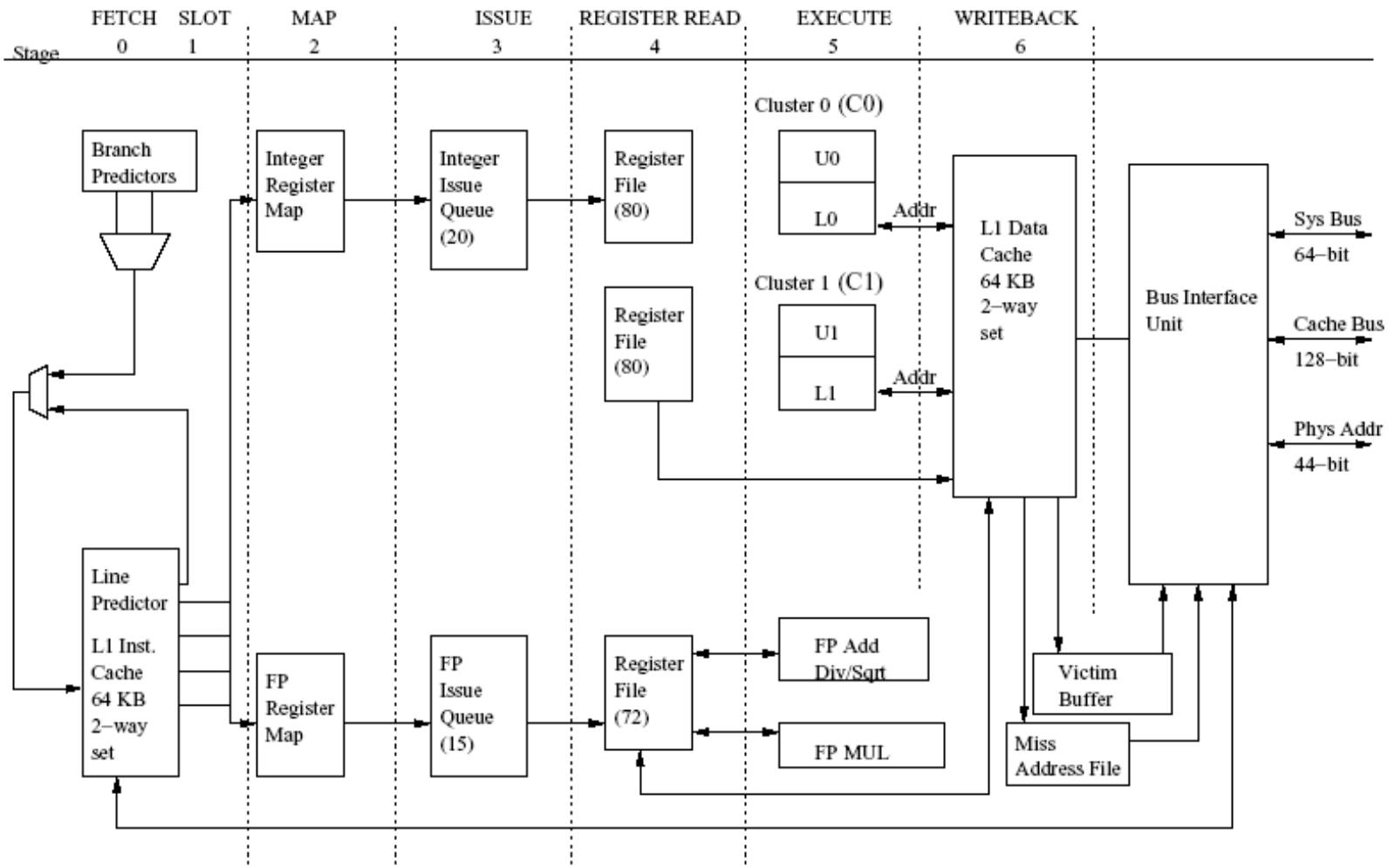
**Performance simulators** implement the microarchitecture.

- Model system resources/internals
- Concern about time
- Do not implement what programmers see

# What is Sim-Alpha ?

- sim-alpha: execution-driven simulator
- Execution-driven simulation is the most accurate simulation technique
  - Detailed simulation of the memory system and the processor pipeline are done simultaneously.
- It models the implementation constraints and the performance low-level features in Alpha 21264.

# EV6 Pipeline



# Pipeline stages

- **Fetch stage & Slot stage :**
  - Instruction cache access
  - Fetch_width number of instructions per cycle (default:4)
  - Static assignment of instructions
  - Control instructions access the branch predictor.
- **Map stage:**
  - Identifies the input and output registers
  - Checks for availability of reorder buffer entry, integer or floating point issue queue entry, physical output register and load or store queue entry (if instruction is load or store).
  - If input physical registers are ready, instruction is placed in ready queue.
- **Issue stage:**
  - Picks instructions from ready queues, checks the availability of functional units and issues the instruction to FUs. Register read latency is charged here. Events are set up for queue entry release and instruction completion.

# Pipeline Stages (Cont.)

- **Writeback stage:**
  - Wakes up the dependent instructions when a producing instruction completes.
  - Load instructions access the D-cache
  - Mispredictions are indicated in the corresponding reorder buffer entry
- **Commit stage:**
  - Retires instructions from reorder buffer
  - Examines the head of reorder buffer for mispredictions and flushes the pipeline in these cases.

# Basic structures

- The main loop of the simulator, located in simulate.c

```
while(TRUE) {
  eventq_service_events();
  commit_stage();
  writeback_stage();
  issue_stage();
  map_stage();
  slot_stage();
  fetch_stage();
}
```

- File: fetch.c
- Accesses the instruction cache and fetch a number of instructions (fetch_width) per access

# Basic structures

- The main loop of the simulator, located in simulate.c

```
while(TRUE) {
  eventq_service_events();
  commit_stage();
  writeback_stage();
  issue_stage();
  map_stage();
  slot_stage();
  fetch_stage();
}
```

- File: slot.c
- Static assignment of instructions to either upper or lower subclusters
- Control instructions access the branch predictor

# Basic structures

- The main loop of the simulator, located in simulate.c

```
while(TRUE) {
  eventq_service_events();
  commit_stage();
  writeback_stage();
  issue_stage();
  map_stage();
  slot_stage();
  fetch_stage();
}
```

- File: map.c
- Identifies the input and output registers
- Checks for availability of reorder buffer entry, integer or floating point issue queue entry, physical output register and load or store queue entry (if instruction is load or store).
- If input physical registers are ready, instruction is placed in ready queue

# Basic structures

- The main loop of the simulator, located in simulate.c

```
while(TRUE) {
  eventq_service_events();
  commit_stage();
  writeback_stage();
  issue_stage();
  map_stage();
  slot_stage();
  fetch_stage();
}
```

- File: issue.c
- Picks instructions from ready queues, checks the availability of functional units and issues the instruction to FUs. Register read latency is charged here. Events are set up for queue entry release and instruction completion.

# Basic structures

- The main loop of the simulator, located in simulate.c

```
while(TRUE) {
  eventq_service_events();
  commit_stage();
  writeback_stage();
  issue_stage();
  map_stage();
  slot_stage();
  fetch_stage();
}
```

- File: writeback.c
- Wakes up the dependent instructions when a producing instruction completes.
- Load instructions access the D-cache
- Mispredictions are indicated in the corresponding reorder buffer entry

# Basic structures

- The main loop of the simulator, located in simulate.c

```
while(TRUE) {
  eventq_service_events();
  commit_stage();
  writeback_stage();
  issue_stage();
  map_stage();
  slot_stage();
  fetch_stage();
}
```

- File: commit.c
- Retires instructions from reorder buffer
- Examines the head of reorder buffer for mispredictions and traps and flushes the pipeline in these cases.

# Basic structures

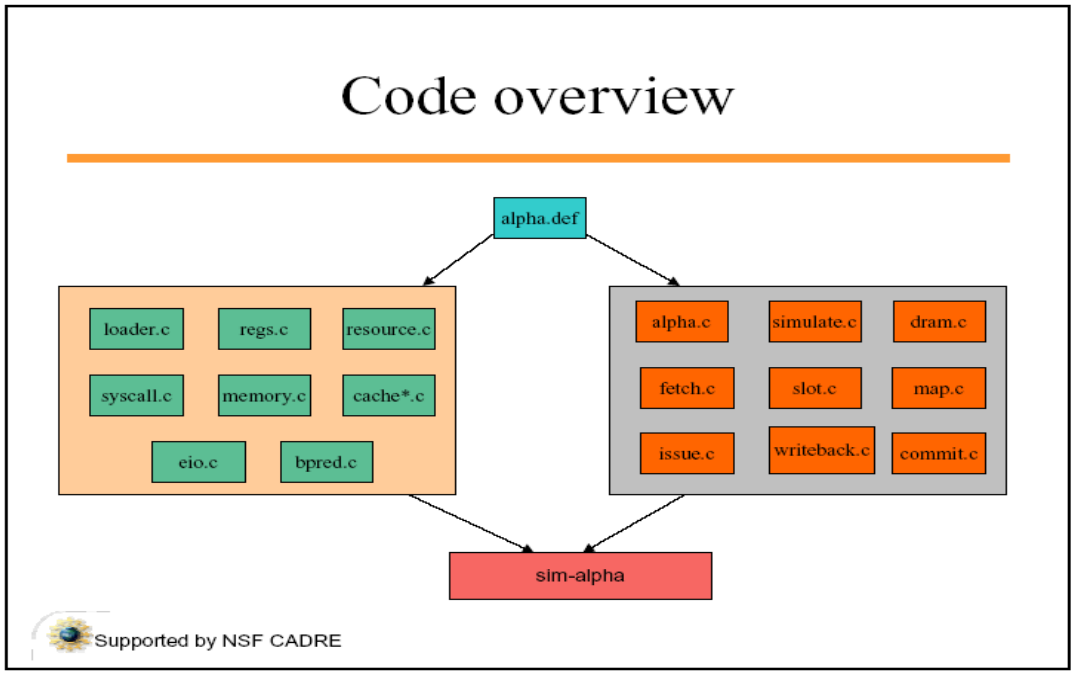
- The main loop of the simulator, located in simulate.c

```
while(TRUE) {
  eventq_service_events();
  commit_stage();
  writeback_stage();
  issue_stage();
  map_stage();
  slot_stage();
  fetch_stage();
}
```

- File: eventq.c
- Schedule the memory code
- At the beginning of each cycle this function checks for memory operations completing this cycle

# Code Structure

- Code for each pipeline stage in a separate .c file
- Each .c file has corresponding .h file containing function prototypes, constants, and extern statements for global variables.



# Using sim-alpha

- List of some options found in sim-alpha
  - **-config** Configuration file to use
  - **-max:inst** Maximum number of committed instructions to simulate
  - **-fastfwd** Number of instructions to fast forward (Caches are not kept warm during this phase)

  Processor core configuration
  - **-mach:freq** Frequency of simulated machine
  - **fetch:width** Number of instructions to fetch each cycle
  - **slot:width** Number of instructions which can be slotted per cycle
  - **-cache:define** - <name>:<nsets>:<bsize>:<subblock>:<asso>:<repl>:<lat>:<trans>:<#resources>:<rescode>

# Small example of a configuration file (small part)

```
#CPU Frequency-mach:freq                    3000000000
# Set issue and commit widths
-issue:intwidth          4 # Integer inst issue width(in insts)
-issue:fpwidth           2 # fp inst issue width(in insts)
-commit:width             4 # commit width(in insts)

 # cache configuration
-cache:define                    DL1:256:64:0:4:l:3:vipt:0:1:0:Onbus
-cache:define                    IL1:256:64:0:4:l:1:vivt:0:1:0:Onbus
-cache:define                    L2:4096:64:0:8:l:12:pipt:0:1:0:Membus

-prefetch:dist     0  # Number of blocks to prefetch on a icache miss
-res:delay                                  0

# flush caches on system calls
-cache:flush                     false

# defines name of first-level data cache
-cache:dcache                    DL1

# defines name of first-level instruction cache
-cache:icache                    IL1

# number of regular mshrs for each cache
-cache:mshrs                     8
```

# How to use Sim-Alpha(cont..)

| | Most common Options |
|---|---|
| -config | Configuration file to use |
| -max:inst | Maximum number of committed instructions to simulate |
| -fastfwd | Number of instructions to fast forward before start simulation. Note the caches are not kept warm during this phase |

**Example:**

**../alpha-sim/sim-alpha -config conf.cfg -max:inst 10000 ./cpu2006_434.zeusmp_kehoste_Alpha_gcc411_O**