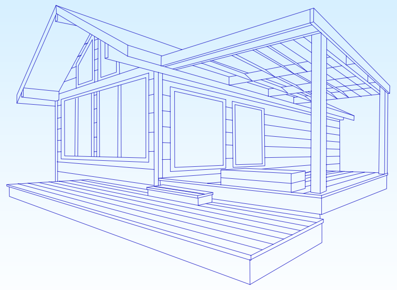
Architectural Simulations



Questions Simulations May Answer

- Capacity questions
  - How big a cache does this processor require?
  - How deep the pipeline should be?
- Performance questions
  - What's the peak performance of the new system on the Linpack benchmark?
- Comparing alternatives
  - Should we schedule the longest instructions first or the shortest instructions first?

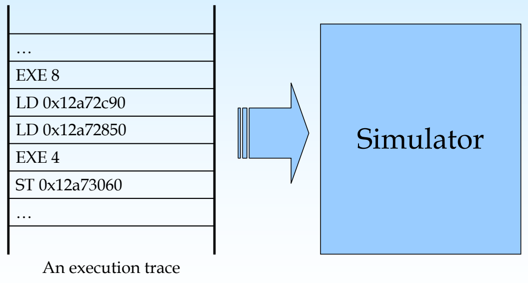
Questions Simulations May Answer

- Fine tuning
  - Is it OK to reduce the size of the input buffer?
- Trouble shooting
  - Which part of the memory system is the bottleneck?
  - Why does the system hang? Is there a deadlock?

Trace-driven Simulations

- Trace-driven simulations use inputs captured from a real system (but not necessarily the system that the simulator models).
  - Typically used for memory system simulations
- Mostly event-based simulators.

Trace-driven Simulations



Trace-driven Simulations

- *Trace collection*: The process of capturing the data sequence.
- *Trace reduction*: filter out the data that is not needed; compress data; etc. Can be combined with collection.
- *Trace processing*: Simulation using the traces.

Trace Collection

- Software execution traces can be obtained through instrumentation. Known as *code annotation*.
  - Trace logging can be manually inserted into the source, if source code is known.
  - Automatic binary instrumentation tools (e.g. *pixie*, *ATOM*) can be used

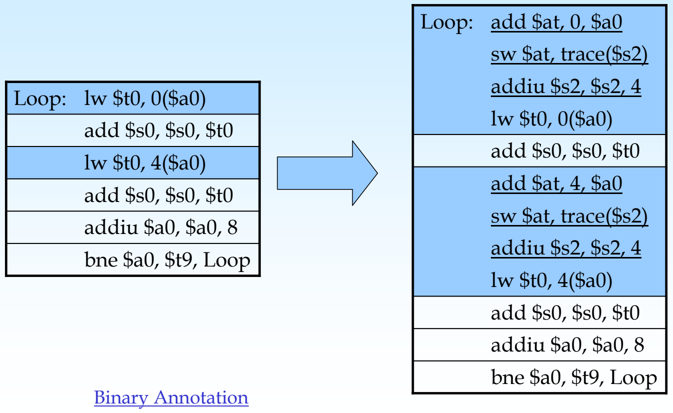
Code Annotation

```
for (i = 0;  i < 10; i +=2) {
   sum += array[i];
   sum += array[i+1];
}
```

```
for (i = 0;  i < 10; i +=2) {
   sum += array[i];
   printf("LD %x", &array[i]);
   sum += array[i+1];
   printf("LD %x", &array[i+1]);
}
```

Source Annotation

Code Annotation



```
Loop:   lw $t0, 0($a0)
        add $s0, $s0, $t0
        lw $t0, 4($a0)
        add $s0, $s0, $t0
        addiu $a0, $a0, 8
        bne $a0, $t9, Loop
```

```
Loop:   add $at, 0, $a0
        sw $at, trace($s2)
        addiu $s2, $s2, 4
        lw $t0, 0($a0)
        add $s0, $s0, $t0
        add $at, 4, $a0
        sw $at, trace($s2)
        addiu $s2, $s2, 4
        lw $t0, 4($a0)
        add $s0, $s0, $t0
        addiu $a0, $a0, 8
        bne $a0, $t9, Loop
```

Binary Annotation

Trace Collection

- Execution traces can also be obtained using performance counters, if they exist
  - Counters are too specific and using them for general trace collection may not be feasible

Trace-driven Simulations

- The traces (i.e. the captured inputs) can be re-used, if stored.
  - Large space may be required for storage
- Pipe the trace directly into the simulator.
  - Trace collection process runs concurrently with the simulation
  - Sometimes called an *execution-driven* simulation
  - Advantage: no storage space required.

Trace-driven Simulations

- The pipe may be a socket-type channel from another machine, or a pipe from another process on the same machine
  - Where more than one pipe is required, named pipes or message queues can be used.
- Other execution-driven solutions: procedure calls or RPC; memory-stream

Trace Collection

- Trace generation process needs to ensure that the normal course of events and the use of resources (such as memory) are not perturbed
  - This may be difficult to achieve, but one must ensure that the errors caused by the perturbation does not affect the validity of the model

Execution Profiling

- Execution profiling is useful to determine program parts that consume most of the execution time.
  - Typically used for program optimization
- Code annotation need only consider those parts that execute most.
  - This discards the traces that generally don't count.
  - Need to ensure that traces that count aren't discarded
    - Compare results with complete traces & reduced traces

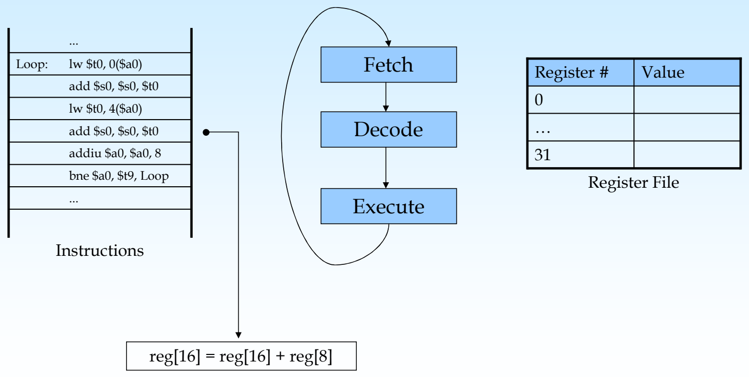
Instruction Emulators

- Instruction emulators execute instruction set of one ISA (target ISA) on another ISA (host ISA).
  - E.g., *SimpleScalar* (www.simplescalar.com)
- Emulators are quite useful in
  - developing software for a machine that has not been built
  - migrating to a new ISA (e.g. Pentium to Itanium)
- They are also useful for studying a new system or investigating enhancements to an existing system.

Instruction Emulators

- Typically execute instructions to provide the results of execution.
  - In general, does not simulate architectural components (e.g. pipelines, caches, etc.)
  - Some call it *execution-driven* simulation!
- Mostly a time-based simulator. The time unit is an instruction step or the clock tick.
  - Detailed simulators that take into account pipelines etc use clock tick as the time unit

Instruction Emulators



Instruction Emulators

- Most emulators run as user-level processes.
- Real programs need kernel services – emulators may intercept system calls and pass them onto the underlying host OS for handling
  - Performance measurement can be skewed because of system calls

Instruction Emulators

- The Java virtual machine and the CLR virtual machine are emulators – the target ISA is the Java byte code/CLI assembly while the host ISA is the ISA that the VM runs on
  - Just-in-time compilation to speed up emulation
  - Compiled code fragments cached for repeated execution
- The emulators can be in hardware. E.g. Transmeta processors (www.transmeta.com).

Further Reading

- Uhlig, R. and Mudge, T., "Trace-driven Memory Simulation: A Survey", ACM Computing Surveys, 29 (2), pages 128-170, 1997.
- Eustace, A. and Srivastava, A., "ATOM: A flexible interface for building high performance program analysis tools", In the *Proceedings of the Usenix Conference on Unix and Advanced Computing Systems*, pages 303-314, 1995.

Further Reading

- Cmelik, B. and Keppel, D., “Shade: A fast instruction-set simulator for execution profiling”, In *Proceedings of the SIGMETRICS Conference on Measurement & Modeling of Computer Systems*, pages 128-137, 1994.