Learning gem5 – Part IV

gem5 execution model, ISAs, and CPUs

Jason Lowe-Power

http://learning.gem5.org/

https://faculty.engineering.ucdavis.edu/lowepower/

Outline

ISA independence and execution model

LearningSimpleCPU

gem5's many CPU models and types of execution

Full system vs Syscall emulation mode

gem5 ISAs

src/arch/

- alpha
- arm
- hsail
- mips
- power
- riscv
- sparc
- x86

Not all equally well supported. **ARM, X86** most used/tested.

Each directory contains devices, ISA-specific objects, system interface, **ISA definition**

ISA definition

src/arch/<isa>/isa

A domain-specific language for ISAs

- Written in python (src/arch/isa_parser.py)
- Honestly, very confusing, not much documentation
- Output in build/…/generated

Decodes instructions (decoder/*.isa)

Implements instructions (insts/*.isa)

- This is what is called when an instruction “executes” (we’ll see)
- Creates “StaticInst” classes

StaticInst

src/cpu/static_inst.hh

Describes the kind of instruction (`isNop()`, `isInteger()`, etc.)

Provides implementation for execution (parameter: ExecContext)

- `execute(…)` **`execute`**: Modify ExecContext based on instruction
- `initiateAcc(…)` **`initiateAcc`**: Send memory reference
- `completeAcc(…)` **`completeAcc`**: Like execute for mem insts
- `advancePC(…)` **`advancePC`**: ISA-specific

LearningSimpleCPU

Too much code to go through it all

```
> git remote add jason-github \
    https://github.com/powerjg/gem5.git
> git fetch jason-github
> git checkout learningsimplecpu
```

Should be in mainline…

LearningSimpleCPU

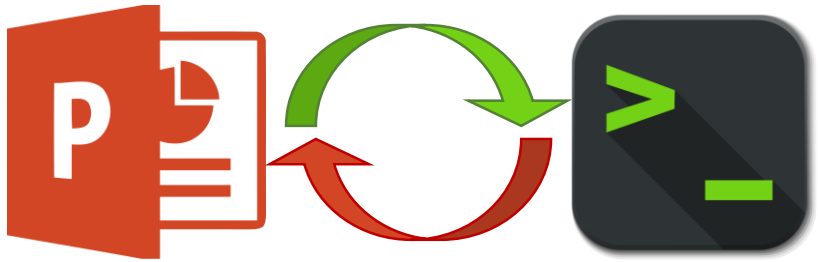
Step 1: Create SimObject

Step 2: Initialization logic

Step 3: Memory request wrapper

Step 4: fetch, decode, execute, writeback

Switch!



The CPU SimObject

Simple unified master port for sending requests

Like SimpleMemObject

SimpleThread: The ***thread*** stores the hardware thread context (regs…)

Must override

init(), startup(), activateContext(), , wakeup(), getPorts, totals

MemoryRequest

Simplifies memory by encapsulating all request/response

Could be used in LSQ implementation…

Provides

translate()

send()

Needs from CPU

finishFetchTranslate()

finishDataTranslate()

decodeInstruction()

dataResponse()

LearningSimpleCPU steps

1) Fetch event (`fetch(Addr)`)
    1) Translate the instruction address (calls `finishFetchTranslate`)
    2) Read memory at the instruction address (calls `decodeInstruction`)
2) Decode (`decodeInstruction(PacketPtr)`)
    1) Call `moreBytes` to prime decoder
    2) Call `decode` to get the `StaticInst`
3) Execute instruction (`executeInstruction(StaticInstPtr)`)
    1) If memory: call `initiateAcc`, wait for translation, and then send request
    2) Otherwise, `execute` the `StaticInst`
4) Increment the PC and schedule another fetch event

CPU Models

gem5 exposes a flexible CPU interface

AtomicSimpleCPU: No timing. Fast-forwarding & cache warming.

TimingSimpleCPU: Single-cycle (IPC=1) except for memory ops.

O3CPU: Out-of-order model. Highly configurable.

MinorCPU: In-order model (not fully tested with x86)

kvmCPU: x86 and ARM support for native execution

Memory modes

Timing

- Used for simulation
- Calls sendTimingRequest, etc.
- All timing is modeled

Atomic

- No timing
- Used for fast-forwarding
- Some structures are warmed up

Atomic_noncaching

- Used for KVM CPU
- Directly manipulates the backing memory

Questions?

We covered

- How ISAs work
- How CPUs execute
- gem5's CPU models
- gem5's memory modes