

SATSim Trace Simulator

Mark Wolff

Introduction

SATSim (Superscalar Architecture Trace Simulator) is a visual simulator for out of order execution microprocessors. The program reads instructions from a trace file and displays them as they progress through a simulated microprocessor. The user can configure the microprocessor by selecting the superscalar factor, the number of reservation stations per execution unit, the number of rename buffer entries, the number of reorder buffer entries, and the number of each type of execution unit.

The program runs on Windows 9x or NT. (Performance is noticeably better on Windows NT). The program was developed using Microsoft Visual C++ Version 6.

Description of opening dialog

The user enters the following data at the start of the program. After the simulation begins, the program must be restarted to change any of these values.

Superscalar Factor – The number of instructions that can be fetched and/or completed in one clock cycle. This value determines the width (instructions) of the fetch, decode, issue, and commit stages in the microprocessor. The range of allowable values is 1 to 16.

of Reservation stations per execution unit – Each execution unit needs at least one reservation station to collect the source data from completing instructions. With more than one reservation station, instructions can begin execution as soon as its source operands are available even if a prior instruction has not started. The value entered here applies to all execution units in the processor. The range of allowable values is 1 to 8.

of Integer Execution units – The number of integer ALUs that the microprocessor has. This value determines the number of integer instructions that the microprocessor can execute in a single cycle. The range of allowable values is 1 to 8.

of Floating Point Execution units - The number of floating point units that the microprocessor has. This value determines the number of floating point instructions that the microprocessor can execute in a single cycle. The range of allowable values is 1 to 8.

of Branch Execution units – The number of branch units that the microprocessor has. This value determines the number of branch instructions that the microprocessor can execute in a single cycle. The range of allowable values is 1 to 8.

of Memory Execution units – The number of memory units that the microprocessor has. This value determines the number of load and store instructions that the microprocessor can execute in a single cycle. The range of allowable values is 1 to 8.

of Rename Entries – The number of entries in the microprocessor's register renaming buffer. This value limits the number of instructions, which potentially modify an architectural

register, that can simultaneously be ‘in-flight’. Any instruction that has issued (left the issue stage) and has not committed (left the commit stage) is considered in-flight. The range of allowable values is 0 to 500.

of Reorder Entries – The number of entries in the microprocessor’s instruction reorder buffer. This value limits the number of instructions that can simultaneously be ‘in-flight’. Any instruction that has issued (left the issue stage) and has not committed (left the commit stage) is considered in-flight. Instructions that do not modify an architectural register, (branches and stores), do not require a rename entry, but they do require a reorder entry. The range of allowable values is 0 to 500.

Miss Predicted Branches per 1000 branches – Branch prediction miss rate (miss rate = 1 – hit rate). The value here is the number of misses per 1000 branches (1 \Leftrightarrow 0.1%). Each time the simulator fetches a branch instruction, a random number from 0 to 999 is generated. If the random number is less than the value entered here, the branch instruction is marked as miss-predicted. The random number generator replaces the Branch History Table and Branch Target Address Cache that would be in a real microprocessor. The range of allowable values is 0 to 1000.

I-Cache Miss Rate per 1000 instructions – Instruction Cache miss rate (miss rate = 1 – hit rate). The value here is the number of cache misses per 1000 instructions (1 \Leftrightarrow 0.1%). Each time the simulator fetches an instruction, a random number from 0 to 999 is generated. If the random number is less than the value entered here, the instruction is marked as a cache miss. The range of allowable values is 0 to 1000.

I-Cache Miss Penalty – The number of cycles that the fetch unit stalls on a cache miss. The range of allowable values is 0 to 100.

D-Cache Miss Rate per 1000 Loads – Data Cache miss rate (miss rate = 1 – hit rate). The value here is the number of cache misses per 1000 Loads (1 \Leftrightarrow 0.1%). Each time a Load instruction reaches the end of the memory unit pipeline, a random number from 0 to 999 is generated. If the random number is less than the value entered here, the Load is considered a cache miss. Any instructions in the same memory unit are blocked from further processing until the cache miss is resolved. The range of allowable values is 0 to 1000.

D-Cache Miss Penalty – The number of clock cycles that the memory unit stalls on a cache miss. The range of allowable values is 0 to 100.

Trace file – The path and name of the ascii trace file. The path is relative to the location of the executable file.










Output file – The path and name of the ascii output file. The program will write results to this file at the end of a trace or when the program is terminated.

Overview of menus and toolbar

Menus All menu items can be activated by pressing <Alt> and the key sequence indicated by the underlined letters in the menu.

- File menu - New Trace will start a new simulation
Open Batch is not implemented yet. Meanwhile it has the same effect as New Trace
Close will end the active trace's simulation.
Print, Print Preview, and Print Setup perform their usual Windows function by printing the current state of the simulator.
Exit will end the program.
- View menu – Allows the user to control whether or not the toolbar is displayed and whether or not the status bar at the bottom of the screen is displayed.
- Window menu – Allows the user to open and manage multiple windows to show the simulator.
- Cycle menu - Allows the user to switch between single step mode and several timer modes.
Single Step will shut off the timer if it's active and step the microprocessor through a single clock cycle.
Cycle Timer will start the timer causing the processor to cycle automatically updating the screen after every cycle.
10 Cycle Timer will start the timer causing the processor to cycle automatically updating the screen after every 10 cycles.
100 Cycle Timer will start the timer causing the processor to cycle automatically updating the screen after every 100 cycles.
1000 Cycle Timer will start the timer causing the processor to cycle automatically updating the screen after every 1000 cycles.
Go To End will cause the simulation to run to the end of the trace without updating the screen. This is the fastest way to complete a trace.
- Timer menu – Allows the user to select the timer interval. The interval applies to the 1 cycle, 10, cycle, 100 cycle, and 1000 cycle modes from the Cycle menu.
10ms – This will run as fast as possible. Depending on the computer and OS the speed will range between 15 and 100 cycles per second.
100ms – The simulation will run at 10 cycles per second.
500ms – The simulation will run at 2 cycles per second.
1 sec – The simulation will run at 1 cycle per second.
- Force menu – Allows the user to force certain events to occur.
Branch Mis-Predict – The next branch instruction fetched will be tagged as a mis-prediction.
Icache miss – The next line of instructions fetched will cause an Icache miss
Dcache miss – The next load instruction to reach the last stage of a memory unit will cause a miss in the data cache.
- Help menu – Provides information about the program and how to use it.

Toolbar The toolbar can be dockable or floating. The toolbar can be moved by dragging it to the desired location.

-  Single Step – Stops the timer and cycles the processor through one clock cycle
-  Timer – Starts the timer causing the simulator to run automatically.
-  Go To End of Trace - will cause the simulation to run to the end of the trace without updating the screen.
-  Mis-Predict Branch – Forces the next fetched branch instruction to be mis-predicted.
-  Icache Miss – Forces the next line of instructions to be an Icache miss
-  Dcache Miss – Forces the next load instruction to reach the end of a memory pipeline to be a Dcache miss.
-  Prints the current view on the default printer.
-  Help – Invokes the help system (not currently active)
-  Context sensitive help (not currently active)

Hot Keys – F3 - Single Step – Stops the timer and steps the processor through on clock cycle.
F4 - Timer – Start the timer causing the simulator to run automatically.
F5 – Go to End of Trace - will cause the simulation to run to the end of the trace without updating the screen.

Description of processor

The first three numbers displayed at the top of the processor are number of cycles, number of instructions committed, and IPC. The remaining numbers are utilization percentages. ei, ef, eb, em, ri, rf, rb, rm, rn, and ro are utilization values for integer execution units, floating point execution units, branch execution units, memory execution units, integer reservation stations, floating point reservation stations, branch reservation stations, memory reservation stations, rename buffer, and reorder buffer respectively.

Fetch

The fetch stage first determines if there is an Icache miss. If there is an Icache miss, “Icache Miss” is displayed in red. The Icache miss is effective for the number of cycles set up as the Icache miss penalty. If there is no Icache miss and the decode stage is entirely empty, then a new line of instructions is fetched (from the ascii trace file). If the decode stage is not empty, the fetch stage stalls.

The addresses of instructions are aligned in the fetch unit. The address of the first instruction determines where it is placed. Instructions are fetched until the end of the fetch line is reached, a

taken branch is encountered, or a mis-predicted branch is encountered (taken or not taken). The end of the fetch line is reached when the instruction address is divisible by the superscalar factor. A taken branch is encountered when the instruction address jumps by a value other than 4 bytes. A mis-predicted branch is determined by random number generator or is forced by the user.

Decode

This is the stage where unneeded instructions from the fetch stage are removed. These comprise unneeded instructions fetched at the front of a line due to address mis-alignment, and unneeded instructions at the end of a line due to a predicted taken branch. Also, instructions are decoded to determine what kind of instruction they are and what their source and destination registers are. This information is passed on to the issue stage.

In the simulator, the instruction number and instruction word address are displayed. The instruction number is black numbers on white and is just an identifier that the simulator assigns the instruction so that it can be tracked through the processor, (a number from 0 to 499). The word address is white numbers on black and corresponds to the PC at fetch divided by four (32-bit instructions). The word address is shown to indicate why certain instructions were fetched and why they were placed in their respective slots in the decode stage.

As many instructions are passed on to the issue stage as there is room in the issue stage.

Issue

The issue stage will issue, in order, as many instructions as it can to reservation stations. The first instruction encountered that can not issue halts the issuing. Any instructions that can not issue are shifted left to make room for more instructions coming from the decode stage on the next cycle.

There are two rows of information shown for each instruction in the issue stage. The top row shows the instruction number and the type of instruction. The second row shows the architectural destination and source registers if there are any. The destination register is on the left and is in blue letters. Destination registers are shown in black letters.

An instruction will issue to a reservation station, corresponding to an execution unit of the appropriate type, unless one of the following is true:

1. An instruction of the same type has already issued this cycle to each of the corresponding execution units.
2. All the reservation stations for the corresponding execution units that have not been issued to are full.
3. The instruction modifies an architectural register and there are no rename entries available.
4. There are no reorder buffer entries available.

When an instruction issues the following things happen:

1. If the instruction has a destination register, it acquires a color from the rename buffer. This is indicated in the rename buffer by the colored instruction number and corresponding architectural register.
2. Also, if the instruction has a destination register, the colored instruction number is shown in the box for that register in the register file. This is so that any future instructions that depend on this instruction's result will know which rename entry to retrieve it from.
3. All instructions are assigned a reorder buffer entry. This is indicated by the colored instruction number appearing in the first available reorder entry.
4. The instruction is assigned a reservation station. This is indicated by the instruction number appearing on the left side of the reservation station.
5. If an instruction issues and all the source operands are immediately available and the execution unit is not receiving another instruction from its reservation stations, the instruction will issue directly to the execution unit. This is indicated by the instruction number appearing on the left side of the first stage in the execution unit.

Register File

This shows the current rename status of each of the architectural registers. There are 32 integer registers (0-31), 32 floating point registers (0-31), and a floating point status register (fsr). An empty box indicates that the register is not currently renamed, (the register holds the current correct value). If the register has been renamed, the box shows the colored instruction number of the instruction that will produce the newest value for this register. The color corresponds to the rename buffer entry.

It is important to note that the integer register, r0, always contains the value zero. Therefore, there is no reason to rename instructions that write to this register.

Rename Buffer

Empty boxes indicate rename entries that are not currently in use. While in use, the box will hold the instruction number of the instruction that will produce the result, and the name of the architectural register that it is standing in for. If the instruction number is colored then the instruction has not completed execution. An instruction number that is shown black on white indicates an instruction that has completed and broadcasted its results, but has not yet committed (written back).

Reorder Buffer

The reorder buffer holds the instruction number of all in-flight instructions. If the instruction number is colored or is white letters on black, then the instruction has not completed execution. If the instruction number is black numbers on white, then the instruction has completed execution and is waiting its turn to commit and write back.

Reservation Stations

Reservation stations hold the instructions that are waiting to begin execution or are currently executing. The first number is the colored instruction number. Branches and stores have no color since they do not change the value of any architectural registers. The second and third numbers, if present, are the colored instruction numbers of any source operands. If there is no

number or if all the numbers are not colored (black numbers on white), then the instruction has all the source operands it needs and is either executing or is ready to begin execution.

Instructions remain in the reservation station until execution is complete. An instruction can begin execution once all of its source operands are available. Of course, only one instruction can begin execution in any given execution unit in a single cycle. If two instructions are ready to begin execution on the same execution unit on the same cycle, the oldest instruction goes first.

An integer, floating point, or branch instruction can only begin execution in the execution unit directly below the reservation station it was issued to. Memory instructions are much more complicated. Memory instructions can not execute out of order as easily as other instructions. For this simulator, all stores execute in order. This is in case two stores write to the same memory location. The final value in memory has to come from the lexically later store instruction. Also, all load instructions must begin execution prior to any store instruction that follows. This is to prevent loading a value that should have been stored later. All store instructions must begin execution prior to any load instruction that follows. This is to prevent loading an out of date value from memory. The only out of order execution for memory instructions is between consecutive loads. It does not matter what order consecutive loads from the same memory location occur. They will still get the same value.

The memory reservation stations act as a single FIFO queue regardless of how many memory units there are. The memory instructions snake down and issue as many as possible instructions without violating the above rules.

Also, load instructions will display 'LD' in the second source operand space. This is to visibly distinguish between loads and stores on the display. Load instructions never have more than one source operand.

Execution Pipelines

The execution units show the progress of an instruction through the execution pipeline. Integer and branch units have one pipeline stage. Floating point units have three pipeline stages, and memory units have two pipeline stages. Results are broadcast the cycle that the instruction leaves the pipeline, and any instructions waiting for that result can begin execution one cycle later.

When there is a Dcache miss, this is indicated in red letters below the memory execution unit. This can be thought of as the third memory pipeline stage. Only loads cause a cache miss for this simulator. The guilty load instruction will stall in the second stage of the pipeline for the number of cycles set up as the Dcache miss penalty. Succeeding memory instructions in that pipeline will also stall.

Also, load instructions will display 'LD' in the second source operand space. This is to visibly distinguish between loads and stores on the display. Load instructions never have more than one source operand.

Commit Stage

The commit stage shows the instruction number of the instructions that will commit and write back this cycle. The superscalar factor determines how many instructions can commit and write back in a single cycle. Instructions commit in the order they were issued. Instructions must wait in the reorder (and rename) buffer until all preceding instructions have committed. After an instruction commits the following happens:

1. The instruction is removed from the reorder buffer and the reorder entry can be used by an issuing instruction the same cycle.
2. If the instruction is using a rename entry, it is removed from the rename entry and an issuing instruction can use the rename entry on the same cycle.
3. If the instruction is the latest renamed value for the corresponding architectural register then its result value is written to the register. This is indicated by the color shown in the register file going blank.
4. The number of instructions committed is incremented at the top of the screen.

Description of files

SATSim.exe – This is the executable. Double click the icon to start the program.

SATSim.out – This is the default ascii file that holds the results of the simulations.

The file is tab delimited, and is best viewed using a spreadsheet such as Excel. The following is a description of what is recorded in the file.

Run name – for now this has no meaning and will always be “interactive”.

Date, Time – the data and time the simulation was recorded.

Trace – name of the trace file used as input.

SS factor – Superscalar factor

Int units - number of integer execution units

Float units - number of floating point execution units

Branch units - number of branch execution units

Mem units - number of memory execution units

Res per exe - number of reservation stations per execution unit

#Reorder – number of reorder buffer entries

#Rename – number of renaming buffer entries

ICache MR – instruction cache miss rate (misses per 1000 lines fetched)

ICache penalty – instruction cache miss penalty (cycles)

DCache MR – data cache miss rate (misses per 1000 loads)

DCache penalty – data cache miss penalty (cycles)

Branch MR – branch misprediction rate (misses per 1000 branches)

#Cycles – number of clock cycles completed

#Committed – number of instructions that completed and committed

Numint – number of integer instructions fetched

Numfl – number of floating point instructions fetched

Numbr – number of branch instructions fetched

Numld – number of load instructions fetched

Numst – number of store instructions fetched

ICache misses – number of times the fetch stage missed in the instruction cache
DCache misses – number of times a load missed in the data cache
MP branches – number of mispredicted branches
MP Branch cycles – total number of clock cycles due to mispredicted branches
Exe int – total number of cycles each integer execution unit was in use. (integer execution unit utilization is $(\text{Exe int} / \#Cycles / \text{Int units})$).
Exe fl – total number of cycles each floating point execution unit was in use.
Exe br – total number of cycles each branch execution unit was in use.
Exe mem – total number of cycles each memory execution unit was in use.
Res int – total number of cycles each integer reservation station was in use. (integer reservation station utilization is $(\text{Res int} / \#Cycles / \text{Int units} / \text{Res per exe})$).
Res fl – total number of cycles each floating point reservation station was in use.
Res br – total number of cycles each branch reservation station was in use.
Res mem – total number of cycles each memory reservation station was in use.
Reorder use – total number of cycles each reorder entry was in use. (reorder buffer utilization is $(\text{Reorder use} / \#Cycles / \#Reorder)$).
Rename use – total number of cycles each rename entry was in use.
Pipe stalls – number of cycles that the fetch unit stalled due solely to the decode stage not being empty.

P1log.out – This is the error file. If you're having problems with the program you might check this file to see if there are any hints as to why. This file should normally be empty after the program ends.

Nnn.tra – Trace file. You should have received at least one ascii trace file to use as input to the simulator. The file can be changed (with care) and not affect the simulator. But, be careful! As of now, the parser for the simulator is not very robust.

FAQs

Where is the Writeback stage?

This simulator is modeled largely after the PowerPC. In the PowerPC most instructions can commit and write back in the same cycle. The stage labeled as commit actually shows the instructions that are being written back.

Why aren't any instructions displayed in Fetch?

Each of the stages in the simulator display the instructions that are present at the beginning of a cycle. At the beginning of a cycle the fetch unit effectively holds the entire L1 cache. Therefore, the only information displayed in the fetch stage is whether there is an Icache miss or a mis-predicted branch. Instructions that are successfully fetched are displayed in the decode stage on the next cycle.

Why does Fetch stall on a mis-predicted branch?

In a real processor, the fetch unit does not stall after a mis-predicted branch. Instead, the fetch unit would continue to fetch instructions along the incorrect path and feed them to the decode stage. Once the branch execution unit determined that the branch was mis-predicted, all these instructions would be killed. However, since this is a trace simulator, the only instructions

available are those along the correct path. The simulator simulates the mis-predicted branch by stalling the fetch unit until the mis-predicted branch completes execution. This is not an entirely accurate simulation since the instructions along the correct path do not have to compete for resources with instructions along the incorrect path (as would happen in an actual processor).

What's going on in the memory reservation stations?

Memory instructions can not execute out of order as easily as other instructions. For this simulator, all stores execute in order. This is in case two stores write to the same memory location. The final value in memory has to come from the lexically later store instruction.

Also, all load instructions must begin execution prior to any store instruction that follows. This is to prevent loading a value that should have been stored later. All store instructions must begin execution prior to any load instruction that follows. This is to prevent loading an out of date value from memory.

The only out of order execution is between consecutive loads. It does not matter what order consecutive loads from the same memory location occur. They will still get the same value.

So, the memory reservation stations act as a single FIFO queue regardless of how many memory units there are. The memory instructions snake down and issue as many as possible instructions without violating the above rules.

Known Bugs

The fonts display differently on every machine I've tried. Occasionally, this results in unwanted display artifacts remaining on the screen. This is only a cosmetic problem.

The memory unit, cache miss logic, is partially non-blocking (stores) and partially blocking (loads). Fully non-blocking might be the best assumption.

The Icache miss rate is applied as misses per line of instructions fetched. The miss rate should be interpreted as misses per instruction word.