ARTIFICIAL NEURAL NETWORKS FOR BRANCH PREDICTION

By

Brian Adam Dazsi

and

Richard Enbody

MSU-CSE-01-22

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering
Department of Computer Science and Engineering
2001

ABSTRACT
ARTIFICIAL NEURAL NETWORKS FOR BRANCH PREDICTION
By
Brian Adam Dazsi

Current microprocessor technology is being enhanced at an amazing rate. According to “Moore’s Law”, microprocessor transistor count and clock speed doubles every 18 months. With the speed that superscalar microprocessors can execute multiple instructions out-of-order, it is imperative that an extremely efficient branch predictor is implemented on the microprocessor, to keep the frequency of recovering from mispredicted instructions low. With current transistor counts in microprocessors being so high, more complex microprocessor components can now be considered, such as multiple pipelines, larger caches, and more complex branch predictors.

Artificial Neural Networks have recently been showing amazing usefulness in the areas of pattern recognition. They have also been used rather well for prediction applications. Through the use of C programming, the SPEC95 benchmarks, and a microprocessor simulator called SimpleScalar, this thesis explores the possibility of using artificial neural networks for branch prediction. The feed-forward, back-propagation artificial neural network coded for this thesis did not perform as well as expected; however, the area of Artificial Neural Networks is a rapidly growing field, and this thesis is just the beginning of the possibilities for using Artificial Neural Networks in branch prediction.

For my parents, who have helped me become the man I am today,
and my wife, who will continue life's journey with me.

ACKNOWLEDGEMENTS

There are many people throughout the years who have helped me get to where I am today. All of my professors and teachers over the years, and all of my family and friends have made this educational journey a bearable trip. To them I am eternally grateful. There are also a few special people who have helped me with this thesis - the toughest obstacle I had to overcome in all my education. I would like to thank them here.

My thanks go to Dr. Richard Enbody for his extreme patience. There were times that I doubted that I would get this thesis completed, but he never let me think that was an option. I would also like to thank him for his enthusiasm and teaching ability. His Advanced Computer Architecture class was my favorite graduate class - it was the most fun and most rewarding challenge. It instilled the desire to learn and explore further - hence this thesis.

Mark Brehob assisted me with using the SimpleScalar simulator. His few minutes of help here and there saved me from hours of wild goose chasing when I ran into problems that weren't really problems at all, or had simple solutions. His help is priceless.

I would like to thank Jackie Carlson for her endless support, encouragement and great C reference. Without her help I would not have had all the resources I needed to complete this project. Her time to sit and talk put my mind at ease and helped me get through the day-to-day struggles of this thesis and other things that distracted me from the thesis.

Dr. Fathi Salam introduced me to Artificial Neural Networks. His Neural Network classes were a wonderful learning experience, and helped me develop the idea for this thesis. For that I thank him.

My undying love and thanks goes to my wife, Sara, for putting up with me, and helping me through it all. I could not have done it without her support and love. Together we can accomplish anything.

I am thankful to my parents for absolutely everything. For obvious reasons, none of this would be possible without them; but they have also always been a guiding light for me. They have always being there for me - they have never let me down. They have always been tremendous role models. They have instilled in me the personal characteristics and responsibility that have guided me through my education and this thesis. Without those traits, I could never have overcome all of life’s struggles to get this far.

I would like to thank God, for the opportunity to share my talents.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

Background

The number of transistors in a microprocessor is growing at an enormous rate and how to use all those transistors is a topic of much debate. The number of, and specialization of, the execution units in the microprocessor pipeline are increasing. As a result, out-of-order instruction processing is standard practice. Since so many instructions are being fetched and executed out-of-order, when a branch instruction is encountered, it is important that the next instruction fetched is really the instruction that would be fetched if the correct answer to the branch decision was already known (at least two clock cycles are needed in order to process a branch instruction and calculate the branch decision). If the wrong path is chosen, then instructions start being executed that should not be executed and the microprocessor will have to recover from executing those invalid instructions. Therefore, it is imperative to have a good branch predictor.

Artificial Neural Networks are becoming more useful in the areas of pattern recognition and prediction. ANNs are starting to be used alongside standard statistical prediction models used for years in the fields of finance and marketing. The ANNs are performing as well as, if not better than, the statistical models [West]. Because of their success in these fields, an Artificial Neural Network might perform well as a microprocessor branch predictor.

Goals

When this thesis was started, there was no readily available documentation published about using a neural network for branch prediction. The goal of this thesis was to obtain a working simulation to compare a neural network branch predictor with current branch prediction technology. In order to achieve that goal, four tasks were established:

- Develop an Artificial Neural Network

Before an ANN could be used for branch prediction a set of tools and structures needed to be defined. The ANN was to be programmed in C, so that it could be incorporated into SimpleScalar.

- Modify the SimpleScalar simulator to accept an ANN branch predictor

To gather branch predictor performance data and add another predictor to the simulator, the coding for SimpleScalar needed to be explored.

- Train the Neural Network

Training data was to be gathered from a normal run of SimpleScalar and train the neural network. In order to obtain the working simulation in a

reasonable amount of time, a feed-forward, back-propagation neural network using Hebbian learning would be used.

- Evaluate the ANN branch predictor

Finally, information about branch predictor performance was to be gathered by running SimpleScalar for each branch predictor. The branch predictor performance could then be plotted and examined.

Thesis Organization

The rest of the thesis is organized as follows. Chapter 1 discusses the background behind Branch Prediction. The Post-RISC architecture is discussed to show the importance of branch prediction in the microprocessor pipeline. Artificial Neural Networks are discussed in Chapter 2. A brief introduction of neural networks from a biological and mathematical perspective is given, and the Hebbian learning algorithm is talked about. Chapter 3 briefly discusses the SimpleScalar microprocessor simulator, its components and how it was utilized in this project. Chapter 4 discusses the methodologies used to add a new branch predictor to SimpleScalar and programming, verifying and training the neural network. Chapter 5 examines the results of the neural network training and the branch predictors' performance. Finally, the possibilities of future work using Artificial Neural Networks for Branch Prediction are examined in Chapter 6.

Chapter 1: Branch Prediction

1.1 Microprocessor Architecture

To understand the importance of branch prediction, an examination of the overall microprocessor must be done. Most microprocessors today are of a superscalar design, meaning that they execute multiple instructions at once. The Post-RISC architecture [Brehob] is worth a brief examination since most current microprocessors share much in common with this superscalar architecture [Hsieh]. Figure 1.1 [Brehob] shows the generic layout of a Post-RISC processor pipeline.

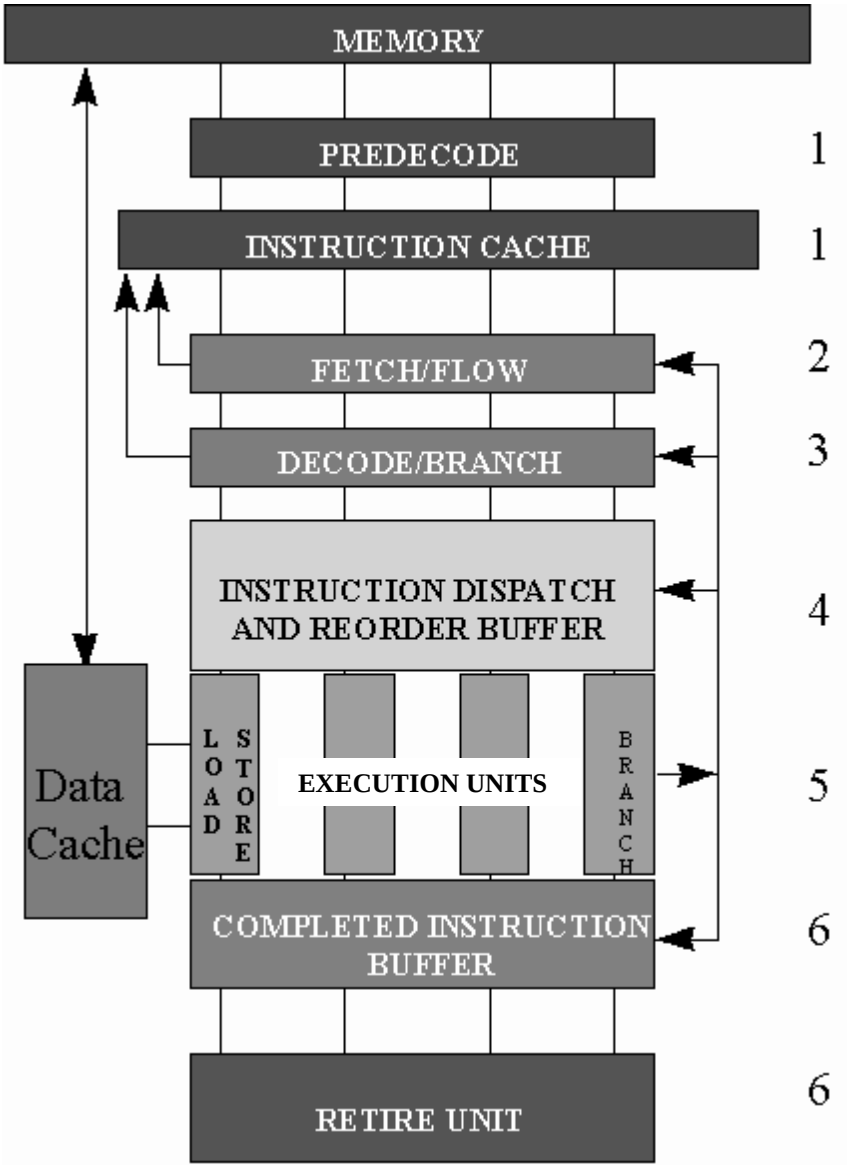


Figure 1.1: The Post-RISC Architecture

From examining Figure 1.1, there are six important steps in the processor pipeline. First, instructions are read from memory in the Predecode stage and stored into the Instruction Cache (I-Cache). In the

superscalar architecture, multiple instructions are read into the cache at one time (for most current processors, four instructions are read [Hsieh]). As a part of predecoding, extra bits are appended to the instructions in order to assist in decoding in a later stage. During the Fetch/Flow stage of the pipeline, which instructions are fetched from the I-Cache is decided. However, it is not until the third pipeline stage, the Decode/Branch stage, that a prediction on a branch instruction is actually made. At this point, "not taken" branches are discarded from the pipeline, and decoded instructions are passed on to the Instruction Dispatch and Reorder Buffer stage. In the Instruction Dispatch and Reorder Buffer stage, instructions are queued up and wait to move on to an available execution unit in stage five. Examples of execution units are load/store units, branch units, floating point units and arithmetic logic units. Which types and how many of each of these execution units is decided by the designers of the microprocessor. At this point in stage five, after the appropriate calculations are done for a branch instruction, the Fetch/Flow stage of the pipeline is informed of branch mispredictions so that it can start recovering from mispredictions. The Branch/Decode stage is also informed of mispredictions so that it can update its prediction methodology (typically, updating of tables or registers). After an instruction is successfully executed it is sent on to the Completed Instruction Buffer (stage six) and the Retire Unit successfully updates the state of registers based on the completed instruction (usually at a rate equal to that of Predecode stage instruction fetching). Instructions could be waiting in the Completed Instruction Buffer until information about a branch instruction becomes available so that they can be retired or erased. Figure 1.1 shows that branch prediction effects multiple stages of the pipeline.

1.2 Branch Prediction

An examination of the superscalar structure presented in Section 1.1 shows the importance of a good branch predictor. With multiple instructions being executed out-of-order in parallel, recovering from a misprediction can be extremely difficult and can cost valuable processor time and resources. Instructions that have been placed in the pipeline that should not have been, have to be nullified, and the correct instructions have to start being fetched. This recovery period is referred to as a stall.

In order to prevent misprediction stalls, a good branch predictor is needed. A few easy methods exist to provide branch prediction. First, a branch could always be assumed "taken" or always be assumed "not taken". In these static cases, the logic for the prediction is extremely simple, but also allows for an average 50% misprediction rate. This is just not acceptable. A more dynamic approach is to use a branch prediction buffer, which can be used to keep track of prediction history. In the simplest form, this history table is referenced by the lower portion of the branch instruction address and uses a 1-bit counter to record if the branch

was recently “taken” or not. This simple method also does not provide for very accurate predictions [Hennessy]. However, a slight modification by adding another bit helps overcome the one-bit method’s shortcoming.

1.2.1 Two-bit Predictor

In this method, two bits are used to keep track of the prediction history. Only after two consecutive mispredictions is the prediction state changed from predict taken to predict not taken. Figure 1.2 [Hennessy] shows how this is achieved in a state diagram.

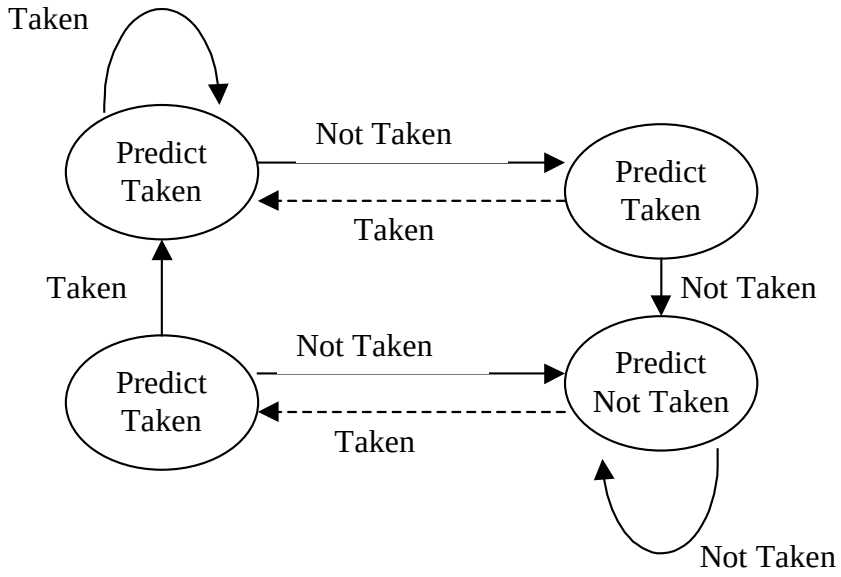


Figure 1.2: Two-bit prediction method states

While this method provides fairly good prediction accuracy, it is still not as ideal as current technology needs, nor as good as current technology can provide.

1.2.2 Two-level Adaptive Branch Prediction

This prediction method, introduced by Tse-Yu Yeh and Yale N. Patt, is one of the most successful prediction methods used today. It provides the most accurate predictions [Yeh93]. In the two-level prediction scheme, two levels of branch history are stored. In the first level, a record of the last n branches is stored in the Branch History Table (BHR). In the second level, the branch behavior of the last x occurrences of a specific pattern in the BHR is recorded. The second level table is called the Pattern History Table (PHT). Figure 1.3 [Driesen] shows the general layout of the two-level method.

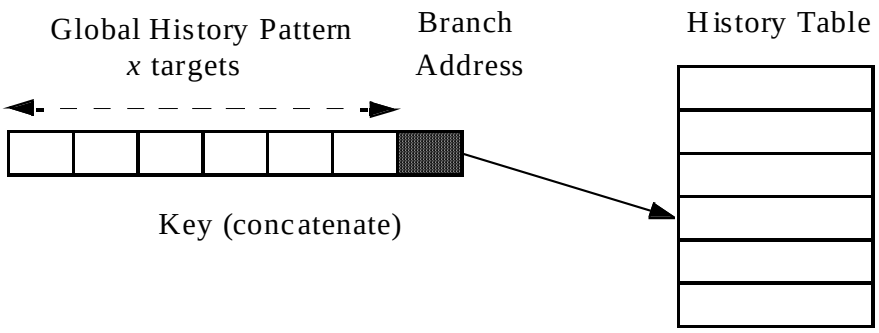


Figure 1.3: Generic Two-Level Branch Prediction

There are three manners in which information is kept in these tables: globally, per-address and per-subset (which was introduced by Pan, So and Rahmeh [Yeh93]). When referring to the Branch History Table, these methods are labeled as GA, PA, SA, respectively. When referring to the Pattern History Table, these methods are labeled as g, p, and s, respectively. This gives rise to nine different variations of the Two-Level Adaptive Branch Predictor, as shown in Table 1.1 [Yeh93].

Table 1.1: Two-Level Branch Predictor Variations

Variation	Description
GAg	Global Adaptive Branch Prediction using one global pattern history table.
GAp	Global Adaptive Branch Prediction using per-address pattern history tables.
GAs	Global Adaptive Branch Prediction using per-set pattern history tables.
PAg	Per-Address Adaptive Branch Prediction using one global pattern history table.
PAp	Per-Address Adaptive Branch Prediction using per-address pattern history tables.
PAs	Per-Address Adaptive Branch Prediction using per-set pattern history tables
SAg	Per-set Adaptive Branch Prediction using one global pattern history table.
SAp	Per-set Adaptive Branch Prediction using per-address pattern history tables.
SAs	Per-set Adaptive Branch Prediction using per-set pattern history tables.

Figure 1.4, Figure 1.5, and Figure 1.6 show how the pattern history tables are referenced in the variations of each type of Two-level Adaptive Branch Prediction.

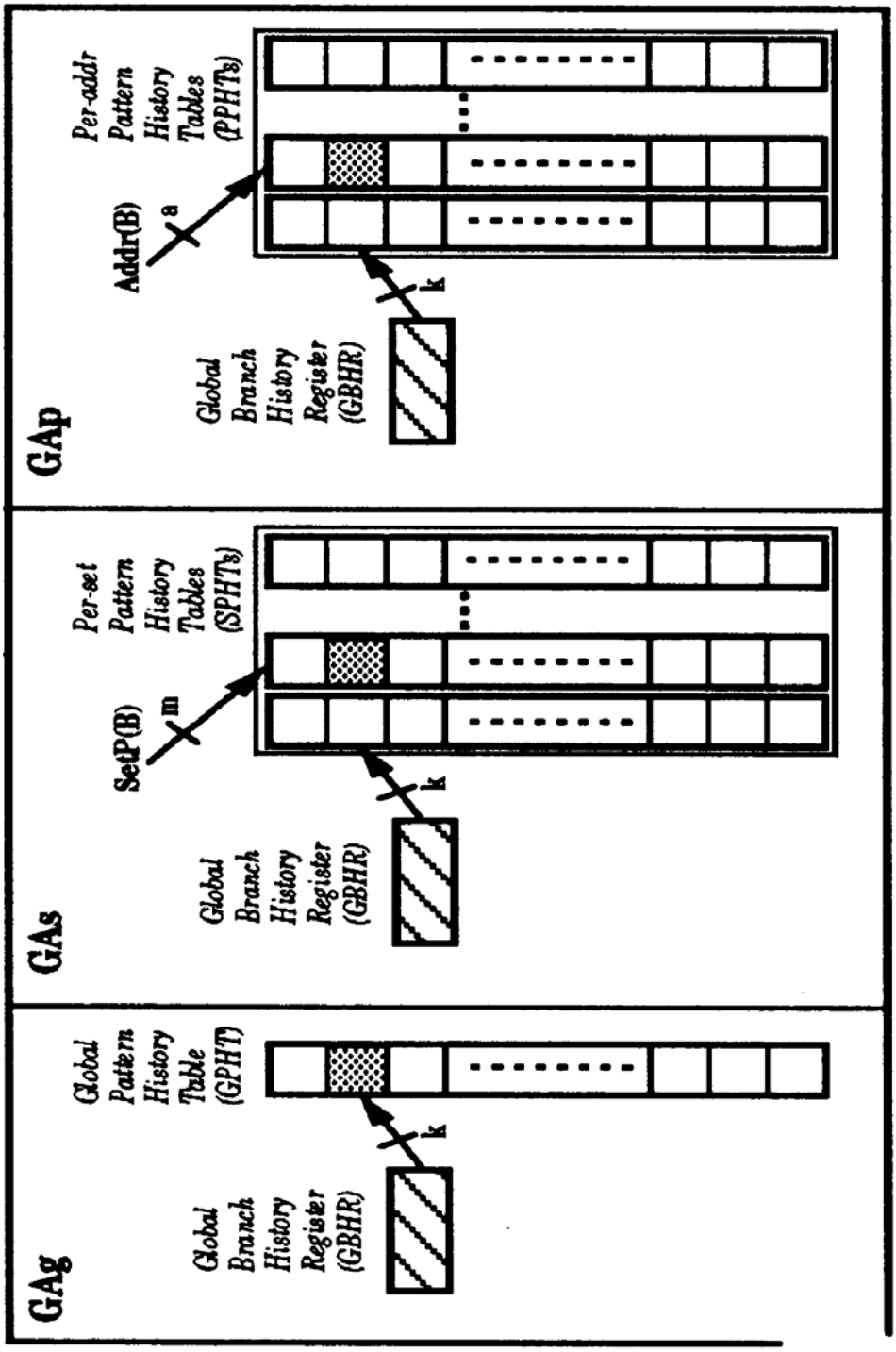


Figure 1.4: Global Adaptive Branch Prediction Methods

In the Global Adaptive Branch Prediction methods, the global history of the last k branches is recorded. Therefore, the history of

all branches influences each prediction, since each prediction uses the same history register [Yeh93].

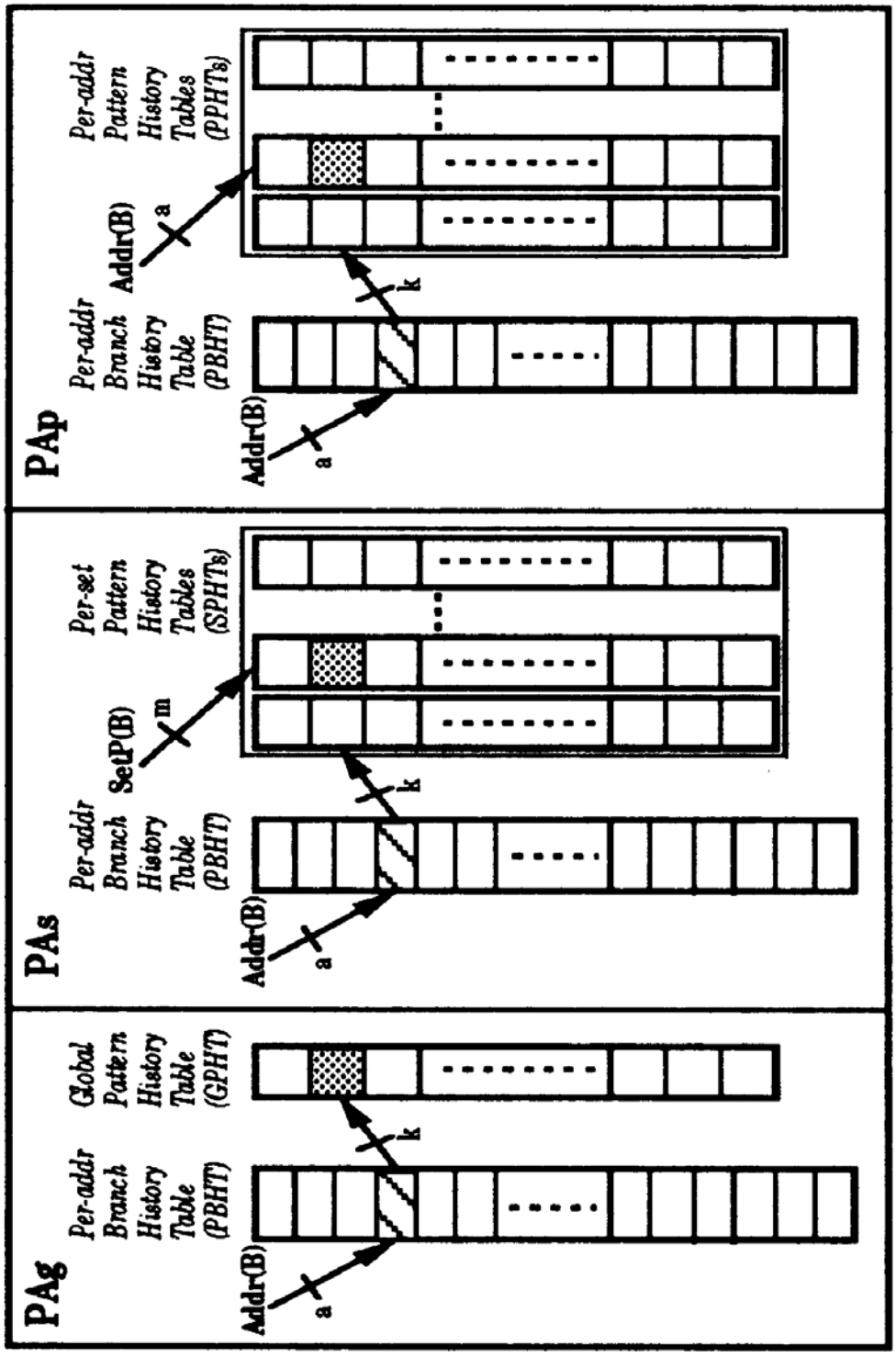


Figure 1.5: Per-Address Adaptive Branch Prediction Methods

In the Per-Address Adaptive Branch Prediction methods, the first-level branch history accesses the pattern history of the last k occurrences of the same branch address [Yeh93]. Each prediction is, therefore, only influenced by the history of its branch address, and not by other branches.

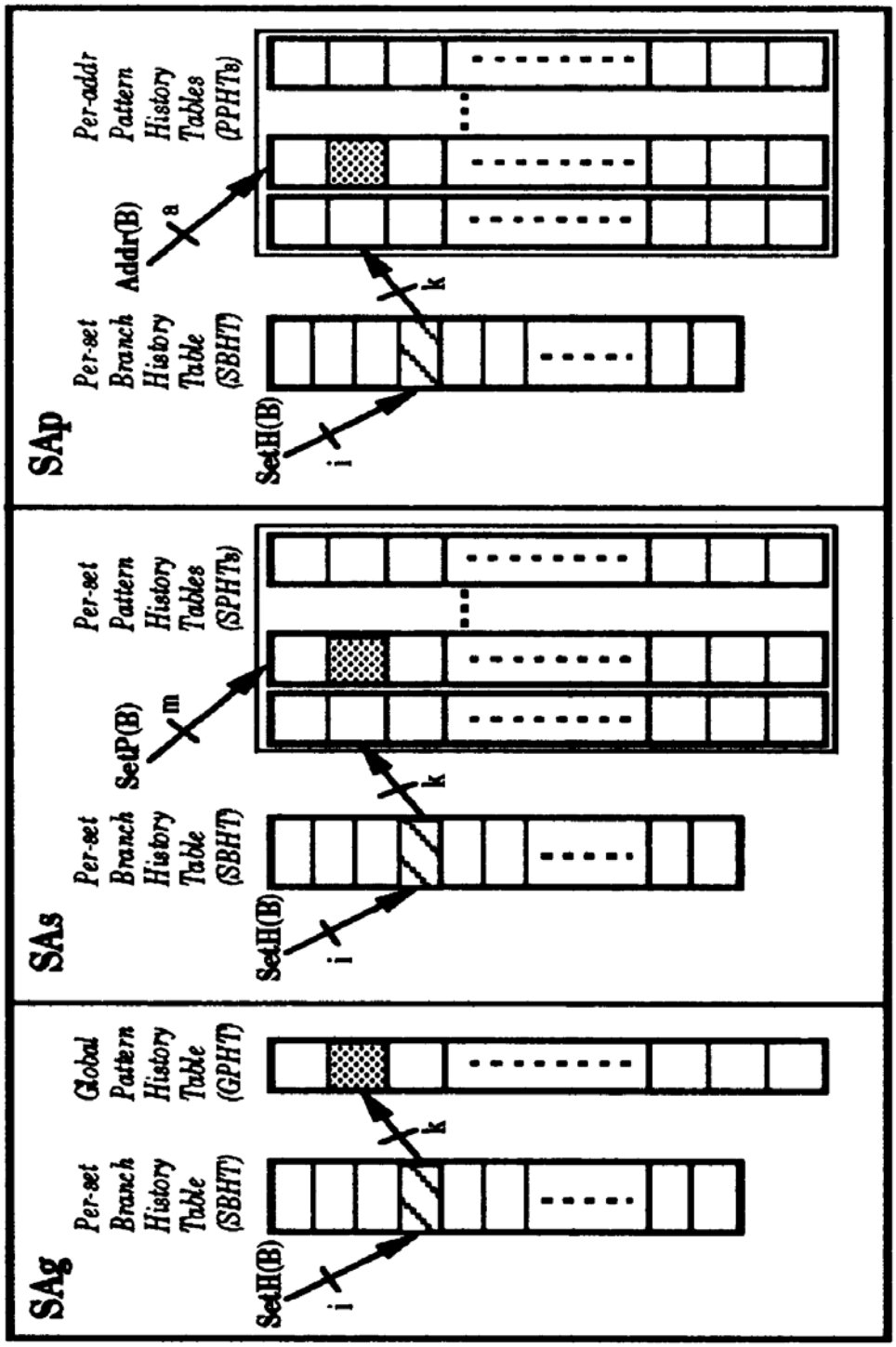


Figure 1.6: Per-Set Adaptive Branch Prediction Methods

For per-subset references, the set attribute can be any of the following: the branch opcode, the branch class (which is assigned by the compiler), or the branch address. In this way, the behavior of a branch effects other branches of the same type, or subset. The first-level of Per-set Adaptive Branch Prediction refers to the last k occurrences of a branch of the same set attribute. The second level can be accessed either globally, by a subset of a set, or by branch address [Yeh93].

1.2.3 The Branch-Target Buffer

To further improve prediction methods, the branch-target buffer (or cache) is introduced. In the BTB, the branch address and respective predicted address (or branch target) are stored. The branch address is used as a reference to the BTB to predict what the branch target should be. If the branch address is not in the table, then the branch is predicted "not taken". Using a BTB in combination with the Two-bit predictor increases prediction accuracy significantly [Driesen].

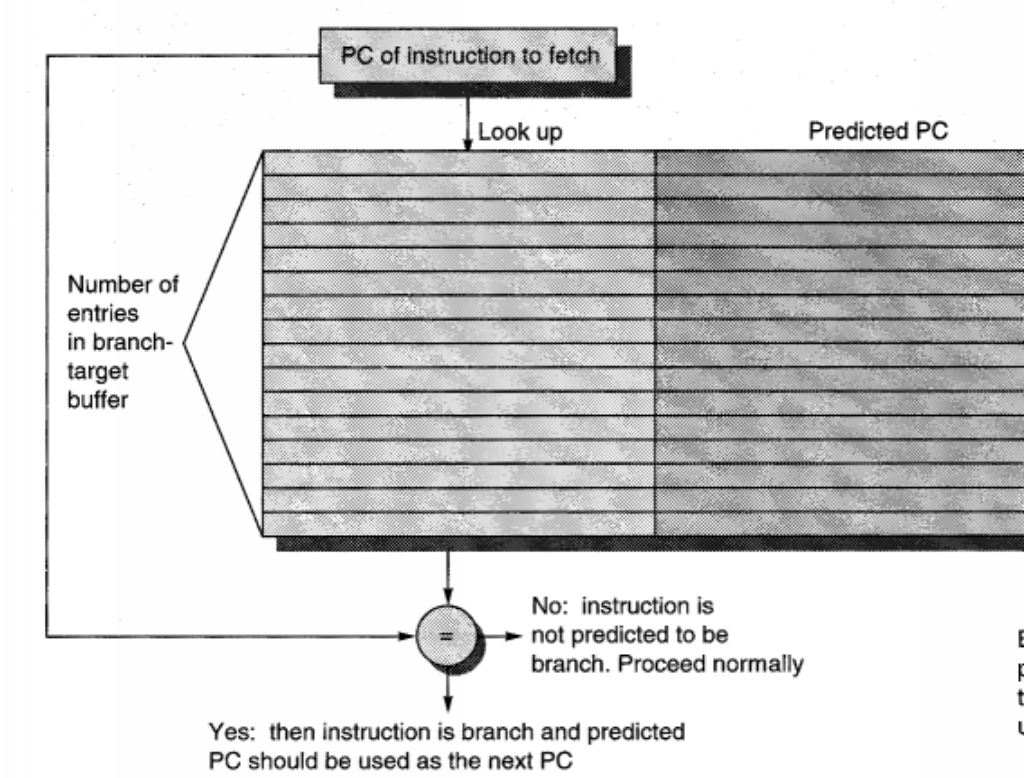


Figure 1.7: Branch Target Buffer

1.2.4 Hybrid or combinational predictors

Another solution when using more resources (transistors) on a microprocessor chip for branch prediction is to combine multiple

dynamic branch predictors [Driesen]. When using two predictors at the same time a special Branch Predictor Selection Table, or metapredictor, must be used to choose which predictor to use for a given situation. This metapredictor is similar to the two-bit predictor discussed above. It uses a two-bit counter to keep track of which predictor is more accurate for a specific branch address.

There are a couple of variations of this metapredictor. One variation uses run-time or compile-time information to choose which predictor outcome to use. Another variation is to use an n-bit confidence counter to keep track of the success rate of a predictor over the last 2n-1 prediction requests [Driesen].

1.3 Current Branch Prediction Methods

1.3.1 AMD K-6 2

AMD's K-6 2 microprocessor contains a 2-level branch predictor with an 8192 entry Branch History Table, a 16 entry BTC, and a 16 entry RAS.

1.3.2 Pentium III

The branch predictor methodology in Intel's Pentium III microprocessor is referred to as "deep" branch prediction. It contains a 512 entry Branch-Target Buffer.

1.3.3 Power PC 630

Motorola's Power PC 630 uses a Branch-Target Buffer and a two-bit predictor.

Chapter 2: Artificial Neural Networks

2.1 The Neuron

2.1.1 Biological Model

The basic model of the neuron is founded upon the functionality of a biological neuron. "Neurons are the basic signaling units of the nervous system" and "each neuron is a discrete cell whose several processes arise from its cell body" [Kandel].

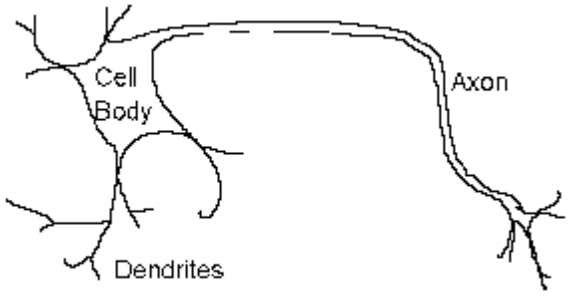


Figure 2.1: A Neuron

The neuron has four main regions to its structure. The cell body, or soma, has two offshoots from it, the dendrites, and the axon, which end in presynaptic terminals. The cell body is the heart of the cell, containing the nucleus and maintaining protein synthesis. A neuron may have many dendrites, which branch out in a treelike structure, and receive signals from other neurons. A neuron usually only has one axon which grows out from a part of the cell body called the axon hillock. The axon conducts electric signals generated at the axon hillock down its length. These electric signals are called action potentials. The other end of the axon may split into several branches, which end in a presynaptic terminal.

Action potentials are the electric signals that neurons use to convey information to the brain. All these signals are identical. Therefore, the brain determines what type of information is being received based on the path that the signal took. The brain analyzes the patterns of signals being sent and from that information it can interpret the type of information being received.

Myelin is the fatty tissue that surrounds and insulates the axon. Often short axons do not need this insulation. There are uninsulated parts of the axon. These areas are called Nodes of Ranvier. At these nodes, the signal traveling down the axon is regenerated. This ensures that the signal traveling down the axon travels fast and remains constant (i.e. very short propagation delay and no weakening of the signal).

The synapse is the area of contact between two neurons. The neurons do not actually physically touch. They are separated by the synaptic cleft, and electric signals are sent through chemical

interaction. The neuron sending the signal is called the presynaptic cell and the neuron receiving the signal is called the postsynaptic cell. The signals are generated by the membrane potential, which is based on the differences in concentration of sodium and potassium ions inside and outside the cell membrane.

Neurons can be classified by their number of processes (or appendages), or by their function. If they are classified by the number of processes, they fall into three categories. Unipolar neurons have a single process (dendrites and axon are located on the same stem), and are most common in invertebrates. In bipolar neurons, the dendrite and axon are the neuron's two separate processes. Bipolar neurons have a subclass called pseudo-bipolar neurons, which are used to send sensory information to the spinal cord. Finally, multipolar neurons are most common in mammals. Examples of these neurons are spinal motor neurons, pyramidal cells and Purkinje cells (in the cerebellum).

If classified by function, neurons again fall into three separate categories. The first group is sensory, or afferent, neurons, which provide information for perception and motor coordination. The second group provides information (or instructions) to muscles and glands and is therefore called motor neurons. The last group, interneuronal, contains all other neurons and has two subclasses. One group called relay or projection interneurons have long axons and connect different parts of the brain. The other group called local interneurons are only used in local circuits.

2.1.2 Mathematical Model

When creating a functional model of the biological neuron, there are three basic components of importance. First, the synapses of the neuron are modeled as weights. The strength of the connection between an input and a neuron is noted by the value of the weight. Negative weight values reflect inhibitory connections, while positive values designate excitatory connections [Haykin]. The next two components model the actual activity within the neuron cell. An adder sums up all the inputs modified by their respective weights. This activity is referred to as linear combination. Finally, an activation function controls the amplitude of the output of the neuron. An acceptable range of output is usually between 0 and 1, or -1 and 1.

Mathematically, this process is described in Figure 2.2 [Haykin].

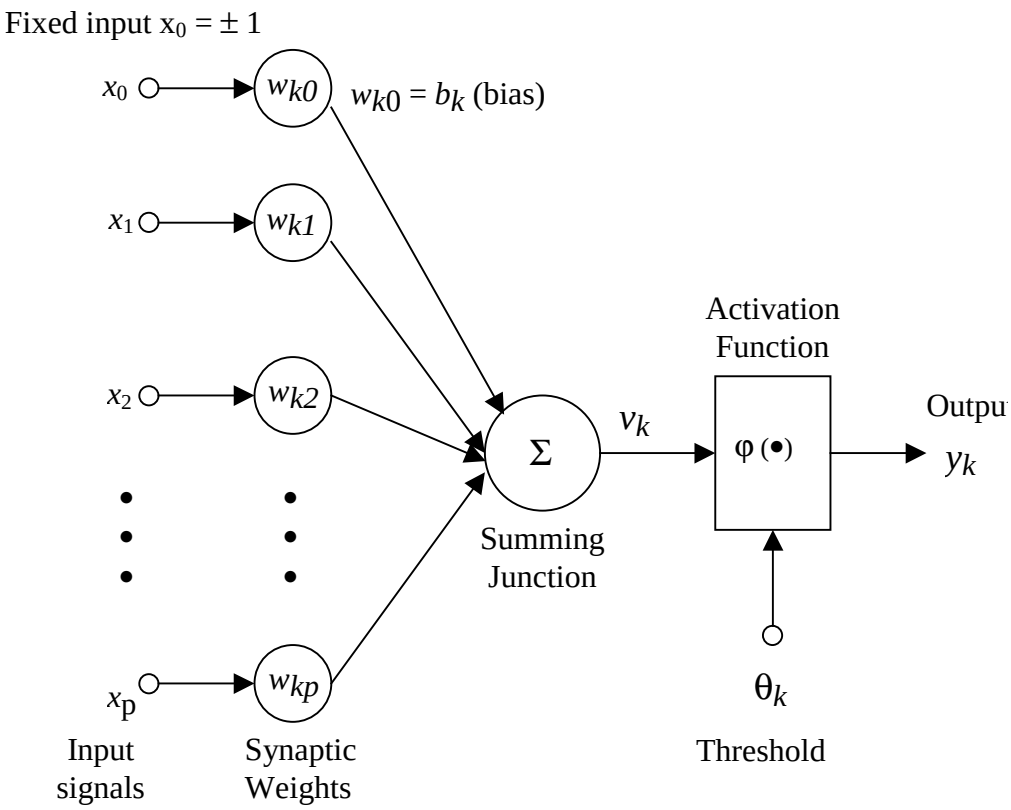


Figure 2.2: Model of a Neuron

From this model the interval activity of the neuron can be shown to be:

$$v_k = \sum_{j=1}^{p} w_{kj} x_j \qquad (2.1)$$

The output of the neuron, yk, would therefore be the outcome of some activation function on the value of vk.

2.1.3 Activation functions

As mentioned previously, the activation function acts as a squashing function, such that the output of a neuron is between certain values (usually 0 and 1, or -1 and 1). In general, there are three types of activation functions, denoted by φ(•) in Figure 2.2. First, there is the Threshold Function which takes on a value of 0 if the summed input is less than a certain threshold value (v), and the value 1 if the summed input is greater than or equal to the threshold value.

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases} \qquad (2.2)$$

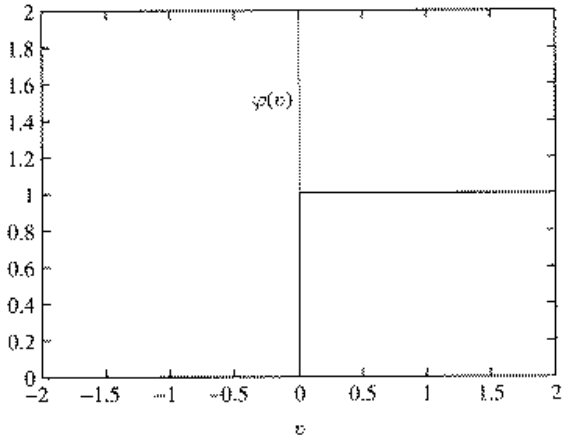
Secondly, there is the Piecewise-Linear function. This function again can take on the values of 0 or 1, but can also take on values between that depending on the amplification factor in a certain region of linear operation.

$$\varphi(v) = \begin{cases} 1 & v \geq \frac{1}{2} \\ v & -\frac{1}{2} > v > \frac{1}{2} \\ 0 & v \leq -\frac{1}{2} \end{cases} \quad (2.3)$$

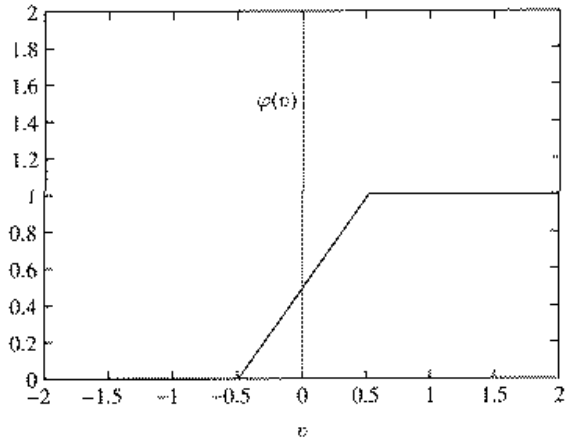
Thirdly, there is the sigmoid function. This function can range between 0 and 1, but it is also sometimes useful to use the -1 to 1 range. An example of the sigmoid function is the hyperbolic tangent function.

$$\varphi(v) = \tanh\left(\frac{v}{2}\right) = \frac{1-\exp(-v)}{1+\exp(-v)} \quad (2.4)$$

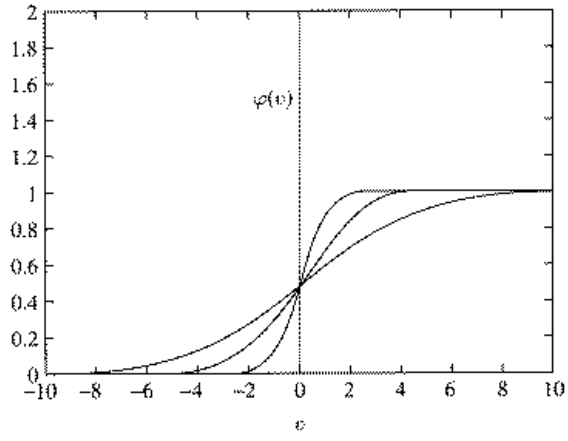
Figure 2.3 shows these activation functions plotted [Haykin].



theshold



piecewise-linear



sigmoid functions

Figure 2.3: Activation Functions

In Figure 2.2, two special values should be noted: the bias and the threshold. These are optional values that can be used,

determined by the specific situation for which the neural network will be used. The bias can be eliminated from the network by making the bias weight, wk0, equal to zero. The threshold value can also be set equal to zero to eliminate is effect on the network. For obvious reasons, if a Threshold Activation Function is not being used, the threshold value is not needed.

2.1.4 Directed Graphs

The neuron model described in Section 2.1.2 can be described in two other manners in order to simplify discussions. First, there is the signal-flow graph of a neuron, shown in Figure 2.4 [Haykin], where the symbols used are simplified.

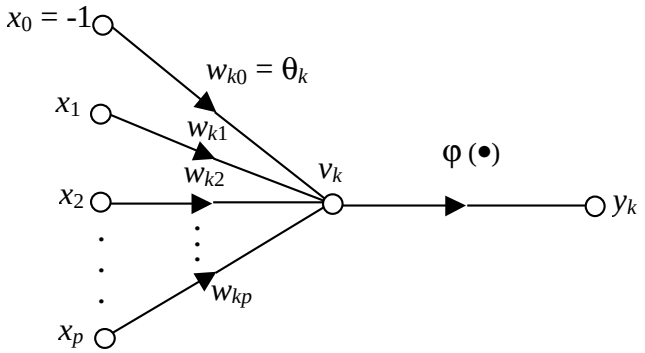


Figure 2.4: Signal-Flow Graph of a Neuron

An architectural graph can also be used to show the general layout of a neuron, as displayed in Figure 2.5 [Haykin]. This is useful when describing multilayer networks.

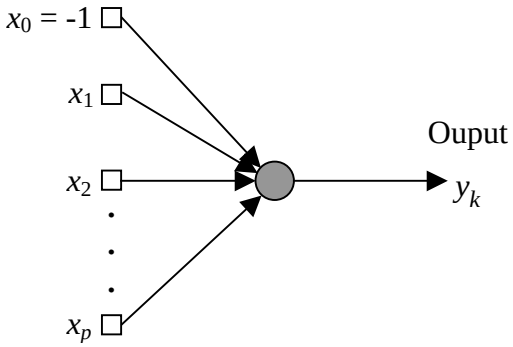


Figure 2.5: Architectural Graph of a Neuron

2.2 Learning

2.2.1 Description

One of the most important aspects of Artificial Neural Networks is the ability to learn and therefore update and improve

performance. The definition of this neural network learning according to Haykin is:
Learning is a process by which the free parameters of a neural network are adapted through a continuing process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place.

Haykin goes on to state three important events that occur during learning:
1. The neural network is stimulated by an environment.
2. The neural network undergoes changes as a result of this stimulation.
3. The neural network responds in a new way to the environment, because of the changes that have occurred in its internal structure.

Overall, the updating of the neural network is done by changing (or updating) the interconnected weights between neurons. Mathematically, this is described as follows:

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n) \quad (2.5)$$

How the update, or Δw, is calculated is determined by a set of rules applied with a certain paradigm. A general overview of the Artificial Neural Network learning process is shown in Figure 2.6 [Haykin].

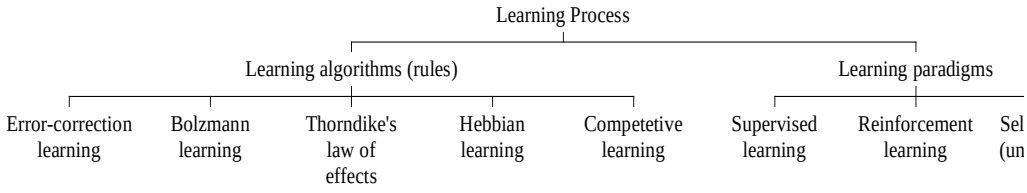


Figure 2.6: The Taxonomy of Learning

Learning algorithms are a set of rules by which the network is updated. The Learning paradigms are the styles or methodology in which the learning algorithm is performed. There are many different styles of learning, and new algorithms are continuously being explored and developed. For this thesis, Hebbian learning will be discussed.

2.2.2 Hebbian Learning

Hebbian learning is one of the oldest and most popular learning methods today [Haykin]. It derives its methodology from the idea that the more often a synapse between two neurons is fired, the

stronger the connection between the two neurons becomes. Expanding that statement, it can be said that when the connection between two neurons is excited simultaneously (or synchronously), the connection is strengthened, and when the connection is excited asynchronously, the connection is weakened [Haykin].

Mathematically, the change of weight in Hebbian learning is a function of both presynaptic (input) and postsynaptic (output) activities. In defining the weight update, a parameter called the learning rate is introduced. This learning rate, a positive value, defined the speed at which the weight changes. In order to keep the weight from growing too fast (which may lead to saturation), another parameter called the forgetting factor is introduced (this is also a positive value). Finally, a formula for the weight update for Hebbian learning can be defined as:

$$\Delta w_{kj}(n) = \eta y_k(n) x_j(n) - \alpha y_k(n) w_{kj}(n) \quad (2.6)$$

2.3 The Perceptron and Multilayer Perceptrons

The perceptron is the simplest form of a neural network used for the classification of linearly separable patterns [Haykin]. Its structure can be easily shown using the signal-flow graph described above, since a single layer perceptron operates as a single neuron. Figure 2.7 shows this [Haykin].

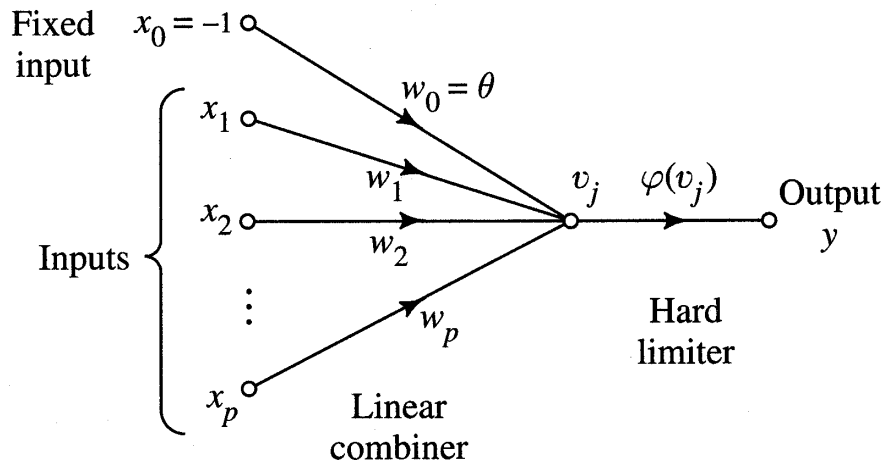


Figure 2.7: A Single Layer Perceptron

Multilayer perceptrons are an expansion of the perceptron idea, and can be used to solve much more difficult problems. They consist of an input layer, one or more hidden layers and an output layer. The hidden layers

give the network its power and allow for it to extract extra features from the input. Figure 2.8 shows a multilayer perceptron with two hidden layers [Haykin].

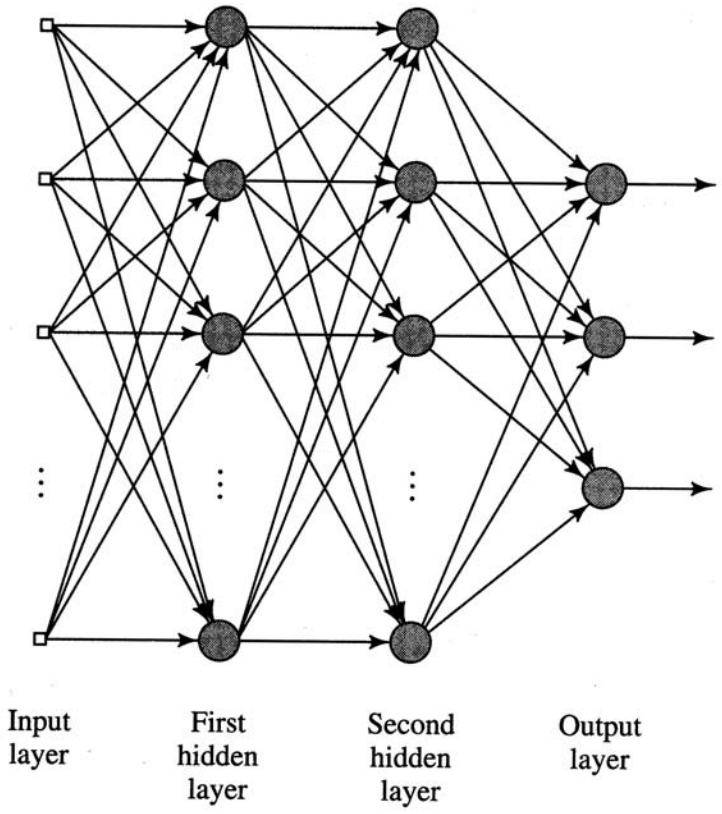


Figure 2.8: Multilayer Perceptron

One of the most popular methods used in training a multilayer perceptron is the error back-propagation method, which includes two passes through each layer of the network: a forward pass and a backward pass [Haykin].

2.4 Feedforward and Backpropagation

In error back-propagation, there are two types of signal flow. First, signals are calculated in a forward manner, from left to right, through each layer. These are called function signals. Second, the error at each neuron is calculated, from right to left, in a backward manner through each layer. These are called error signals.

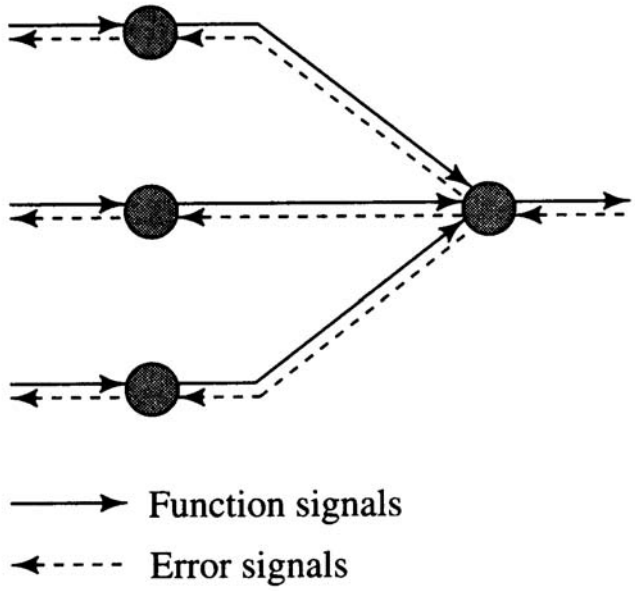


Figure 2.9: Signals in a Multilayer Perceptron

The goal of the error back-propagation method is to make the average squared error of the entire network as small as possible. Once the function signals have been calculated, the error back be back-propagated though the network. The error is calculated using the local gradient, δj(n)[Haykin]. At the error of the output layer, the local gradient is just defined as:

$$\delta_j(n) = e_j(n)\varphi_j'(v_j(n)) \tag{2.7}$$

When calculating the error for a neuron in the hidden layer, the local gradient is defined as:

$$\delta_j(n) = \varphi_j'(v_j(n))\sum_k \delta_k(n) w_{kj}(n) \tag{2.8}$$

The weight correction, Δwji(n), is defined as the learning rate parameter (η) times the local gradient (δ) times the input signal of the neuron (j):

$$\Delta w_{ji}(n) = \eta \cdot \delta_j(n) \cdot y_i(n) \tag{2.9}$$

A momentum term is also used in calculating the weight update.

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n) \tag{2.10}$$

Chapter 3: SimpleScalar

3.1 SimpleScalar

3.1.1 Software Architecture

SimpleScalar has a modular layout. This layout allows for great versatility. Components can be added or modified easily. Figure 3.1 shows the software structure for SimpleScalar. Most of the performance core are optional modules [Austin].

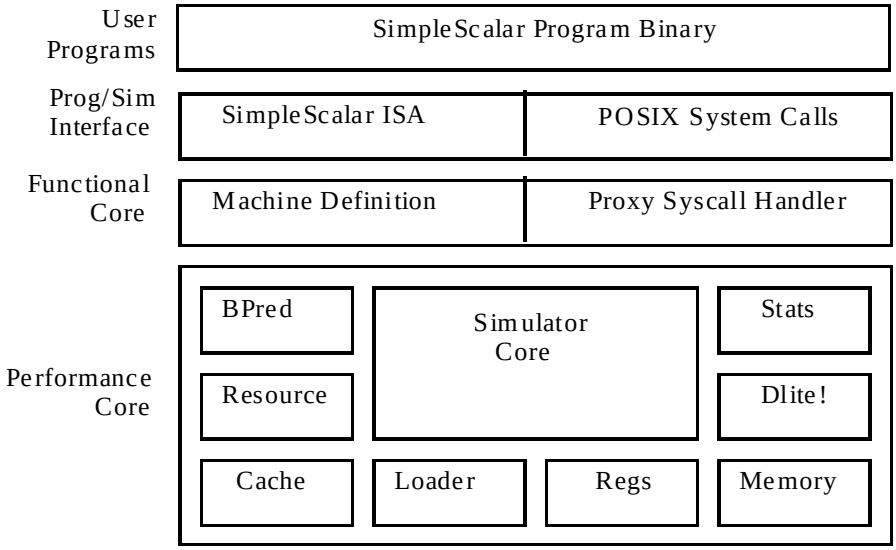


Figure 3.1: SimpleScalar Software Architecture

3.1.2 Hardware Architecture

The Hardware architecture of the SimpleScalar simulator closely follows the Post-RISC architecture previously described. Figure 3.2 shows the out-of-order pipeline for the SimpleScalar hardware architecture.

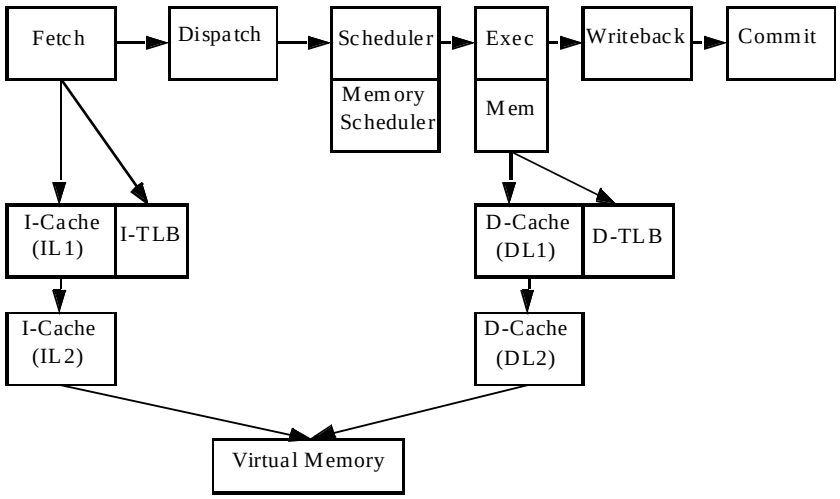


Figure 3.2: Out-of-Order Issue Architecture

3.1.3 Instruction Set Architecture

The instruction set architecture is based on MIPS and DLX [Patterson] instruction set architectures, with some additional addressing modes [Austin]. It is a "big-endian" instruction set architecture definition, which allows for easier porting of the simulator to new hosts since the simulator is compiled to match the host's endian. The instructions are 64-bit. Only 48 bits are currently used; the extra 16 bits allow for expansion or research of instructions.

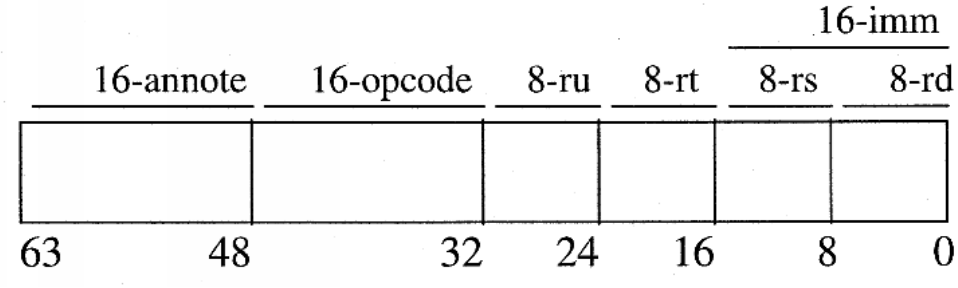


Figure 3.3: Instruction Format

3.1.4 Running SimpleScalar

For this thesis, the Out-of-Order execution simulator will be used. The correct command-line format for using the out-of-order simulator is as follows: sim-outorder <options> <sim_binary>. The <options> include the branch prediction configuration.

3.1.5 Branch Prediction

SimpleScalar supports a variety of branch prediction methods. It allows for the static always “taken” and always “not taken” prediction methods. It also has options for the Two-Level Adaptive Branch Predictors. It supports a Two-bit Predictor referred to as a bimodal predictor. SimpleScalar even supports a Hybrid predictor, a combination of the two-level and bimodal predictors, using a metapredictor to choose between the two predictors. SimpleScalar also utilizes a branch address predictor through Branch Target Buffer.

The command line option for specifying the branch predictor is -bpred <type>. The types are as described in Table 3.1.

Table 3.1: Branch Prediction Types

predictor option	Description
nottaken	always predict not taken
taken	always predict taken
perfect	perfect predictor
bimod	bimodal predictor: BTB with 2-bit counters
2lev	two-level adaptive predictor
comb	hybrid predictor: combination of 2level and bimodal

Figure 3.4 shows the Two-level adaptive branch predictor layout.

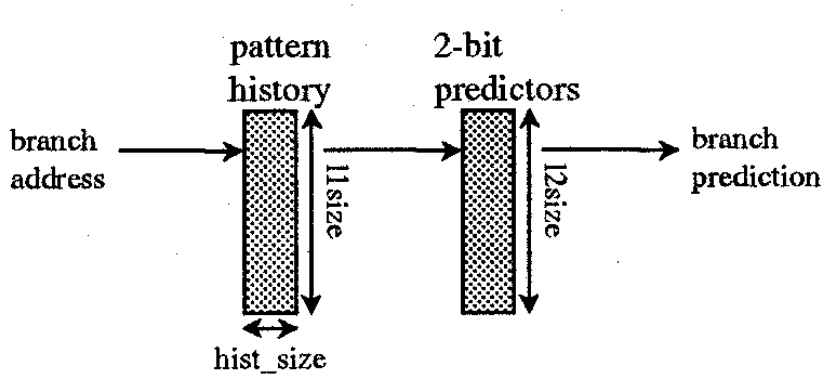


Figure 3.4: Two-level Predictor Layout

The configuration options for each predictor are described in Table 3.2.

Table 3.2: Branch Prediction Options

configuration parameters	Description
-bpred:bimod <size>	the size of the direct mapped BTB
-bpred:2lev <l1size> <l2size> <hist_size>	l1size - the size of the first level table (number of shift registers) l2size - the size of the second level table (number of

	counters) hist_size - the history pattern width (width of shift registers)
-bpred:comb <size>	The number of entries in the metapredictor table

Table 3.3shows how specific 2-level design options may be implemented in the SimpleScalar simulator. Note that each counter has one BTB entry.

Table 3.3: Two-Level Branch Prediction

type	l1size	l2size	hist_size
counter based	1	m	0
GAg	1	2^w	w
GAp	1	m(m>2^w)	w
PAg	n	2^w	w
PAp	n	m(m == 2^(N+W))	w

When selecting the hybrid predictor, both the bimod and 2lev configuration options may be specified.

3.2 Using the SPEC95 benchmarks

Available with the SimpleScalar simulator are SimpleScalar binaries of the SPEC95 benchmark suite. These binaries are useless without the input files available only by purchasing the license to use the benchmarks. In conjunction with a PERL script, Run.pl, written by Dirk Grunwald and Artur Klauser, the SPEC95 benchmarks can be run on the SimpleScalar simulator. This thesis uses the following SPECint95 benchmarks to test the branch predictors: 099.go, 126.gcc, 130.li, 134.perl; and the following SPECfp95 benchmarks: 101.tomcatv, 107.mgrid, 145.fpppp. The correct command-line implementation used by Run.pl for running the SPEC95 benchmarks on the SimpleScalar architecture is as follows: sim-outorder <sim_options> <benchmark_binary> <benchmark_options> < <input_file> > <output_file>.

Chapter 4: Methodology

4.1 Programming the Neural Network

To more easily integrate the ANN code into SimpleScalar, the neural network was programmed in C. Also, the error subroutines used in the SimpleScalar package were used in programming the neural network (specifically the fatal subroutine for run-time error messages, which are included in the misc.h and misc.c files).

The two files that define the feed-forward back-propagation artificial neural network code are a header file and a C code file, ffbpann.h and ffbpann.c. Three structures were created to define the overall design of the FFBPANN: bounds, layer and ffbpann. Bounds is a simple template of two integers that define the minimum and maximum number that will be represented as input to the FFBPANN. These values were used to normalize the inputs. Layer is a more complicated structure that is used in defining the overall FFBPANN structure. A layer contains a variable of the number of nodes in the layer, an array for the error calculations of each node, an array for the output calculations of each node, a multi-dimensional array for the weights, a multi-dimensional array for saveweights (used during training of the FFBPANN), and a multi-dimensional array of deltas (used during back-propagation). The FFBPANN structure itself possesses four variables: the number of layers in the FFBPANN, the momentum, learning rate and sigmoid function gain (all used for training of the FFBPANN), and the net error calculated for the ANN. It also has a pointer to the array of input bounds, a pointer to the input layer, a pointer to the output layer, and a pointer to an array of layers (described above).

Several functions were defined so that the FFBPANN would behave properly. Create_ffbpann allocates memory for the entire FFBPANN, and ffbpann_free frees up the memory allocated for the FFBPANN. Init_ffbpann initializes the FFBPANN weights either to random numbers, or predefined weights (pre-trained values saved in a text file). Nnet_random is used to calculate the random values used when initializing a weight matrix. Calc_ffbpann calculates the output of the FFBPANN given the current weight matrices. Backprop performs the back-propagation algorithm of the FFBPANN, and is usually followed by the updateweights function, which calculates the new weight values, based on the recent back-propagation. Restore_weights and save_weights are used during training to either revert to the last saved weight matrices or save the currently calculated matrices, depending on how successful the current epoch of training was. Dump_weights is the function used to write the current weight matrix to a file. The format of this file is as follows: the bounds for each input node are at the beginning of the file (one pair per line); each subsequent line contains a weight value starting with the first node of the first layer, then the second node of the first layer, and continuing until the last node of the last (output) layer. Train_ffbpann and

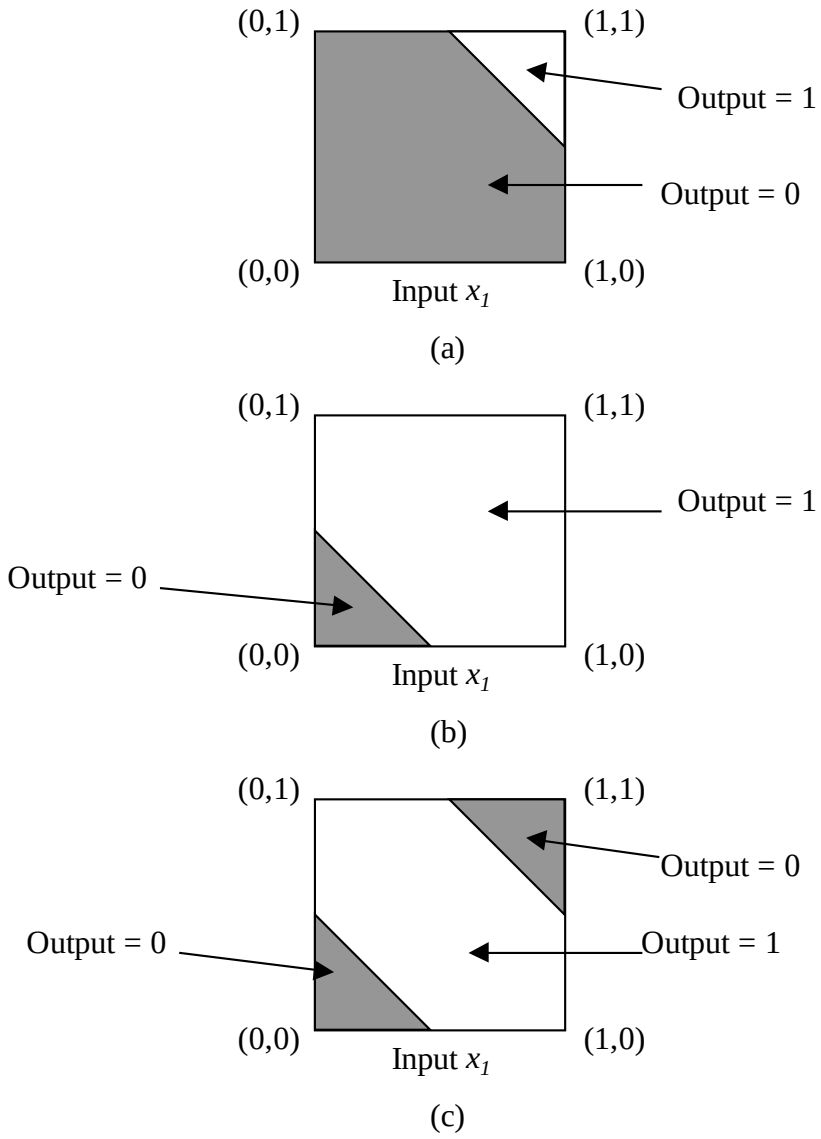
learn_ffbpann are short sample functions for setting up training and testing routines.

To accommodate testing and training, small C programs were used as "wrappers" to the four main files used for the FFBPANN. The layer and node parameters are defined in the "wrapper" program so that various FFBPANN configurations could be tested. Examples of these programs follow in Appendix B, and are discussed below.

4.2 Verifying the FFBPANN Code

4.2.1 The XOR Problem

To verify that the FFBPANN code was working properly the standard XOR (exclusive OR) test was used. The XOR test is a very specific application of a common problem of a single-layer perceptron. A single-layer perceptron is incapable of classifying input patterns that are not linearly separable, as is the case in the XOR problem. To overcome this limitation, a multi-layer neural network is used to classify the XOR inputs as shown in Figure 4.1 and Figure 4.2 [Haykin].



(a) Decision boundary constructed by hidden neuron 1 of the network in Figu
(b) Decision boundary constructed by hidden neuron 2 of the network.
(c) Decision boundaries constructed by the complete network.

Figure 4.1: XOR Classifications

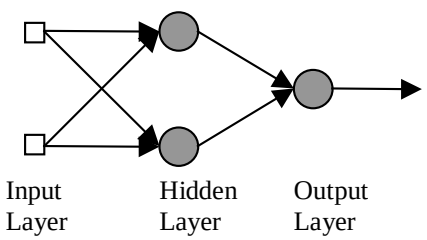


Figure 4.2: Multi-layer network for solving the XOR Problem

A 2x1 FFBPANN was trained using the very small data set of the XOR function. The weights were saved in a file and another short program was used to verify that the saved weights were correct to solve the XOR problem. The training and test files follow in Appendix B. The output from the testing file is as follows:

inputs: 0.000000 0.000000 output: 0.000068
inputs: 0.000000 1.000000 output: 0.999886
inputs: 1.000000 0.000000 output: 0.999923
inputs: 1.000000 1.000000 output: 0.000093

4.2.2 Predicting Sunspots

A second test was used to verify that the FFBPANN used for this experiment could indeed do prediction. A data set of sunspot levels was discovered [Kutza] and was used to train and test a 10x1 FFBPANN. The inputs to the FFPBANN were the sunspot levels from the previous 10 years, and the output is the prediction of the sunspot level for the current year. The code used follows in Appendix B. The output of the test program follows in Table 4.1.

Table 4.1: Sunspots Test Program Output

Year	Sunspots	Prediction
1960	0.587	0.547
1961	0.282	0.306
1962	0.196	0.048
1963	0.146	0.082
1964	0.053	0.183
1965	0.079	0.153
1966	0.246	0.174
1967	0.491	0.454
1968	0.554	0.620
1969	0.552	0.533
1970	0.546	0.431
1971	0.348	0.473
1972	0.360	0.157
1973	0.199	0.209
1974	0.180	0.121
1975	0.081	0.099
1976	0.066	0.047
1977	0.143	0.076
1978	0.484	0.313

1979	0.813	0.615

4.3 Adding a new predictor to SimpleScalar

To add a new predictor to SimpleScalar the following files in the package had to be modified: bpred.c, bpred.h, sim-outorder.c, sim-bpred.c and the Makefile. The Makefile was modified so that the extra FFBPANN code was included during compilation. The modifications to sim-outorder.c and sim-bpred.c accomplished three tasks. Firstly, so that additional information (variables) would be sent to the branch predictor functions specifically for use by the FFBPANN predictor; secondly, to define the FFBPANN predictor command line options and how the predictor is called; and lastly, to display the usage information for the command line options. The majority of modifications were made to the bpred.h and bpred.c files. The predictor structure was defined in bpred.h, and the predictor functions were coded in detail in bpred.c. For more detailed information, see Appendix C for the code listing.

The procedure for adding the new predictor was to add another hybrid predictor to the code so that there existed two hybrid predictors, but the second one possessed a different name (the FFPBANN predictor). Then, slowly, code for the 2-level component of the new hybrid predictor was modified to accommodate the FFPBANN predictor code. See the code listing in Appendix C for more detailed information, and Section 4.4.1 for a discussion of the predictor design.

4.4 The new predictor

4.4.1 Design

The overall structure of the new predictor (the FFBPANN predictor) is similar to the Combination (or Hybrid) predictor. In the FFBPANN predictor, a two-bit predictor and a purely artificial neural network predictor are used in combination, with a meta predictor used to choose between the outputs of the two predictors.

The purely artificial neural network predictor is the feed-forward back-propagating artificial neural network described in Section 4.1. There are four inputs to the network and one output. The inputs are the following variables in the SimpleScalar program: the branch address, the opcode portion of the instruction, and the RS and RT registers. The single output is the branch target (“taken” or “not taken”). The number of neurons in each layer can be configured at runtime, and there may be from 1 to 4 hidden layers (the input and output layers are, of course, required). The only prerequisite for selecting a certain configuration at runtime is that a weights file must exist for the ANN to function. The format for the file name of the weights file is comprised of the layer configuration followed by the string “.weights.dump”. For example, a 4x3x2x1 FFBPANN would have a weights matrix file named 4x3x2x1.weights.dump. This weights file can be a set of

randomly generated numbers or a set of trained data. It is recommended that a set of trained data be used. This file can be created using the dump_weights function. The FFBPANN will adjust, or update, its weights as SimpleScalar runs, and therefore improve its predictions. Figure 4.3 shows a simple network diagram of the ANN used in this scenario.

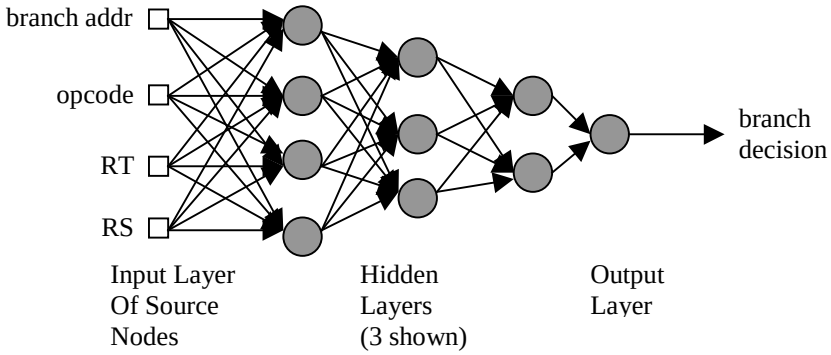


Figure 4.3: FFBPANN Structure

4.4.2 Training the FFBAPNN

To obtain a data set to train the FFBPANN for branch prediction, the SimpleScalar code was modified to output branch information to a file while it ran a simulation. The branch information consists of data available to the branch predictor function that could be used as input for the FFBPANN (which was the aforementioned branch address, the opcode portion of the instruction, and the RS and RT registers), and the correct prediction.

Wrapper programs, discussed in Section 4.1, were written to train and test a weight matrix using this branch information. The training data was divided into three portions for training and testing. The first 60% of the data was used for training, the next 30% was used for testing the progress of the training routines (used to calculate the stopping criteria for training), and the remaining 10% was reserved exclusively for testing a trained FFBPANN.

Several different techniques were used for training. This was done to find the quickest and most efficient way to train the neural network. This was also done because, early on, the training was not producing a quality branch predictor. Therefore, modifications to the training data set, varying the training parameters, and modifying the ANN structure were tried in the hopes of improving the training procedure. The following procedures were taken to vary the attempts of finding a weight matrix that would classify the inputs.

- The whole of the raw training data set was used.

- Modifications to the raw data set were made to reduce the size of the data set. The most common entries in the data set – lines that appeared more than 10 times – were extracted. This was attempted again for lines that appeared only more that 50 times. This left only single entries in the resulting data set for the most common entries.

- Another attempt to reduce the data set was to pull the same frequently occurring entries, described above, out of the raw data set; however, this time the total number of occurrences was reduced by a fraction instead of being reduced to one entry. Now an entry occurring 10000 times in the raw data set would only occur 10 times in the reduced data set.
- The momentum (alpha) and the learning rate (eta) parameters of the FFBPANN were varied. These values changed only slightly, as the values initially chosen were suitable for most cases.
- Different layer structures were tested. The following four FFBPANN configurations were tried: 4x3x2x1, 4x4x3x1, 4x4x2x1, 4x3x3x1. Though, the 4x3x2x1 structure was the one most commonly used for training and testing.

Chapter 5: Results and Discussion

5.1 Training

The training data was gathered, as described in the previous chapter, by running a version of SimpleScalar modified to output the branch information (inputs and output) to a file. Several experiments to train a FFBPANN branch predictor were done using this data by varying the training slightly with each training attempt. None of the training techniques used produced a neural network branch predictor that performed more effectively than the other branch predictors already used in the SimpleScalar simulator. The results of these training attempts are as follows.

The modifications to the raw data sets did not produce better results than the attempts on the raw data sets described below. The attempt to reduce the data set by extracting only unique lines from the data would not help since frequently occurring data (i.e. a pattern for the ANN to pick out) would not be represented. The other reductions of the data set just pulled out infrequently occurring lines, but did not help in the training attempts either. The reason for reducing the size of the data sets was to speed up training so that the following two modification to the training procedures could be tried numerous times.

The attempts to vary the momentum (alpha) and the learning rate (eta) of the FFBPANN also did not produce satisfactory results. Again, these values only varied slightly, since the standard momentum and learning rate were used. Changing these parameters should help speed up the training procedure (or give it a little boost). However, since the training was performing poorly, all that these modifications did was assist the training perform poorly faster. However, changing these values had little or no effect on the outcome, and at worst would cause the training to finish quicker, but with poorly calculated weights. The testing of these weights never produced satisfactory predictions.

Surprisingly, changing the layer structure of the FFBPANN had no effect on the training of the FFBPANN. With the reduced data sets, multiple layer changes could be tested more frequently. Since training with the full raw data sets would take so long, the only way to test multiple later options was to use the reduced data sets. None of the four FFBPANN architectures tested (4x3x2x1, 4x4x3x1, 4x4x2x1, 4x3x3x1) produced improved training performance or more accurate prediction results.

5.2 Discussion

One oddity to note occurred while gathering the training data. The SimpleScalar code was modified to write the selected inputs for the FFBPANN to a file. However, the branch data for every branch was not captured due to a file size limitation of Solaris 2.6. The maximum file size turned out to be about 2 Gigabytes (231 bytes). When this maximum was reached, subsequent branch prediction information was no longer

gathered. The gcc SPEC95 benchmark was the only benchmark for which the full branch prediction data was gathered. It is significant to note the importance of gathering the gcc branch information, since gcc performance is used by the industry to gauge its own improvements to such components as compilers. The limitation on the amount of the branch information that could be gathered could be a contributing factor for the poor FFBPANN training. Table 5.1 shows the amount of branch prediction information that could be gathered.

Table 5.1: Limitations of Branches Recorded

	fpppp	gcc	fpppp and gcc	mgrid	go
total branches	2774271287	50378431	2824649718	1444135667	2423767703
precent of total branches recorded	4.63%	100.00%	6.33%	9.28%	5.38%
total branches recorded	128405869	50378431	178784300	134019707	130442404
unique entries	127421801	49530881	176952682	16040941	128871783
precent of unique entries	99.23%	98.32%	98.98%	11.97%	98.80%
unique inputs	126942800	49298484	176241284	13146619	128249968

Another possible reason for non-convergence of the FFBPANN weights is that the training data is too chaotic. An ANN is well suited to pick out patterns from the input. In this situation, no unique patterns exist for the FFBPANN to effectively classify. Even though the FFBPANN is much more suited to this task than the human eye given the amount of inputs examined, using the four inputs available to the predictor the FFBPANN was still unable to recognize any consistency in the data. Table 5.1 shows that almost every branch examined for the gcc, fpppp, and go benchmarks had a unique pattern – all four inputs and the branch outcome were different. However, this does not necessarily prove that the data is too chaotic for the FFBPANN weights to converge, since when examining the case of the mgrid benchmark, only about 12% of the branches were unique. However, during training with the full raw data set from the mgrid benchmark, something very interesting was noticed.

A closer look at the training progress using the mgrid data set shows that the FFBPANN was behaving almost exclusively as a “branch-always-taken” predictor. This is shown in Figure 5.1. While the output never fully reached the value of “1,” it was extremely close at a maximum value of 0.969877.

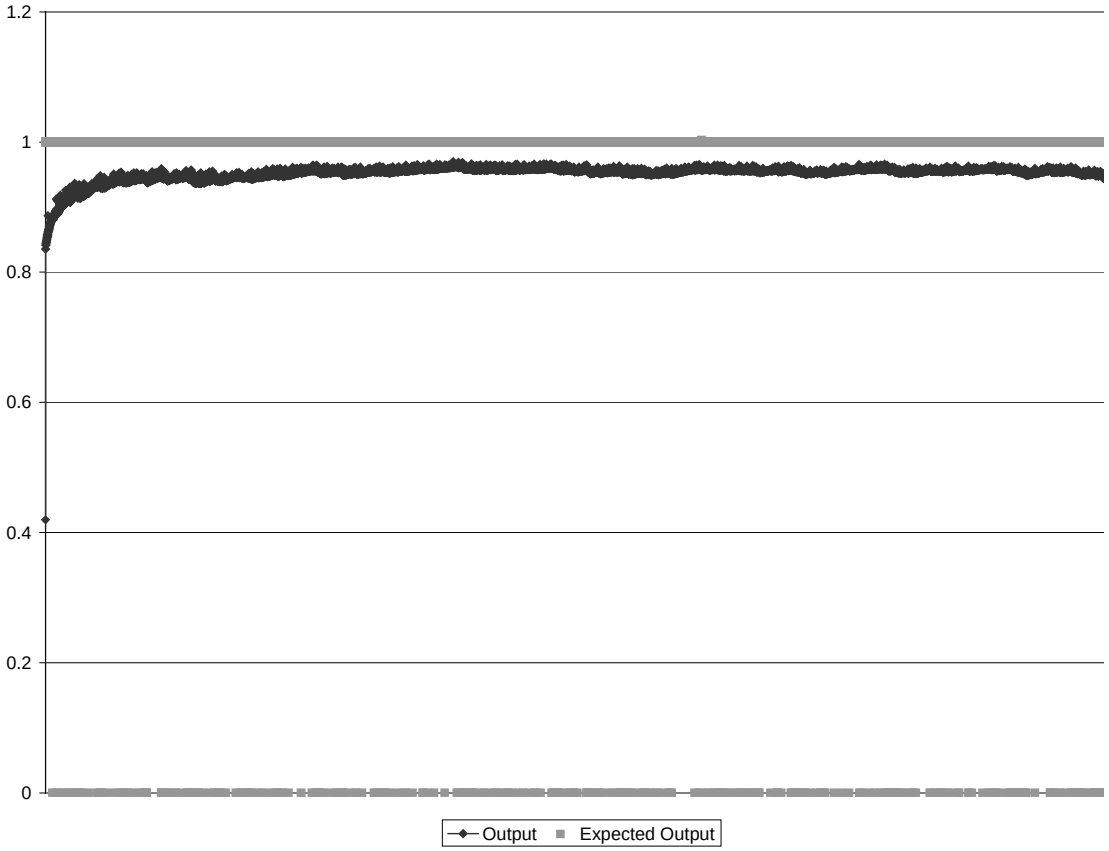


Figure 5.1: Mgrid Training Progress

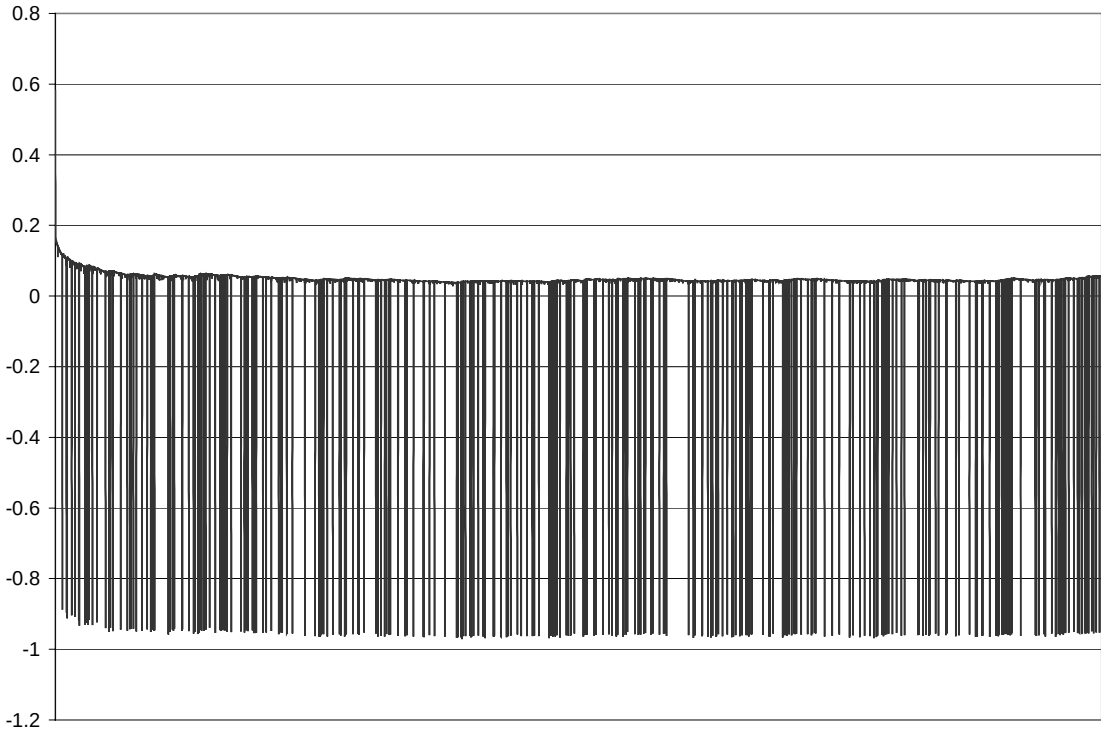


Figure 5.2: Mgrid Training Error

Another item to note about the mgrid training data is that the majority of the branches were “taken”. The training error shown in Figure 5.2 more clearly shows how often the branch should have been “not taken”. About 95% (127569741 of 134019707 recorded branches) of the mgrid branch decisions were “taken”. This could definitely account for why the FFBPANN appeared to be acting as a “branch-always-taken” predictor. Taking a closer look at the training output shows how quickly the ANN starts behaving this way. Figure 5.3 zooms in on the beginning of the mgrid training progress.

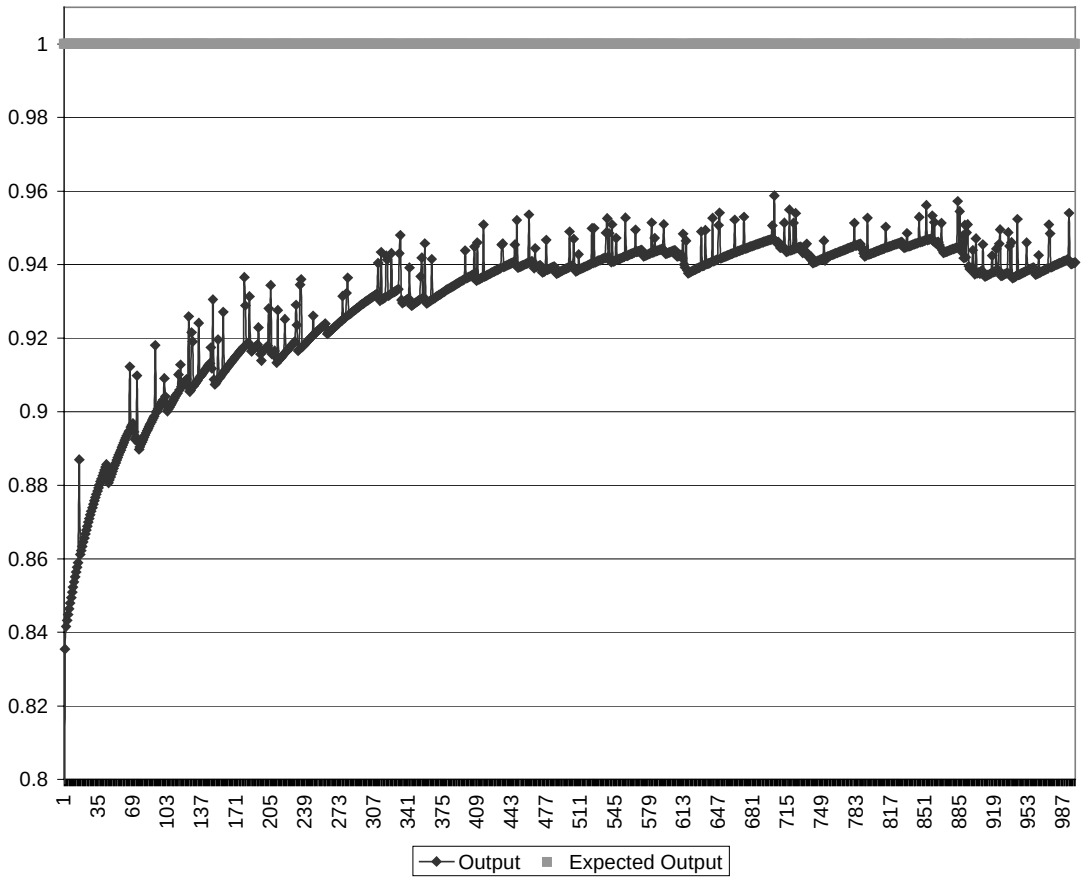


Figure 5.3: Mgrid Training Progress Zoom

Table 5.2 examines how frequently the other SPEC95 benchmarks' branches were "taken" or "not taken".

Table 5.2: Branch Data

Branches	Gcc	fpppp	mgrid	go
"taken"	30558126	77594080	127569741	84216932
"not taken"	19820305	50662550	6449966	46225472
Total	50378431	128256630	134019707	130442404
Percent "taken"	60.66%	60.50%	95.19%	64.56%

Interestingly enough, while training using the gcc and go benchmark data, which have a more even distribution of "taken" and "not taken" branches (about 61% and 65%, respectively), the same training error occurred. The ANNs produced by those training attempts also performed very closely to a "branch-always-taken" predictor. Figure 5.4 and Figure

5.5 show the training progress using the gcc and go branch prediction data, respectively.

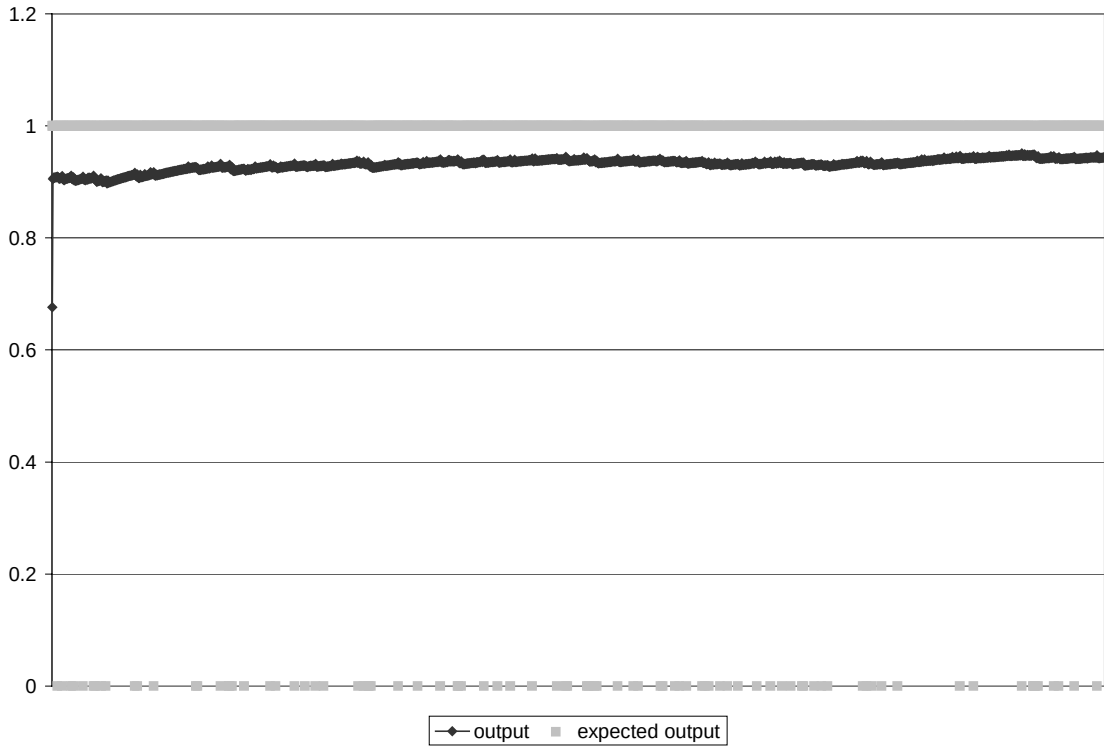


Figure 5.4: Gcc Training Progress

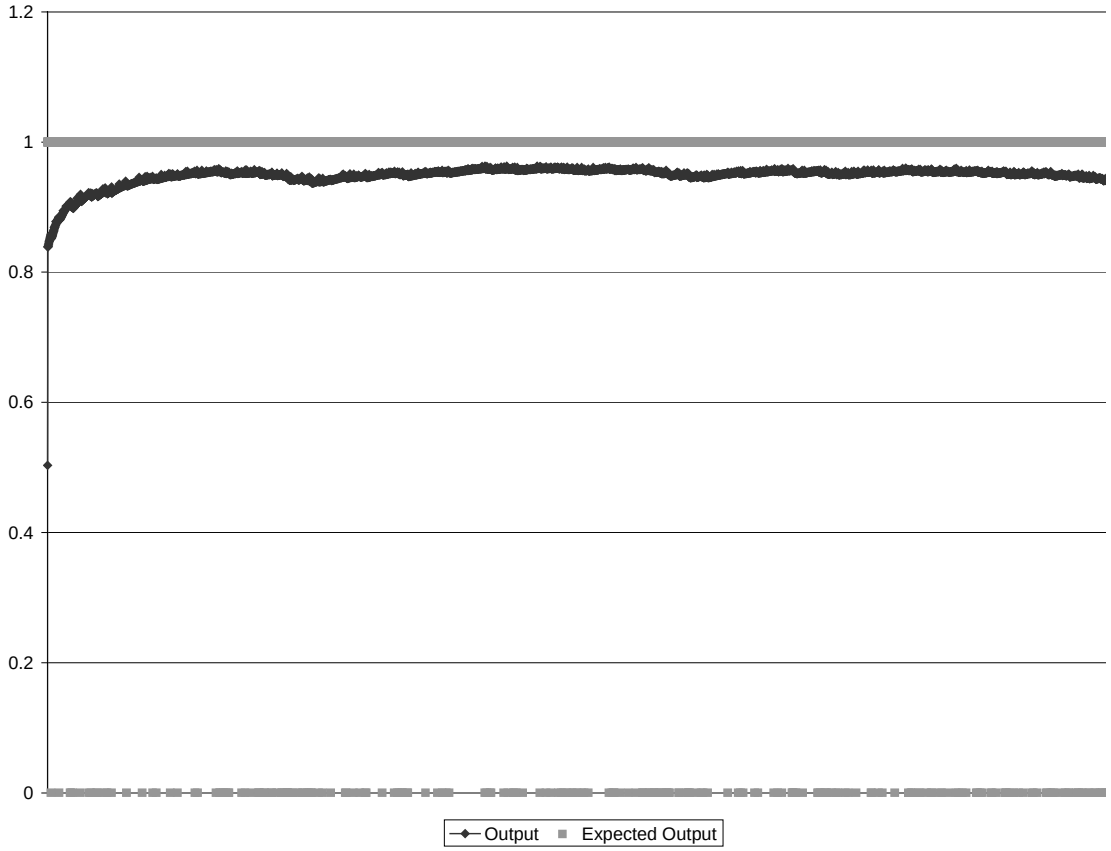


Figure 5.5: Go Training Progress

Overall, it appears that the FFBPANN predictor is defaulting to a purely “branch-always-taken” predictor since it cannot pick out any patterns in the training data set. One idea that is not fully discovered from this data is that the predictor is performing as a branch always predictor because the majority of the branches were “taken”. Since none of the benchmarks had more exclusively “not taken” branches, it cannot be determined whether the predictor would perform exclusively as a “branch-always-not-taken” predictor if there were more “not taken” branches than “taken” branches in the training data.

However, using the Unix command sed, the gcc training data was modified to create a data set that had more “not taken” branches that “taken” branches. The lines that had a branch decision of “not taken” were changed to “taken,” and the lines with “taken” branches were changed to “not taken.” This would effectively give the opposite training data set. The training procedure was run again. The training progress that was expected was that the neural network would very quickly start functioning as a “branch-always-not-taken” predictor. Figure 5.6 shows the true training progress of this “opposite” run.

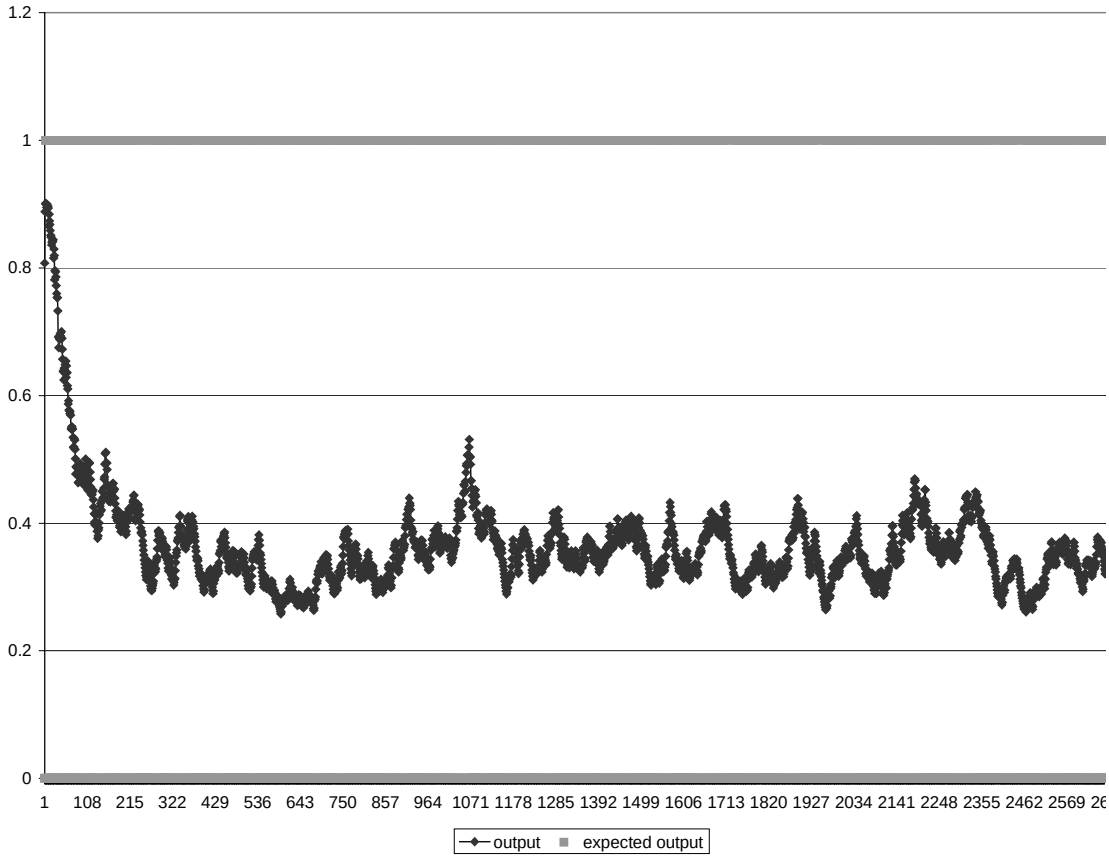


Figure 5.6: “Opposite” Gcc Training Progress

The output of the neural network in this case did trend toward behaving as a “branch-always-not-taken” predictor. The values started high (around 0.7) and tapered off toward 0.3. However, it did not reach these values as quickly as the real gcc data trained toward a “branch-always” predictor. Also, to truly be classified as a “branch-always-not-taken” predictor the output should be closer to 0 (such as a value of 0.12) – just as the “branch-always” predictor trained with the true gcc training data had output values much closer to 1 (about 0.97).

5.3 Branch Predictor Results

For completeness, the other branch predictor results gathered are included here. Table 5.3, Table 5.4, and Table 5.5 show the results for the 2-level, bimodal, and hybrid predictors, respectively. The number of instructions and branches are, of course, the same in each case. The prediction rate is calculated from the total number of successful branches (total branches minus the mispredictions) divided by the total number of branches.

$$\text{prediction rate} = \frac{\text{branches - mispredictions}}{\text{branches}} \quad (5.1)$$

Table 5.3: Branch Predictor Results - 2-level

Simulation	Instructions	Branches	Mispredictions	Prediction Rate
fpppp	174687565856	2774271287	177200663	0.9361
gcc	253018428	50378431	5476501	0.8920
go	32718301958	4828119436	941659343	0.8050
li	55389884875	13200007732	519133700	0.9607
mgrid	110557152489	1444135667	35077426	0.9757
perl	14237817453	2713714020	19806753	0.9927

Table 5.4: Branch Predictor Results - Bimodal

Simulation	Instructions	Branches	Mispredictions	Prediction Rate
fpppp	174687565856	2774271287	216288715	0.9220
gcc	253018428	50378431	5476501	0.8913
go	32718301958	4828119436	880996257	0.8175
li	55389884875	13200007732	1056731094	0.9199
mgrid	110557152489	1444135667	36025665	0.9751
perl	14237817476	2713714028	106292585	0.9608

Table 5.5: Branch Prediction Results - Hybrid

Simulation	Instructions	Branches	Mispredictions	Prediction Rate
fpppp	174687565856	2774271287	166465777	0.9400
gcc	253018428	50378431	4699856	0.9067
go	32718301958	4828119436	848553348	0.8242
li	55389884875	13200007732	513342336	0.9611
mgrid	110557152489	1444135667	34751699	0.9759
perl	14237817453	2713714020	19742856	0.9927

Unfortunately, results for the FFBPANN predictor are not included, because the training did not produce a better branch predictor. Since the FFBPANN appears to have been behaving as a “branch-always-taken” predictor, the prediction rate would be equal to the percent of “taken” branches in the benchmark. Except for the mgrid benchmark, in which 95% of the training data were “taken” branches, the prediction rate would be significantly poorer compared to the other predictors (i.e. about 60%). Even though the FFBPANN predictor was actually a hybrid predictor – a combination of a bimodal and a purely neural network predictor – it would

have performed at best as well as the bimodal predictor. Effectively, the FFBPANN predictor was a combination of a “branch always” predictor and the bimodal predictor.

The more common predictors (2-level and bimodal), used in SimpleScalar, employ much simpler techniques for branch prediction – i.e., from a software perspective, shift registers and tables. They perform very well under most conditions – however, sometimes performance drops drastically, as in the case of the go benchmark. This can be easily observed in Figure 5.7, and Figure 5.8. These figures graph the predictor data presented in the preceding tables. Figure 5.7 shows the predictor performance per predictor. Figure 5.8 displays how the predictors performed in each of the 6 different SPEC95 simulations.

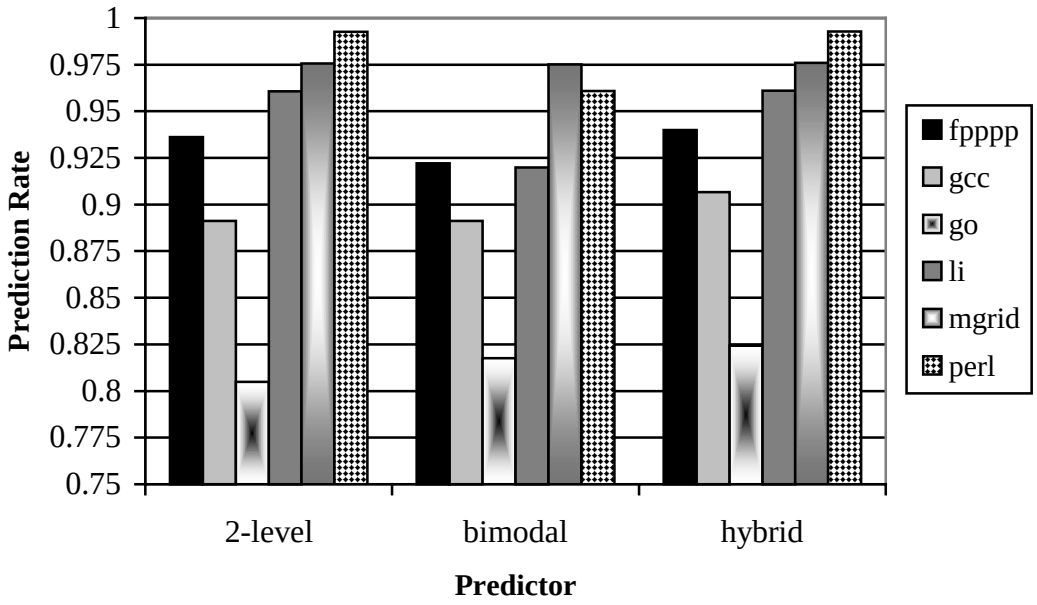


Figure 5.7: Prediction Rate by Predictor

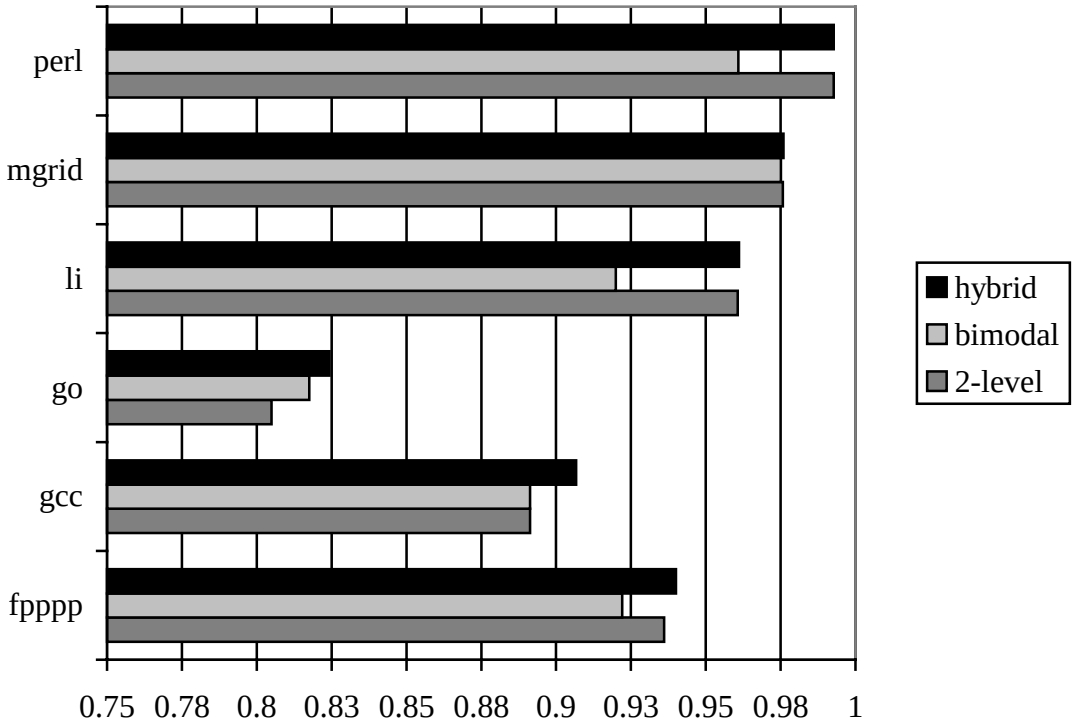


Figure 5.8: Prediction Rate by Simulation

Figure 5.9 shows the overall predictor performance by averaging the number of correct predictions over the total number of branches from all the simulations.

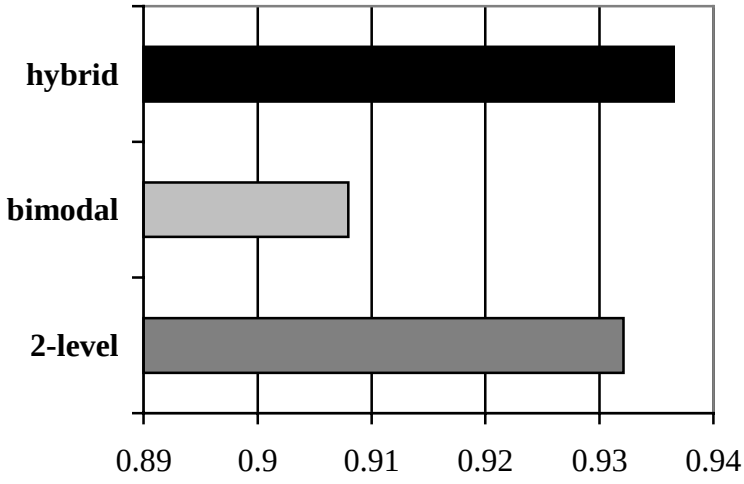


Figure 5.9: Overall Predictor Performance

Chapter 6: Conclusions and Future Work

6.1 Conclusions

From the results, the hybrid, or combination, predictor performs the best. For branch prediction to benefit from an artificial neural network, it can safely be stated that a combination of a purely ANN predictor and another more common predictor (one that has shown consistently good results, such as the bimodal or 2-level predictors) is the proper solution. The fact remains that artificial neural networks' characteristics lend itself well to prediction. However, in this case, a purely feed-forward back-propagation ANN did not perform well at all. Two tasks exist to continue on with testing the use of artificial neural networks for branch prediction. In-depth statistical analysis of branch instruction behavior is needed to determine if another ANN design is better suited for prediction of microprocessor branches. Also, the inputs used for the ANN should be examined closer to determine which inputs should be used and not used to successfully train the ANN.

6.2 Future Work

6.2.1 Other inputs

Further examination of the variables available in SimpleScalar to be used as inputs should be done. Certain inputs may not be desirable and others may have been overlooked. Specifically, the branch address could probably not be used. This prospect should also be tied to the following area of exploration: the selection of better training methods and use of other ANN designs. By changing or adding new inputs a different learning method could be used. For example, if previous branch decisions are taken as inputs, a recurrent learning method could be utilized. Also, variables from previous branch instructions could be examined and taken as inputs.

6.2.2 Other Neural Networks and Training methods

The methodology used for training in this thesis was fairly simple, as the focus was to obtain a working simulation. Other methods of training the ANN should be explored. New training methodologies for Artificial Neural Networks are constantly a topic of research, and new methods may be found that may allow an ANN for branch prediction to be successful. Other artificial neural network styles should be explored. Statistical analysis of branch prediction may hint toward another neural network structure that is better suited for this task. Figure 6.1 shows other learning processes that could be explored.

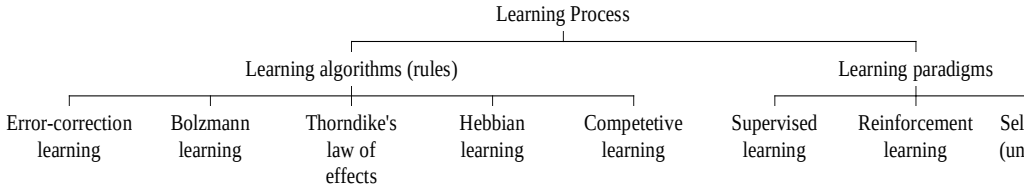


Figure 6.1: the taxonomy of learning

A recurrent network should probably be explored. By choosing a recurrent network, previous branch prediction performance could be taken into account. An example of a recurrent network is shown in Figure 6.2.

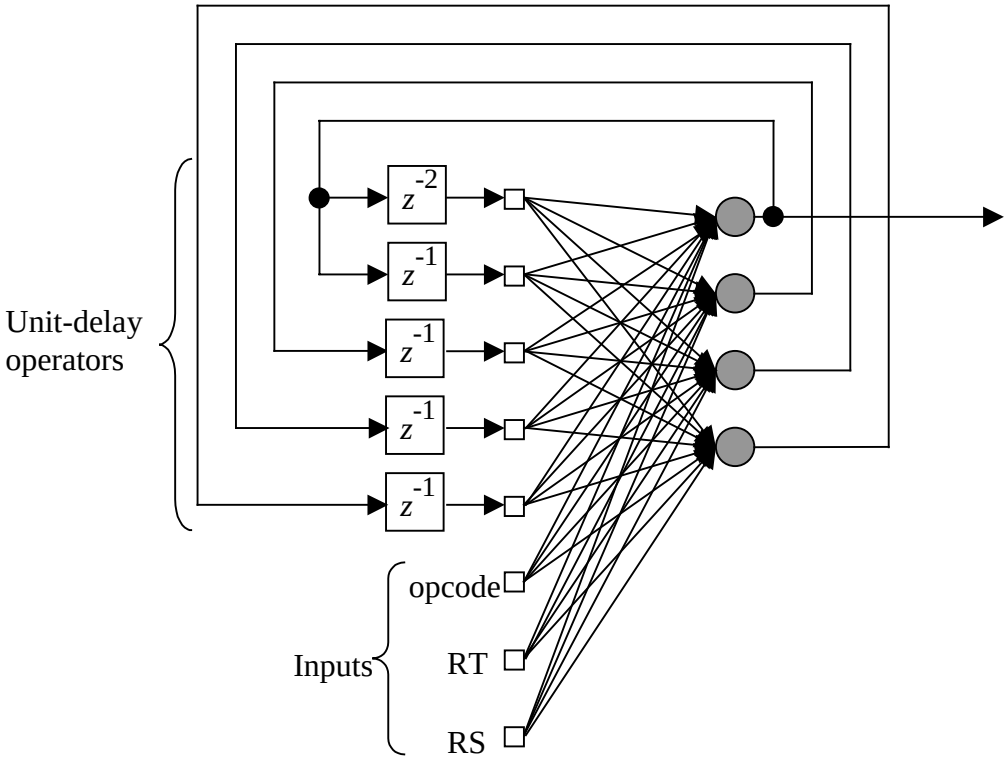


Figure 6.2: Recurrent network

One such recurrent network is a Hopfield network. However, the Hopfield network is equivalent to an associative memory or content-addressable memory [Haykin]. Because of this, the Hopfield network might not be the best choice for branch prediction. Since it is a form of memory – good at retrieving a pattern from memory – and it has been determined that there not very many recurring patterns in the branch prediction training data, the Hopfield network may not improve the neural network as a branch predictor. One limitation of a Hopfield network is that is does not have any hidden neurons. Also, the Hopfield network

cannot be operated in a supervised manner, which could impede training of the network.

If using a recurrent network is expanded upon (i.e. lifting the two limitations discussed for Hopfield networks), one learning method that looks promising is Boltzmann Learning. Taking a stochastic approach to branch prediction seems reasonable, since the question that this would answer is: “what is the probability that this branch instruction will be ‘taken’?” Boltzmann learning should probably operate in the Clamped Condition since the visible neurons should be used to supply the network with inputs from the environment. The Boltzmann learning is a very different learning method from Hebbian learning since there is no error-correction learning [Haykin]. Boltzmann learning has symmetric connections, which allow for two-way communications between neurons. The weight update function, therefore, becomes a function of correlations:

$$\Delta w_{ji} = \eta(\rho_{ji}^{+} - \rho_{ji}^{-}), \qquad \begin{matrix} i, j = 1,2,...,N \\ i \neq j \end{matrix}$$

In this gradient descent rule, the two averages ρ_{ji}^{+} and ρ_{ji}^{-} are the conditional and unconditional correlations, respectively, between the states of neuron j and neuron i [Haykin]. Figure 6.3 shows a basic Boltzmann machine.

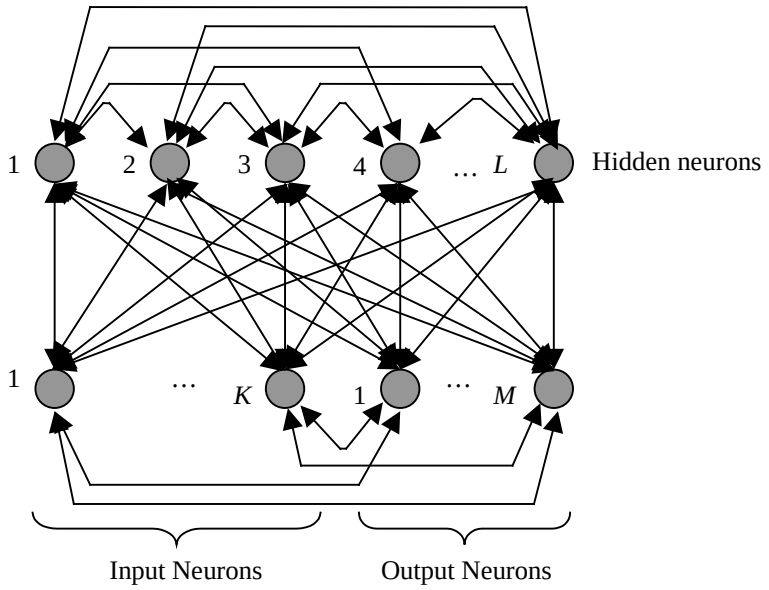


Figure 6.3: Boltzmann Machine

Another option could be to use a Competitive Learning technique. In this manner, two output neurons, one representing a “taken” decision, and the other representing a “not taken” decision,

would compete for an active, or "on" state. The weight update function for a competitive network is determined by which neuron wins the competition:

$$\Delta w = \begin{cases} \eta(x_i - w_{ji}) & \text{if neuron } j \text{ wins the competition} \\ 0 & \text{if neuron } j \text{ loses the competition} \end{cases}$$

Also, it is assumed that the sum of the synaptic weight for a given node is 1:

$$\sum_i w_{ji}^2 = 1 \quad \text{for all } j$$

A simple possibility for a competitive learning network is shown in Figure 6.4.

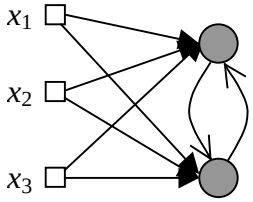


Figure 6.4: Competitive learning network

6.2.3 BTB

Another way that an ANN could be used to improve branch prediction within the SimpleScalar architecture would be to use an ANN for address prediction. The ANN in this thesis was used just for direction prediction (do not take or take the branch). SimpleScalar has the capability for branch address prediction through the use of a Branch-Target Buffer. An ANN could be used to enhance the branch address prediction.

6.2.4 State Output

One possibility that was not explored in this thesis is to train the ANN to be a branch state predictor – using the state of the bimodal or 2-level predictor as the output for training. While re-examining the code for SimpleScalar, it was determined that the coding for the bimodal and 2-level predictors determine which state the predictor is in (0, 1, 2, or 3). When the state is 2 or 3, the prediction is "taken", 0 and 1 are "not taken". Instead of recording the 0 or 1 ("not taken" or "taken") output, the state output could be gathered with the appropriate inputs. Then, instead of training for the "taken" or "not taken" output, the predictor state could be

trained for. In essence, this would be training the ANN to be a better bimodal or 2-level predictor.

6.2.5 Hardware Feasibility

This thesis did not explore the hardware implementation of ANNs for branch prediction. However, both Haykin (in his book, Neural Networks: A Comprehensive Foundation) and Wang (in his dissertation, CMOS VLSI Implementations of a New Feedback Neural Network Architecture) state that VLSI implementations of ANNs are possible, and they are good for real time operations like control, signal processing and pattern recognition. A 6 neuron ANN would occupy 2.2 x 2.2 mm2 using 2μm n-well technology. A 50 neuron CMOS analog chip uses 63,025 transistors and occupies 6.8 mm x 4.6 mm using 2μm CMOS n-well technology [Wang]. The 4x3x2x1 ANN discussed in this thesis uses 10 neurons. In comparison, Intel's Pentium 4 microprocessor contains 42 millions transistors [Intel]. So, if an ANN as a branch predictor can be successfully simulated, a hardware design could be attainable.

APPENDICES

APPENDIX A

FFBPANN C Code

ffbpann.h

```
#include <stdlib.h>

/* Defaults */
#define ALPHA 0.9
#define ETA   0.05
#define GAIN  2
#define BIAS  -1
#define sqr(x)       ((x)*(x))
#define ffbpann_inputs 2
#define sigmoid 1   /* 1 = tansig, 0 = hardlim */

typedef struct {
  double max;
  double min;
} bounds ;

typedef struct {
  int nodes;
  double *errors;
  double *outputs;
  double **weights ;
  double **weightsave ;
  double **deltas;
} layer;

typedef struct {
   int num_layers;       /* number of layers */
   bounds **inputbounds;
   layer *inputlayer;
   layer **layers;       /* array: pointer to layers */
   layer *outputlayer;
   double alpha;         /* momentum */
   double eta;           /* learning rate */
   double gain;          /* sigmoid function gain */
   double error;         /* Net error */
} ffbpann;

ffbpann *create_ffbpann(int, int *);
int init_ffbpann(int, ffbpann *) ;
int train_ffbpann(int, int, double **, ffbpann *, double **);
```

```
int learn_ffbpann(double *, ffbpann *,double *);
int calc_ffbpann(double *, ffbpann *, double *);
void ffbpann_free(ffbpann *);
void updateweights(ffbpann *);
void backprop(ffbpann *);
void restore_weights(ffbpann *);
double nnet_random();
void save_weights(ffbpann *);
```

ffbpann.c

```
#include "ffbpann.h"
#include "misc.h"
#include <math.h>
#include <sys/time.h>

ffbpann *create_ffbpann(layers,nodes)
int layers;
int *nodes;
{
  int i,j ;
  ffbpann *net ;

  if (!(net = (ffbpann *) calloc(1, sizeof(ffbpann))))
    fatal("out of virtual memory");

  if(!(net->layers = (layer **) calloc(layers, sizeof(layer *))))
    fatal("out of virtual memory");

  for(i=0;i<layers;i++){
    if(!(net->layers[i] = (layer *) malloc(sizeof(layer))))
      fatal("out of virtual memory");

    net->layers[i]->nodes = nodes[i];
    if(!(net->layers[i]->errors = (double *)
calloc(nodes[i]+1,sizeof(double))))
      fatal("out of virtual memory");

    if(!(net->layers[i]->outputs = (double *)
calloc(nodes[i]+1,sizeof(double))))
      fatal("out of virtual memory");
    net->layers[i]->outputs[0] = BIAS;

    if(!(net->layers[i]->weights = (double **)
calloc(nodes[i]+1,sizeof(double *))))
```

```
      fatal("out of virtual memory");

    if(!(net->layers[i]->weightsave = (double **)
calloc(nodes[i]+1,sizeof(double *))))
      fatal("out of virtual memory");

    if(!(net->layers[i]->deltas = (double **)
calloc(nodes[i]+1,sizeof(double *))))
      fatal("out of virtual memory");

    if(i!=0) {
      for(j=1;j<=nodes[i];j++) {
        if(!(net->layers[i]->weights[j] =  (double *) calloc(nodes[i-
1]+1,sizeof(double))))
          fatal("out of virtual memory");
        if(!(net->layers[i]->weightsave[j] =  (double *)
calloc(nodes[i-1]+1,sizeof(double))))
          fatal("out of virtual memory");
        if(!(net->layers[i]->deltas[j] =  (double *) calloc(nodes[i-
1]+1,sizeof(double))))
          fatal("out of virtual memory");
      }
    }
  }

  if(!(net->inputbounds = (bounds **) calloc(nodes[0]+1,
sizeof(bounds *))))
    fatal("out of virtual memory");
  for(j=1;j<=nodes[0];j++) {    /* initialize input bounds */
    if(!(net->inputbounds[j] = (bounds *) malloc(sizeof(bounds))))
      fatal("out of virtual memory");
    net->inputbounds[j]->min = 100000000;
    net->inputbounds[j]->max = 0 ;
  }

  net->inputlayer = net->layers[0];
  net->outputlayer = net->layers[layers-1];
  net->alpha = ALPHA ;
  net->eta = ETA ;
  net->gain = GAIN;
  net->num_layers = layers;

  return net;
}

int init_ffbpann(type, net)
```

```
int type;
ffbpann *net;
{
  int i,j,k;
  char *filename ;
  char *ffname = "filename";
  FILE *file;

  if(type == 0){                                    /* RANDOM */
    for(i=1; i<net->num_layers; i++){
      for(j=1; j<=net->layers[i]->nodes; j++){
        for(k=0; k<=net->layers[i-1]->nodes; k++){
          net->layers[i]->weights[j][k] = nnet_random();
        }
      }
    }
  } else {   /* read from file */

  filename = (char *) calloc(13+2*(net->num_layers-
1)+1,sizeof(char));

  /* Create filename string    (I love perl :)  )*/
  file = fopen(ffname,"w");
  for(i=1; i<net->num_layers-1; i++){
    fprintf(file,"%i",net->layers[i]->nodes);
    fprintf(file,"x");
  }
  fprintf(file,"%i",net->layers[net->num_layers-1]->nodes);
  fclose(file);
  file = fopen(ffname,"r");
  fscanf(file,"%s",filename);
  fclose(file);

  filename = strcat(filename,".weights.dump");

  file = fopen(filename, "r");

  /* read in min and max values */
  if(file)
  {
    for(i=1; i<=net->layers[0]->nodes; i++)
    {
      fscanf(file, "%lf %lf", &net->inputbounds[i]->min, &net-
>inputbounds[i]->max);
    }
  }
```

```
  for(i=1; i<net->num_layers; i++){
    for(j=1; j<=net->layers[i]->nodes; j++){
      for(k=0; k<=net->layers[i-1]->nodes; k++){
        if(file != NULL && type == 1)     /**  load trained values **/
        {
          fscanf(file, "%lf", &net->layers[i]->weights[j][k]);
        }
        else                                  /** initialize random values
**/
          net->layers[i]->weights[j][k] = nnet_random();
      }
    }
  }
} /* end else */
  return 1;
}


int train_ffbpann(type,epochs,inputs,net,targets)
int type;
int epochs;
double **inputs;
ffbpann *net;
double **targets;
{
  int i,j,k;
  double output,error ;
  double *out;
  out = (double *) calloc(net->outputlayer->nodes, sizeof(double));

  for(i=0; i<epochs;i++){
    calc_ffbpann(inputs[i],net,out);

    net->error = 0;
    for(j=1; j<=net->outputlayer->nodes;j++) {
      output = net->outputlayer->outputs[j];
      error = targets[i][j-1] - output ;
      net->outputlayer->errors[j] = net->gain * output * (1-output) *
error ;
/*      net->error += 0.5 * sqr(error);
*/     net->error = sqrt(sqr(error));
    }

    backprop(net);
    updateweights(net);
```

```
   free(out);
  }
}

/*  Weight update function */
int learn_ffbpann(input,net,target)
double *input;
ffbpann *net;
double *target;
{
  int i,j,k;
  double output,error ;
  double *out;
  out = (double *) calloc(net->outputlayer->nodes, sizeof(double));

  calc_ffbpann(input,net,out);

  net->error = 0;
  for(j=1; j<=net->outputlayer->nodes;j++) {
     output = net->outputlayer->outputs[j];
     error = target[j-1] - output ;
     net->outputlayer->errors[j] = net->gain * output * (1-output) *
error ;
/*      net->error += 0.5 * sqr(error);
*/     net->error = sqrt(sqr(error));

    }

  backprop(net);
  updateweights(net);
  free(out);
}


int calc_ffbpann(input,net,output)
double *input;
ffbpann *net;
double *output;
{
  int i,j,k;
  double subtotal;
  double min,max;

  for(i=1; i<=net->inputlayer->nodes; i++){
/*    net->inputlayer->outputs[i] =  input[i-1] ;
*/
```

```
   min = net->inputbounds[i]->min ;
   max = net->inputbounds[i]->max ;
   net->inputlayer->outputs[i] = (input[i-1] - min ) / (max - min);
  }

  for(i=0; i<net->num_layers-1; i++){
   for(j=1; j<=net->layers[i+1]->nodes; j++){
subtotal=0;
     for(k=0; k<=net->layers[i]->nodes; k++){
/*printf("%lf times %lf\n", net->layers[i+1]->weights[j][k],  net-
>layers[i]->outputs[k]);*/

       subtotal += net->layers[i+1]->weights[j][k] * net->layers[i]-
>outputs[k];
     }
     if (sigmoid == 1) {   /* tansig */
/*       net->layers[i+1]->outputs[j] = 2/(1+exp(-net->gain *
subtotal))-1;
*/     net->layers[i+1]->outputs[j] = 1/(1+exp(-net->gain *
subtotal));
     } else if (sigmoid == 2) {  /* sum */
        net->layers[i+1]->outputs[j] = subtotal;
     } else if (sigmoid == 0) { /* hardlim */
        if(subtotal <0) { net->layers[i+1]->outputs[j] = 0; }
        else { net->layers[i+1]->outputs[j] = 1; }
     }
   }
  }

  for(i=1;i<=net->outputlayer->nodes;i++){
   output[i-1] = net->outputlayer->outputs[i] ;
  }

  return 1;
}

void ffbpann_free(net)
ffbpann *net;
{
  int i,j;
  void free();

  for(i=1;i<=net->num_layers;i++){
   for(j=1;j<=net->layers[i]->nodes;j++ ){
     free(&net->layers[i]->weights[j]);
     free(&net->layers[i]->deltas[j]);
```

```
    }
   }
/*   free(net->layers);
*/
   free(*net);
   free(net);

}


void backprop(net)
ffbpann* net;
{
  int  i,j,k;
  double output, error;

  for(i=net->num_layers-1;i>1; i--){
    for (j=1; j<=net->layers[1-1]->nodes; j++) {
      output = net->layers[i-1]->outputs[j];
      error = 0;
      for (k=1; k<=net->layers[i]->nodes; k++) {

        error += net->layers[i]->weights[k][j] * net->layers[i]-
>errors[k];
      }

      net->layers[i-1]->errors[j] = net->gain * output * (1-output) *
error;

    }
  }
}


void updateweights(ffbpann* net)
{
  int  l,i,j;
  double output, error, delta;

  for (l=1; l<net->num_layers; l++) {
    for (i=1; i<=net->layers[l]->nodes; i++) {
      for (j=0; j<=net->layers[l-1]->nodes; j++) {
        output = net->layers[l-1]->outputs[j];
        error = net->layers[l]->errors[i];
        delta = net->layers[l]->deltas[i][j];
        net->layers[l]->weights[i][j] += net->eta * error * output +
```

```
net->alpha * delta;
       net->layers[l]->deltas[i][j] = net->eta * error * output;
     }
    }
  }
}


double nnet_random()
{
struct timeval tp;
struct timezone tzp;

gettimeofday(&tp, &tzp);

srandom(tp.tv_usec);

return (double) (random()%1001)/1000;
}

int randint(int low, int high)
{
int r;
struct timeval tp;
struct timezone tzp;

gettimeofday(&tp, &tzp);
srand(tp.tv_usec);
r= rand();
return rand() % (high - low + 1) + low;
}

void dump_weights(net)
ffbpann *net;
{
  char *filename ;
  FILE *file;
  int i,j,k;
  char *ffname = "filename" ;

  filename = (char *) calloc(13+2*(net->num_layers-
1)+1,sizeof(char));

  file = fopen(ffname,"w");
  for(i=1; i<net->num_layers-1; i++){
    fprintf(file,"%i",net->layers[i]->nodes);
```

```
  fprintf(file,"x");
 }
 fprintf(file,"%i",net->layers[net->num_layers-1]->nodes);
 fclose(file);

 file = fopen(ffname,"r");
 fscanf(file,"%s",filename);
 fclose(file);

 filename = strcat(filename,".weights.dump");

 file = fopen(filename, "w");
 if (!file)
   fatal("cannot open weight dump file `%s'", filename);

 /* write min and max values */
 for(i=1; i<=net->layers[0]->nodes; i++)
 {
  fprintf(file, "%lf %lf\n", net->inputbounds[i]->min, net-
>inputbounds[i]->max);
 }

 for(i=1; i<net->num_layers; i++){
  for(j=1; j<=net->layers[i]->nodes; j++){
   for(k=0; k<=net->layers[i-1]->nodes; k++){
    fprintf(file, "%lf\n", net->layers[i]->weights[j][k]);
   }
  }
 }
 fclose(file);
 free(filename);
}

void save_weights(net)
ffbpann *net;
{
 int i,j,k;

 for (i=1; i<net->num_layers; i++) {
  for (j=1; j<=net->layers[i]->nodes; j++) {
   for (k=0; k<=net->layers[i-1]->nodes; k++) {
    net->layers[i]->weightsave[j][k] = net->layers[i]-
>weights[j][k];
   }
  }
 }
```

```
}

void restore_weights(net)
ffbpann *net;
{
  int i,j,k;

  for (i=1; i<net->num_layers; i++) {
    for (j=1; j<=net->layers[i]->nodes; j++) {
      for (k=0; k<=net->layers[i-1]->nodes; k++) {
        net->layers[i]->weights[j][k] = net->layers[i]-
>weightsave[j][k];
      }
    }
  }
}
```

misc.h

```
/*
 * misc.h - miscellaneous interfaces
 *
 * This file is a part of the SimpleScalar tool suite written by
 * Todd M. Austin as a part of the Multiscalar Research Project.
 *
 * The tool suite is currently maintained by Doug Burger and Todd
M. Austin.
 *
 * Copyright (C) 1994, 1995, 1996, 1997 by Todd M. Austin
 *
 * This source file is distributed "as is" in the hope that it will be
 * useful.  The tool set comes with no warranty, and no author or
 * distributor accepts any responsibility for the consequences of its
 * use.
 *
 * Everyone is granted permission to copy, modify and redistribute
 * this tool set under the following conditions:
 *
 *    This source code is distributed for non-commercial use only.
 *    Please contact the maintainer for restrictions applying to
 *    commercial use.
 *
 *    Permission is granted to anyone to make or distribute copies
 *    of this source code, either as received or modified, in any
 *    medium, provided that all copyright notices, permission and
 *    nonwarranty notices are preserved, and that the distributor
```

```
 *    grants the recipient permission for further redistribution as
 *    permitted by this document.
 *
 *    Permission is granted to distribute this file in compiled
 *    or executable form under the same conditions that apply for
 *    source code, provided that either:
 *
 *    A. it is accompanied by the corresponding machine-readable
 *       source code,
 *    B. it is accompanied by a written offer, with no time limit,
 *       to give anyone a machine-readable copy of the
corresponding
 *       source code in return for reimbursement of the cost of
 *       distribution.  This written offer must permit verbatim
 *       duplication by anyone, or
 *    C. it is distributed by someone who received only the
 *       executable form, and is accompanied by a copy of the
 *       written offer of source code that they received concurrently.
 *
 * In other words, you are welcome to use, share and improve this

 * source file.  You are forbidden to forbid anyone else to use,
share
 * and improve what you give them.
 *
 * INTERNET: dburger@cs.wisc.edu
 * US Mail:  1210 W. Dayton Street, Madison, WI 53706
 *
 * $Id: misc.h,v 1.4 1997/03/11 01:18:24 taustin Exp taustin $
 *
 * $Log: misc.h,v $
 * Revision 1.4  1997/03/11  01:18:24  taustin
 * updated copyright
 * supported added for non-GNU C compilers
 * ANSI C compiler check added
 * long/int tweaks made for ALPHA target support
 * hacks added to make SYMCAT() portable
 *
 * Revision 1.3  1997/01/06  16:02:01  taustin
 * comments updated
 * system prototypes deleted (-Wall flag no longer a clean compile)
 *
 * Revision 1.1  1996/12/05  18:50:23  taustin
 * Initial revision
 *
 *
```

```
 */

#ifndef MISC_H
#define MISC_H

#include <stdio.h>
#include <string.h>
#include <sys/types.h>

#ifndef __STDC__ /* an ansi C compiler is required */
#error The SimpleScalar simulators must be compiled with an
ANSI C compiler.
#endif /* __STDC__ */
#if 0 /* as of rel 2, the tool set should compile with any ansi C
compiler */
/*  FIXME: SimpleScalar currently only compiles with GNU GCC,
since I use
  GNU GCC extensions extensively throughout the simulation
suite, one day
  I'll fix this problem... */
#ifndef __GNUC__
#error The SimpleScalar simulation suite must be compiled with
GNU GCC.
#endif /* __GNUC__ */
#endif

/* enable inlining here */
#undef INLINE
#if defined(__GNUC__)
#define INLINE                  inline
#else
#define INLINE
#endif

/* boolean value defs */
#ifndef TRUE
#define TRUE 1
#endif
#ifndef FALSE
#define FALSE 0
#endif

/* various useful macros */
#ifndef MAX
#define MAX(a, b)    (((a) < (b)) ? (b) : (a))
#endif
```

```
#ifndef MIN
#define MIN(a, b)    (((a) < (b)) ? (a) : (b))
#endif

/* for printing out "long long" vars */
#define LLHIGH(L)            ((int)(((L)>>32) & 0xffffffff))
#define LLLOW(L)             ((int)((L) & 0xffffffff))

/* bind together two symbols, at preprocess time */
#ifdef __GNUC__
/* this works on all GNU GCC targets (that I've seen...) */
#define SYMCAT(X,Y)        X##Y
#else /* !__GNUC__ */
#ifdef OLD_SYMCAT
#define SYMCAT(X,Y)        X/**/Y
#else /* !OLD_SYMCAT */
#define SYMCAT(X,Y)        X##Y
#endif /* OLD_SYMCAT */
#endif /* __GNUC__ */

/* size of an array, in elements */
#define N_ELT(ARR)   (sizeof(ARR)/sizeof((ARR)[0]))

/* rounding macros, assumes ALIGN is a power of two */
#define ROUND_UP(N,ALIGN)      (((N) + ((ALIGN)-1)) &
~((ALIGN)-1))
#define ROUND_DOWN(N,ALIGN)          ((N) & ~((ALIGN)-
1))

/* verbose output flag */
extern int verbose;

#ifdef DEBUG
/* active debug flag */
extern int debugging;
#endif /* DEBUG */

/* register a function to be called when an error is detected */
void
fatal_hook(void (*hook_fn)(FILE *stream));/* fatal hook function
*/

#ifdef __GNUC__
/* declare a fatal run-time error, calls fatal hook function */
#define fatal(fmt, args...)       \
  _fatal(__FILE__, __FUNCTION__, __LINE__, fmt, ## args)
```

```
void
_fatal(char *file, char *func, int line, char *fmt, ...)
__attribute__ ((noreturn));
#else /* !__GNUC__ */
void
fatal(char *fmt, ...);
#endif /* !__GNUC__ */

#ifdef __GNUC__
/* declare a panic situation, dumps core */
#define panic(fmt, args...)      \
  _panic(__FILE__, __FUNCTION__, __LINE__, fmt, ## args)

void
_panic(char *file, char *func, int line, char *fmt, ...)
__attribute__ ((noreturn));
#else /* !__GNUC__ */
void
panic(char *fmt, ...);
#endif /* !__GNUC__ */

#ifdef __GNUC__
/* declare a warning */
#define warn(fmt, args...)       \
  _warn(__FILE__, __FUNCTION__, __LINE__, fmt, ## args)

void
_warn(char *file, char *func, int line, char *fmt, ...);
#else /* !__GNUC__ */
void
warn(char *fmt, ...);
#endif /* !__GNUC__ */

#ifdef __GNUC__
/* print general information */
#define info(fmt, args...)       \
  _info(__FILE__, __FUNCTION__, __LINE__, fmt, ## args)

void
_info(char *file, char *func, int line, char *fmt, ...);
#else /* !__GNUC__ */
void
info(char *fmt, ...);
#endif /* !__GNUC__ */
```

```
#ifdef DEBUG

#ifdef __GNUC__
/* print a debugging message */
#define debug(fmt, args...)     \
   do {                         \
      if (debugging)            \
         _debug(__FILE__, __FUNCTION__, __LINE__, fmt, ##
args); \
   } while(0)

void
_debug(char *file, char *func, int line, char *fmt, ...);
#else /* !__GNUC__ */
void
debug(char *fmt, ...);
#endif /* !__GNUC__ */

#else /* !DEBUG */

#ifdef __GNUC__
#define debug(fmt, args...)
#else /* !__GNUC__ */
/* the optimizer should eliminate this call! */
static void debug(char *fmt, ...) {}
#endif /* !__GNUC__ */

#endif /* !DEBUG */

/* copy a string to a new storage allocation (NOTE: many
machines are missing
   this trivial function, so I funcdup() it here...) */
char *                          /* duplicated string */
mystrdup(char *s);              /* string to duplicate to heap storage
*/

/* find the last occurrence of a character in a string */
char *
mystrrchr(char *s, char c);

/* case insensitive string compare (NOTE: many machines are
missing this
   trivial function, so I funcdup() it here...) */
int                             /* compare result, see strcmp() */
mystricmp(char *s1, char *s2);          /* strings to compare, case
insensitive */
```

```
/* allocate some core, this memory has overhead no larger than a
page
   in size and it cannot be released. the storage is returned cleared */
void *getcore(int nbytes);

/* return log of a number to the base 2 */
int log_base2(int n);

/* return string describing elapsed time, passed in SEC in seconds
*/
char *elapsed_time(long sec);

/* assume bit positions numbered 31 to 0 (31 high order bit),
extract num bits
   from word starting at position pos (with pos as the high order bit
of those
   to be extracted), result is right justified and zero filled to high
order
   bit, for example, extractl(word, 6, 3) w/ 8 bit word = 01101011
returns
   00000110 */
unsigned int
extractl(int word,              /* the word from which to extract */
        int pos,            /* bit positions 31 to 0 */
        int num);           /* number of bits to extract */

#if defined(sparc) && !defined(__svr4__)
#define strtoul strtol
#endif

/* same semantics as fopen()/fclose() except that filenames ending
with a
   ".gz" or ".Z" will be automagically get compressed */
typedef struct {
  enum { ft_file, ft_prog } ft;
  FILE *fd;
} ZFILE;

ZFILE *
zfopen(char *fname, char *type);

int
zfclose(ZFILE *zfd);

#endif /* MISC_H */
```

misc.c

```
/*
 * misc.c - miscellaneous routines
 *
 * This file is a part of the SimpleScalar tool suite written by
 * Todd M. Austin as a part of the Multiscalar Research Project.
 *
 * The tool suite is currently maintained by Doug Burger and Todd
M. Austin.
 *
 * Copyright (C) 1994, 1995, 1996, 1997 by Todd M. Austin
 *
 * This source file is distributed "as is" in the hope that it will be
 * useful.  The tool set comes with no warranty, and no author or
 * distributor accepts any responsibility for the consequences of its
 * use.
 *
 * Everyone is granted permission to copy, modify and redistribute
 * this tool set under the following conditions:
 *
 *    This source code is distributed for non-commercial use only.
 *    Please contact the maintainer for restrictions applying to
 *    commercial use.
 *
 *    Permission is granted to anyone to make or distribute copies
 *    of this source code, either as received or modified, in any
 *    medium, provided that all copyright notices, permission and
 *    nonwarranty notices are preserved, and that the distributor
 *    grants the recipient permission for further redistribution as
 *    permitted by this document.
 *
 *    Permission is granted to distribute this file in compiled
 *    or executable form under the same conditions that apply for
 *    source code, provided that either:
 *
 *    A. it is accompanied by the corresponding machine-readable
 *       source code,
 *    B. it is accompanied by a written offer, with no time limit,
 *       to give anyone a machine-readable copy of the
corresponding
 *       source code in return for reimbursement of the cost of
 *       distribution.  This written offer must permit verbatim
 *       duplication by anyone, or
 *    C. it is distributed by someone who received only the
```

```
 *       executable form, and is accompanied by a copy of the
 *       written offer of source code that they received concurrently.
 *
 * In other words, you are welcome to use, share and improve this
 * source file.  You are forbidden to forbid anyone else to use,
share
 * and improve what you give them.
 *
 * INTERNET: dburger@cs.wisc.edu
 * US Mail:  1210 W. Dayton Street, Madison, WI 53706
 *
 * $Id: misc.c,v 1.5 1997/03/11 01:17:36 taustin Exp taustin $
 *
 * $Log: misc.c,v $
 * Revision 1.5  1997/03/11  01:17:36  taustin
 * updated copyright
 * supported added for non-GNU C compilers
 *
 * Revision 1.4  1997/01/06  16:01:45  taustin
 * comments updated
 *
 * Revision 1.1  1996/12/05  18:52:32  taustin
 * Initial revision
 *
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <ctype.h>

#include "misc.h"

/* verbose output flag */
int verbose = FALSE;

#ifdef DEBUG
/* active debug flag */
int debugging = FALSE;
#endif /* DEBUG */

/* fatal function hook, this function is called just before an exit
   caused by a fatal error, used to spew stats, etc. */
static void (*hook_fn)(FILE *stream) = NULL;
```

```
/* register a function to be called when an error is detected */
void
fatal_hook(void (*fn)(FILE *stream))          /* fatal hook function
*/
{
  hook_fn = fn;
}

/* declare a fatal run-time error, calls fatal hook function */
#ifdef __GNUC__
void
_fatal(char *file, char *func, int line, char *fmt, ...)
#else /* !__GNUC__ */
void
fatal(char *fmt, ...)
#endif /* __GNUC__ */
{
  va_list v;
  va_start(v, fmt);

  fprintf(stderr, "fatal: ");
  vfprintf(stderr, fmt, v);
#ifdef __GNUC__
  if (verbose)
    fprintf(stderr, " [%s:%s, line %d]", func, file, line);
#endif /* __GNUC__ */
  fprintf(stderr, "\n");
  if (hook_fn)
    (*hook_fn)(stderr);
  exit(1);
}

/* declare a panic situation, dumps core */
#ifdef __GNUC__
void
_panic(char *file, char *func, int line, char *fmt, ...)
#else /* !__GNUC__ */
void
panic(char *fmt, ...)
#endif /* __GNUC__ */
{
  va_list v;
  va_start(v, fmt);

  fprintf(stderr, "panic: ");
```

```
  vfprintf(stderr, fmt, v);
#ifdef __GNUC__
  fprintf(stderr, " [%s:%s, line %d]", func, file, line);
#endif /* __GNUC__ */
  fprintf(stderr, "\n");
  if (hook_fn)
    (*hook_fn)(stderr);
  abort();
}

/* declare a warning */
#ifdef __GNUC__
void
_warn(char *file, char *func, int line, char *fmt, ...)
#else /* !__GNUC__ */
void
warn(char *fmt, ...)
#endif /* __GNUC__ */
{
  va_list v;
  va_start(v, fmt);

  fprintf(stderr, "warning: ");
  vfprintf(stderr, fmt, v);
#ifdef __GNUC__
  if (verbose)
    fprintf(stderr, " [%s:%s, line %d]", func, file, line);
#endif /* __GNUC__ */
  fprintf(stderr, "\n");
}

/* print general information */
#ifdef __GNUC__
void
_info(char *file, char *func, int line, char *fmt, ...)
#else /* !__GNUC__ */
void
info(char *fmt, ...)
#endif /* __GNUC__ */
{
  va_list v;
  va_start(v, fmt);

  vfprintf(stderr, fmt, v);
#ifdef __GNUC__
  if (verbose)
```

```
    fprintf(stderr, " [%s:%s, line %d]", func, file, line);
#endif /* __GNUC__ */
  fprintf(stderr, "\n");
}

#ifdef DEBUG
/* print a debugging message */
#ifdef __GNUC__
void
_debug(char *file, char *func, int line, char *fmt, ...)
#else /* !__GNUC__ */
void
debug(char *fmt, ...)
#endif /* __GNUC__ */
{
    va_list v;
    va_start(v, fmt);

    if (debugging)
      {
        fprintf(stderr, "debug: ");
        vfprintf(stderr, fmt, v);
#ifdef __GNUC__
        fprintf(stderr, " [%s:%s, line %d]", func, file, line);
#endif
        fprintf(stderr, "\n");
      }
}
#endif /* DEBUG */


/* copy a string to a new storage allocation (NOTE: many
machines are missing
   this trivial function, so I funcdup() it here...) */
char *                          /* duplicated string */
mystrdup(char *s)               /* string to duplicate to heap storage
*/
{
  char *buf;

  if (!(buf = (char *)malloc(strlen(s)+1)))
    return NULL;
  strcpy(buf, s);
  return buf;
}
```

```
/* find the last occurrence of a character in a string */
char *
mystrrchr(char *s, char c)
{
  char *rtnval = 0;

  do {
    if (*s == c)
      rtnval = s;
  } while (*s++);

  return rtnval;
}

/* case insensitive string compare (NOTE: many machines are
missing this
   trivial function, so I funcdup() it here...) */
int                                     /* compare result, see strcmp() */
mystricmp(char *s1, char *s2)           /* strings to compare, case
insensitive */
{
  unsigned char u1, u2;

  for (;;)
    {
      u1 = (unsigned char)*s1++; u1 = tolower(u1);
      u2 = (unsigned char)*s2++; u2 = tolower(u2);

      if (u1 != u2)
            return u1 - u2;
      if (u1 == '\0')
            return 0;
    }
}

/* allocate some core, this memory has overhead no larger than a
page
   in size and it cannot be released. the storage is returned cleared */
void *
getcore(int nbytes)
{
#ifndef PURIFY
  void *p = (void *)sbrk(nbytes);

  if (p == (void *)-1)
    return NULL;
```

```
  /* this may be superfluous */
#if defined(__svr4__)
  memset(p, '\0', nbytes);
#else /* !defined(__svr4__) */
  bzero(p, nbytes);
#endif
  return p;
#else
  return calloc(nbytes, 1);
#endif /* PURIFY */
}

/* return log of a number to the base 2 */
int
log_base2(int n)
{
  int power = 0;

  if (n <= 0 || (n & (n-1)) != 0)
    panic("log2() only works for positive power of two values");

  while (n >>= 1)
    power++;

  return power;
}

/* return string describing elapsed time, passed in SEC in seconds
*/
char *
elapsed_time(long sec)
{
  static char tstr[256];
  char temp[256];

  if (sec <= 0)
    return "0s";

  tstr[0] = '\0';

  /* days */
  if (sec >= 86400)
    {
      sprintf(temp, "%ldD ", sec/86400);
      strcat(tstr, temp);
```

```
    sec = sec % 86400;
   }
  /* hours */
  if (sec >= 3600)
   {
    sprintf(temp, "%ldh ", sec/3600);
    strcat(tstr, temp);
    sec = sec % 3600;
   }
  /* mins */
  if (sec >= 60)
   {
    sprintf(temp, "%ldm ", sec/60);
    strcat(tstr, temp);
    sec = sec % 60;
   }
  /* secs */
  if (sec >= 1)
   {
    sprintf(temp, "%lds ", sec);
    strcat(tstr, temp);
   }
  tstr[strlen(tstr)-1] = '\0';
  return tstr;
}

/* assume bit positions numbered 31 to 0 (31 high order bit),
extract num bits
   from word starting at position pos (with pos as the high order bit
of those
   to be extracted), result is right justified and zero filled to high
order
   bit, for example, extractl(word, 6, 3) w/ 8 bit word = 01101011
returns
   00000110 */
unsigned int
extractl(int word,                  /* the word from which to extract */
        int pos,            /* bit positions 31 to 0 */
        int num)            /* number of bits to extract */
{
    return(((unsigned int) word >> (pos + 1 - num)) & ~(~0 <<
num));
}

#if 0 /* FIXME: not portable... */
```

```
static struct {
  char *type;
  char *ext;
  char *cmd;
} zfcmds[] = {
  /* type */      /* extension */          /* command */
  { "r",  ".gz",                    "gzip -dc < %s" },
  { "r",  ".Z",                     "uncompress -c < %s" },
  { "rb",         ".gz",                    "gzip -dc < %s" },
  { "rb",         ".Z",                     "uncompress -c < %s" },
  { "w",          ".gz",                    "gzip > %s" },
  { "w",          ".Z",                     "compress > %s" },
  { "wb",         ".gz",                    "gzip > %s" },
  { "wb",         ".Z",                     "compress > %s" }
};

/* same semantics as fopen() except that filenames ending with a
".gz" or ".Z"
   will be automagically get compressed */
ZFILE *
zfopen(char *fname, char *type)
{
  int i;
  char *cmd = NULL, *ext;
  ZFILE *zfd;

  /* get the extension */
  ext = mystrrchr(fname, '.');
  if (ext != NULL)
    {
      if (*ext != '\0')
          {
            for (i=0; i < N_ELT(zfcmds); i++)
              if (!strcmp(zfcmds[i].type, type) &&
!strcmp(zfcmds[i].ext, ext))
                {
                    cmd = zfcmds[i].cmd;
                    break;
                }
          }
    }

  zfd = (ZFILE *)calloc(1, sizeof(ZFILE));
  if (!zfd)
    fatal("out of virtual memory");
```

```
  if (cmd)
    {
      char str[2048];

      sprintf(str, cmd, fname);

      zfd->ft = ft_prog;
      zfd->fd = popen(str, type);
    }
  else
    {
      zfd->ft = ft_file;
      zfd->fd = fopen(fname, type);
    }

  if (!zfd->fd)
    {
      free(zfd);
      zfd = NULL;
    }
  return zfd;
}

int
zfclose(ZFILE *zfd)
{
  switch (zfd->ft)
    {
    case ft_file:
      fclose(zfd->fd);
      zfd->fd = NULL;
      break;

    case ft_prog:
      pclose(zfd->fd);
      zfd->fd = NULL;
      break;

    default:
      panic("bogus file type");
    }

  free(zfd);

  return TRUE;
}
```

```
#endif
```

APPENDIX B

Training and Testing Programs

train.xor.c

```
#include <math.h>
#include <unistd.h>
#include <stdio.h>
#include "ffbpann.h"

main(argc,argv)
int argc;
char **argv;
{
  int i,r,j,k,l;
  int layers = 3;
  int scan;
  ffbpann *net;
  FILE *file;
/*  double input[ffbpann_inputs];
*/
  double inputs[ffbpann_inputs];
  double output,error,train_error,test_error;
  double target[1];
  double *out;
  int nodes[layers];
  double min_error = 10000000 ;
  int stop = 0;
  char *filename = "train.data.new";
  int lines = 0;
  int epochs = 50;

  out = (double *) calloc(1, sizeof(double));

  nodes[0] = ffbpann_inputs ;     /* input nodes (baddr, opcode, rs,
rt) */
  nodes[1] = 2;
  nodes[2] = 1;

  net=create_ffbpann(layers,nodes);

  if(argv[1])
  {
    init_ffbpann(1,net);
```

```
  }
  else
  {
   init_ffbpann(0,net);

   file = fopen(filename, "r");
   if(!file)
   {
    printf("can't open training data file: %s\n", filename);
    exit(1);
   }

   /* normalize */
   while( fscanf(file, "%lf %lf %lf", &inputs[0], &inputs[1],
&target[0]) > 0 )
   {
    lines++;
    for(i=1;i<=ffbpann_inputs;i++)
    {
     if(net->inputbounds[i]->max < inputs[i-1])
       net->inputbounds[i]->max = inputs[i-1] ;
     if(net->inputbounds[i]->min > inputs[i-1])
       net->inputbounds[i]->min = inputs[i-1];
    }
   }

   fclose(file);
  }

  for(i=1;i<=ffbpann_inputs;i++)
  {
   printf("%i: max: %lf, min: %lf\n", i,net->inputbounds[i]-
>max,net->inputbounds[i]->min);
  }
  fflush(stdout);

  file = fopen(filename, "r");
  if(!file)
  {
   printf("can't open training data file: %s\n", filename);
   exit(1);
  }

  do {
   /* TRAIN */
   train_error = 0;
```

```
   for(j=1;j<=epochs;j++)     /* epochs */
   {
    r = randint(1, (int) (.6*lines)-1);
    rewind(file);

    for(l=0;l<r;l++)
    {
     fscanf(file, "%lf %lf %lf", &inputs[0], &inputs[1],
&target[0]);
    }
/*
printf("%lf %lf\n",inputs[0],inputs[1]);
fflush(stdout);
*/

    calc_ffbpann(inputs,net,out);
    net->error = 0;
    for(i=1; i<=net->outputlayer->nodes; i++) {
     output = net->outputlayer->outputs[1];
     error = target[i-1] - output ;
     net->outputlayer->errors[1] = net->gain * output * (1-output)
* error ;
/*       net->error += 0.5 * sqr(error); */
     net->error += sqrt (sqr(error));
    }

    backprop(net);
    updateweights(net);
   }


   /* TEST */
printf("TESTING");fflush(stdout);
   rewind(file);
   train_error = 0;
   for(l=0;l<0.6*lines;l++) /* skip over training data set */
   {
    fscanf(file, "%lf %lf %lf", &inputs[0], &inputs[1], &target[0]);
    calc_ffbpann(inputs,net,out);
    net->error = 0;
    for(i=1; i<=net->outputlayer->nodes; i++) {
     output = net->outputlayer->outputs[1];
     error = target[i-1] - output ;
     net->outputlayer->errors[1] = net->gain * output * (1-output)
* error ;
/*       net->error += 0.5 * sqr(error); */
```

```
        net->error += sqrt (sqr(error));
       }
       train_error += net->error;
     }

     test_error = 0;
     for(l=0.6*lines; l<0.9*lines;l++)  /* the test data set */
     {
  fscanf(file, "%lf %lf %lf", &inputs[0], &inputs[1], &target[0]);
       calc_ffbpann(inputs,net,out);
       net->error = 0;
       for(i=1; i<=net->outputlayer->nodes; i++) {
         output = net->outputlayer->outputs[1];
         error = target[i-1] - output ;
         net->outputlayer->errors[1] = net->gain * output * (1-output)
* error ;
/*         net->error += 0.5 * sqr(error); */
         net->error += sqrt (sqr(error));
       }
       test_error += net->error ;
     }
     rewind(file);
     printf("\nAve. Train Error: %0.3lf - Ave. Test Error: %0.3lf\n",
train_error/(0.6*lines), test_error/(0.9*lines - 0.6*lines));
printf("test_error: %lf train_error: %lf min_error:
%lf\n",test_error, train_error, min_error);
     fflush(stdout);
     if(test_error < min_error)
     {
       min_error = test_error;
       save_weights(net);
       printf(" ...saving weights\n");
       fflush(stdout);
     }
     else if (test_error > 1.2 * min_error)
     {
       printf(" End Error: %lf",min_error);
       fflush(stdout);
       stop = 1;
       restore_weights(net);
     }

     dump_weights(net);

   } while(!stop);
```

```
  free(out);
  fclose(file);
 }
```

test.xor.c

```
 #include <math.h>
 #include <unistd.h>
 #include <stdio.h>
 #include "ffbpann.h"

 main(argc,argv)
 int argc;
 char **argv;
 {
  int i,r,j,k;
  int layers = 3;
  int scan;
  ffbpann *net;
  FILE *file;
  double input[ffbpann_inputs];
  double inputs[ffbpann_inputs];
  double output,error,train_error,test_error;
  double target[1];
  double *out;
  int nodes[layers];
  double min_error = 10000000 ;
  int stop = 0;
  char *filename = "train.data";
  int lines = 0;
  int epochs = 50;

  out = (double *) calloc(1, sizeof(double));

  nodes[0] = ffbpann_inputs ;     /* input nodes (baddr, opcode, rs,
 rt) */
  nodes[1] = 2;
  nodes[2] = 1;

  net=create_ffbpann(layers,nodes);
  init_ffbpann(1,net);

  file = fopen(filename, "r");
  if(!file)
  {
```

```
    printf("can't open training data file: %s\n", filename);
    exit(1);
  }

  /* TEST */
  test_error = 0;
  for(i=0;i<4;i++)
  {
    fscanf(file, "%lf %lf %lf", &input[0], &input[1], &target[0]);
printf("%lf %lf %lf\n", input[0], input[1], target[0]);
    for(k=0;k<ffbpann_inputs;k++)
    {
      inputs[k] = (double) input[k];
printf("%lf %lf\n", input[k],inputs[k]);
fflush(stdout);
    }
   calc_ffbpann(inputs,net,out);
    output = net->outputlayer->outputs[1];
    printf("inputs: %lf %lf   output: %lf\n", inputs[0], inputs[1],
output);
  }
  free(out);
  fclose(file);
}
```

train.sunspots.c

```
#include <math.h>
#include <unistd.h>
#include <stdio.h>
#include "ffbpann.h"
#include "sunspots.h"

#define ffbpann_inputs 30

main(argc,argv)
int argc;
char **argv;
{
  int i,r,j,k,l;
  int layers = 3;
  int scan;
  ffbpann *net;
/*  double input[ffbpann_inputs];
*/
```

```
  double inputs[ffbpann_inputs];
  double output,error,train_error,test_error;
  double target[1];
  double *out;
  int nodes[layers];
  double min_error = 10000000 ;
  int stop = 0;
  int epochs = 10;
  double Min = MAX_double;
  double Max = MIN_double;

  out = (double *) calloc(1, sizeof(double));

  nodes[0] = ffbpann_inputs ;     /* input nodes (baddr, opcode, rs,
rt) */
  nodes[1] = 10;
  nodes[2] = 1;

  net=create_ffbpann(layers,nodes);
  init_ffbpann(0, net);

/*  NormalizeSunspots();*/

  for (i=0; i<NUM_YEARS; i++) {
    Min = MIN(Min, Sunspots[i]);
    Max = MAX(Max, Sunspots[i]);
  }

  for(i=1;i<=ffbpann_inputs;i++)
  {
    net->inputbounds[i]->max = Max ;
    net->inputbounds[i]->min = Min ;
/*    printf("%i: max: %lf, min: %lf\n", i,net->inputbounds[i]-
>max,net->inputbounds[i]->min);
fflush(stdout);
*/
  }

  do {
    /* TRAIN */
    train_error = 0;
    for(j=1;j<=epochs*TRAIN_YEARS;j++)     /* epochs */
    {
      r = randint(TRAIN_LWB, TRAIN_UPB);
      for(i=0; i<net->inputlayer->nodes; i++) {
        inputs[i] = Sunspots[r-N+i] ;
```

```
      }
      target[0] = Sunspots[r] ;

      calc_ffbpann(inputs,net,out);
      net->error = 0;
      for(i=1; i<=net->outputlayer->nodes; i++) {
        output = net->outputlayer->outputs[1];
        error = target[i-1] - output ;
        net->outputlayer->errors[i] = net->gain * output * (1-output)
* error ;
/*        net->error += 0.5 * sqr(error); */
        net->error += sqrt (sqr(error));
      }

      backprop(net);
      updateweights(net);
    }


    /* TEST */
    train_error = 0;
    for(l=TRAIN_LWB;l<=TRAIN_UPB;l++) /* skip over training
data set */
    {
      for(i=0; i<net->inputlayer->nodes; i++) {
        inputs[i] = Sunspots[l-N+i] ;
      }
      target[0] = Sunspots[l] ;

      calc_ffbpann(inputs,net,out);
      net->error = 0;
      for(i=1; i<=net->outputlayer->nodes; i++) {
        output = net->outputlayer->outputs[1];
        error = target[i-1] - output ;
        net->outputlayer->errors[i] = net->gain * output * (1-output)
* error ;
/*        net->error += 0.5 * sqr(error); */
        net->error += sqrt (sqr(error));
      }
      train_error += net->error;
    }

    test_error = 0;
    for(l=TEST_LWB; l<=TEST_UPB;l++)  /* the test data set */
    {
      for(i=0; i<net->inputlayer->nodes; i++) {
```

```
        inputs[i] = Sunspots[l-N+i] ;
      }
      target[0] = Sunspots[l] ;

      calc_ffbpann(inputs,net,out);
      net->error = 0;
      for(i=1; i<=net->outputlayer->nodes; i++) {
        output = net->outputlayer->outputs[1];
        error = target[i-1] - output ;
        net->outputlayer->errors[i] = net->gain * output * (1-output)
* error ;
/*        net->error += 0.5 * sqr(error); */
        net->error += sqrt (sqr(error));
      }
      test_error += net->error ;
    }
    printf("\nAve. Train Error: %0.3lf - Ave. Test Error: %0.3lf\n",
train_error/(TRAIN_UPB-TRAIN_LWB), test_error/(TEST_UPB-
TEST_LWB));
printf("test_error: %lf train_error: %lf min_error:
%lf\n",test_error, train_error, min_error);
    fflush(stdout);
    if(test_error < min_error)
    {
      min_error = test_error;
      save_weights(net);
      printf(" ...saving weights\n");
      fflush(stdout);
    }
    else if (test_error > 1.2 * min_error)
    {
      printf(" End Error: %lf\n",min_error);
      fflush(stdout);
      stop = 1;
      restore_weights(net);
    }

    dump_weights(net);
  } while(!stop);

  free(out);
}
```

test.sunspots.c

```
#include <math.h>
#include <unistd.h>
#include <stdio.h>
#include "ffbpann.h"
#include "sunspots.h"

#define ffbpann_inputs 30

main(argc,argv)
int argc;
char **argv;
{
  int i,r,j,k,l;
  int layers = 3;
  int scan;
  ffbpann *net;
  FILE *file;
/*  double input[ffbpann_inputs];
*/
  double inputs[ffbpann_inputs];
  double output,error,train_error,test_error;
  double target[1];
  double *out;
  int nodes[layers];
  double min_error = 10000000 ;
  int stop = 0;
  int epochs = 10;
  double Min = MAX_double;
  double Max = MIN_double;

  out = (double *) calloc(1, sizeof(double));

  nodes[0] = ffbpann_inputs ;     /* input nodes (baddr, opcode, rs,
rt) */
  nodes[1] = 10;
  nodes[2] = 1;

  net=create_ffbpann(layers,nodes);
  init_ffbpann(1, net);

/*  NormalizeSunspots();*/

  for (i=0; i<NUM_YEARS; i++) {
    Min = MIN(Min, Sunspots[i]);
    Max = MAX(Max, Sunspots[i]);
  }
```

```
  for(i=1;i<=ffbpann_inputs;i++)
  {
    net->inputbounds[i]->max = Max ;
    net->inputbounds[i]->min = Min ;
/*    printf("%i: max: %lf, min: %lf\n", i,net->inputbounds[i]-
>max,net->inputbounds[i]->min);
fflush(stdout);
*/
  }

    printf("Year    Sunspots        Prediction\n");
    test_error = 0;
    for(l=EVAL_LWB; l<=EVAL_UPB;l++)  /* the test data set */
    {
      for(i=0; i<net->inputlayer->nodes; i++) {
        inputs[i] = Sunspots[l-N+i] ;
      }
      target[0] = Sunspots[l] ;

      calc_ffbpann(inputs,net,out);
      for(i=1; i<=net->outputlayer->nodes; i++) {
        output = net->outputlayer->outputs[1];
      }

    printf("%d       %0.3lf          %0.3lf\n", l+FIRST_YEAR,
Sunspots[l],output);
    }
/*
    printf("\nAve. Train Error: %0.3lf - Ave. Test Error: %0.3lf\n",
train_error/(TRAIN_UPB-TRAIN_LWB), test_error/(TEST_UPB-
TEST_LWB));
printf("test_error: %lf train_error: %lf min_error:
%lf\n",test_error, train_error, min_error);
*/
    fflush(stdout);

  free(out);
}
```

sunspots.h

```
#include <stdlib.h>

#define MIN_double      -HUGE_VAL
```

```
#define MAX_double      +HUGE_VAL
#define MIN(x,y)      ((x)<(y) ? (x) : (y))
#define MAX(x,y)      ((x)>(y) ? (x) : (y))

#define LO          0.1
#define HI          0.9

#define sqr(x)       ((x)*(x))

#define NUM_LAYERS    3
#define N           30
#define M           1
int             Units[NUM_LAYERS] = {N, 10, M};

#define FIRST_YEAR    1700
#define NUM_YEARS     280

#define TRAIN_LWB     (N)
#define TRAIN_UPB     (179)
#define TRAIN_YEARS   (TRAIN_UPB - TRAIN_LWB + 1)
#define TEST_LWB      (180)
#define TEST_UPB      (259)
#define TEST_YEARS    (TEST_UPB - TEST_LWB + 1)
#define EVAL_LWB      (260)
#define EVAL_UPB      (NUM_YEARS - 1)
#define EVAL_YEARS    (EVAL_UPB - EVAL_LWB + 1)

double          Sunspots_[NUM_YEARS];
double          Sunspots [NUM_YEARS] = {

                0.0262,  0.0575,  0.0837,  0.1203,  0.1883,  0.3033,
                0.1517,  0.1046,  0.0523,  0.0418,  0.0157,  0.0000,
                0.0000,  0.0105,  0.0575,  0.1412,  0.2458,  0.3295,
                0.3138,  0.2040,  0.1464,  0.1360,  0.1151,  0.0575,
                0.1098,  0.2092,  0.4079,  0.6381,  0.5387,  0.3818,
                0.2458,  0.1831,  0.0575,  0.0262,  0.0837,  0.1778,
                0.3661,  0.4236,  0.5805,  0.5282,  0.3818,  0.2092,
                0.1046,  0.0837,  0.0262,  0.0575,  0.1151,  0.2092,
                0.3138,  0.4231,  0.4362,  0.2495,  0.2500,  0.1606,
                0.0638,  0.0502,  0.0534,  0.1700,  0.2489,  0.2824,
                0.3290,  0.4493,  0.3201,  0.2359,  0.1904,  0.1093,
                0.0596,  0.1977,  0.3651,  0.5549,  0.5272,  0.4268,
                0.3478,  0.1820,  0.1600,  0.0366,  0.1036,  0.4838,
                0.8075,  0.6585,  0.4435,  0.3562,  0.2014,  0.1192,
                0.0534,  0.1260,  0.4336,  0.6904,  0.6846,  0.6177,
                0.4702,  0.3483,  0.3138,  0.2453,  0.2144,  0.1114,
```

```
                0.0837,  0.0335,  0.0214,  0.0356,  0.0758,  0.1778,
                0.2354,  0.2254,  0.2484,  0.2207,  0.1470,  0.0528,
                0.0424,  0.0131,  0.0000,  0.0073,  0.0262,  0.0638,
                0.0727,  0.1851,  0.2395,  0.2150,  0.1574,  0.1250,
                0.0816,  0.0345,  0.0209,  0.0094,  0.0445,  0.0868,
                0.1898,  0.2594,  0.3358,  0.3504,  0.3708,  0.2500,
                0.1438,  0.0445,  0.0690,  0.2976,  0.6354,  0.7233,
                0.5397,  0.4482,  0.3379,  0.1919,  0.1266,  0.0560,
                0.0785,  0.2097,  0.3216,  0.5152,  0.6522,  0.5036,
                0.3483,  0.3373,  0.2829,  0.2040,  0.1077,  0.0350,
                0.0225,  0.1187,  0.2866,  0.4906,  0.5010,  0.4038,
                0.3091,  0.2301,  0.2458,  0.1595,  0.0853,  0.0382,
                0.1966,  0.3870,  0.7270,  0.5816,  0.5314,  0.3462,
                0.2338,  0.0889,  0.0591,  0.0649,  0.0178,  0.0314,
                0.1689,  0.2840,  0.3122,  0.3332,  0.3321,  0.2730,
                0.1328,  0.0685,  0.0356,  0.0330,  0.0371,  0.1862,
                0.3818,  0.4451,  0.4079,  0.3347,  0.2186,  0.1370,
                0.1396,  0.0633,  0.0497,  0.0141,  0.0262,  0.1276,
                0.2197,  0.3321,  0.2814,  0.3243,  0.2537,  0.2296,
                0.0973,  0.0298,  0.0188,  0.0073,  0.0502,  0.2479,
                0.2986,  0.5434,  0.4215,  0.3326,  0.1966,  0.1365,
                0.0743,  0.0303,  0.0873,  0.2317,  0.3342,  0.3609,
                0.4069,  0.3394,  0.1867,  0.1109,  0.0581,  0.0298,
                0.0455,  0.1888,  0.4168,  0.5983,  0.5732,  0.4644,
                0.3546,  0.2484,  0.1600,  0.0853,  0.0502,  0.1736,
                0.4843,  0.7929,  0.7128,  0.7045,  0.4388,  0.3630,
                0.1647,  0.0727,  0.0230,  0.1987,  0.7411,  0.9947,
                0.9665,  0.8316,  0.5873,  0.2819,  0.1961,  0.1459,
                0.0534,  0.0790,  0.2458,  0.4906,  0.5539,  0.5518,
                0.5465,  0.3483,  0.3603,  0.1987,  0.1804,  0.0811,
                0.0659,  0.1428,  0.4838,  0.8127

               };

double          Mean;
double          TrainError;
double          TrainErrorPredictingMean;
double          TestError;
double          TestErrorPredictingMean;

void NormalizeSunspots()
{
  int  Year;
  double Min, Max;

  Min = MAX_double;
```

```
  Max = MIN_double;
  for (Year=0; Year<NUM_YEARS; Year++) {
    Min = MIN(Min, Sunspots[Year]);
    Max = MAX(Max, Sunspots[Year]);
  }
  Mean = 0;
  for (Year=0; Year<NUM_YEARS; Year++) {
    Sunspots_[Year] =
    Sunspots [Year] = ((Sunspots[Year]-Min) / (Max-Min)) * (HI-
LO) + LO;
    Mean += Sunspots[Year] / NUM_YEARS;
  }
}
```

APPENDIX C

Simplescalar Code

bpred.h

```
/*
 * bpred.h - branch predictor interfaces
 *
 * This file is a part of the SimpleScalar tool suite written by
 * Todd M. Austin as a part of the Multiscalar Research Project.
 *
 * The tool suite is currently maintained by Doug Burger and Todd
M. Austin.
 *
 * Copyright (C) 1994, 1995, 1996, 1997 by Todd M. Austin
 *
 * This source file is distributed "as is" in the hope that it will be
 * useful.  The tool set comes with no warranty, and no author or
 * distributor accepts any responsibility for the consequences of its
 * use.
 *
 * Everyone is granted permission to copy, modify and redistribute
 * this tool set under the following conditions:
 *
 *    This source code is distributed for non-commercial use only.
 *    Please contact the maintainer for restrictions applying to
 *    commercial use.
 *
 *    Permission is granted to anyone to make or distribute copies
 *    of this source code, either as received or modified, in any
 *    medium, provided that all copyright notices, permission and
 *    nonwarranty notices are preserved, and that the distributor
 *    grants the recipient permission for further redistribution as
 *    permitted by this document.
 *
 *    Permission is granted to distribute this file in compiled
 *    or executable form under the same conditions that apply for
 *    source code, provided that either:
 *
 *    A. it is accompanied by the corresponding machine-readable
 *       source code,
 *    B. it is accompanied by a written offer, with no time limit,
 *       to give anyone a machine-readable copy of the
corresponding
 *       source code in return for reimbursement of the cost of
```

```
 *      distribution.  This written offer must permit verbatim
 *      duplication by anyone, or
 *    C. it is distributed by someone who received only the
 *      executable form, and is accompanied by a copy of the
 *      written offer of source code that they received concurrently.
 *
 * In other words, you are welcome to use, share and improve this
 * source file.  You are forbidden to forbid anyone else to use,
share
 * and improve what you give them.
 *
 * INTERNET: dburger@cs.wisc.edu
 * US Mail:  1210 W. Dayton Street, Madison, WI 53706
 *
 * $Id: bpred.h,v 1.1.1.1 1997/05/22 18:04:05 aklauser Exp $
 *
 * $Log: bpred.h,v $
 * Revision 1.1.1.1  1997/05/22 18:04:05  aklauser
 *
 * Revision 1.8  1997/05/01 20:23:06  skadron
 * BTB bug fixes; jumps no longer update direction state; non-
taken
 *    branches non longer update BTB
 *
 * Revision 1.7  1997/05/01 00:05:51  skadron
 * Separated BTB from direction-predictor
 *
 * Revision 1.6  1997/04/29  23:50:44  skadron
 * Added r31 info to distinguish between return-JRs and other JRs
for bpred
 *
 * Revision 1.5  1997/04/29  22:53:10  skadron
 * Hopefully bpred is now right: bpred now allocates entries only
for
 *    branches; on a BTB miss it still returns a direction; and it uses
a
 *    return-address stack.  Returns are not yet distinguished among
JR's
 *
 * Revision 1.4  1997/04/28  17:37:09  skadron
 * Bpred now allocates entries for any instruction instead of only
 *    branches; also added return-address stack
 *
 * Revision 1.3  1997/04/24  16:57:27  skadron
 * Bpred used to return no prediction if the indexing branch didn't
match
```

```
 *    in the BTB.  Now it can predict a direction even on a BTB
address
 *    conflict
 *
 * Revision 1.2  1997/03/25 16:17:21  skadron
 * Added function called after priming
 *
 * Revision 1.1  1997/02/16  22:23:54  skadron
 * Initial revision
 *
 *
 */

#ifndef BPRED_H
#define BPRED_H

#define dassert(a) assert(a)

#include <stdio.h>
#include "misc.h"
#include "ss.h"
#include "stats.h"
#include "ffbpann.h"

/*
 * This module implements a number of branch predictor
mechanisms.  The
 * following predictors are supported:
 *
 *        BPred2Level:  two level adaptive branch predictor
 *
 *                It can simulate many prediction mechanisms that
have up to
 *                two levels of tables. Parameters are:
 *                    N   # entries in first level (# of shift register(s))
 *                    W   width of shift register(s)
 *                    M   # entries in 2nd level (# of counters, or other
FSM)
 *                One BTB entry per level-2 counter.
 *
 *                Configurations:   N, W, M
 *
 *                   counter based: 1, 0, M
 *
 *                   GAg         : 1, W, 2^W
 *                   GAp         : 1, W, M (M > 2^W)
```

```
 *                  PAg         : N, W, 2^W
 *                  PAp         : N, W, M (M == 2^(N+W))
 *
 *        BPred2bit:  a simple direct mapped bimodal predictor
 *
 *                This predictor has a table of two bit saturating
counters.
 *                Where counter states 0 & 1 are predict not taken
and
 *                counter states 2 & 3 are predict taken, the per-
branch counters
 *                are incremented on taken branches and decremented
on
 *                no taken branches.  One BTB entry per counter.
 *
 *        BPredTaken:  static predict branch taken
 *
 *        BPredNotTaken:  static predict branch not taken
 *
 *        BPredFBPANN: feed-forward backpropagation Artificial
Neural Network
 *         It uses the same scheme as the BPredComb predictor, but
uses a
 *         ffbpann instead of the 2 level predictor
 *
 */

/* branch predictor types */
enum bpred_class {
  BPredComb,              /* combined predictor (McFarling) */
  BPred2Level,                    /* 2-level correlating pred
w/2-bit counters */
  BPred2bit,                  /* 2-bit saturating cntr pred (dir
mapped) */
  BPredTaken,                 /* static predict taken */
  BPredNotTaken,              /* static predict not taken */
  BPredFFBPANN,                   /* feed-forward backprop
ANN */
  BPred_NUM
};

/* an entry in a BTB */
struct bpred_btb_ent {
  SS_ADDR_TYPE addr;              /* address of branch being
tracked */
  enum ss_opcode op;          /* opcode of branch corresp. to addr
```

```
*/
  SS_ADDR_TYPE target;              /* last destination of branch
when taken */
  struct bpred_btb_ent *prev, *next; /* lru chaining pointers */
};

/* direction predictor def */
struct bpred_dir {
  enum bpred_class class;       /* type of predictor */
  union {
    struct {
      unsigned int size;  /* number of entries in direct-mapped table
*/
      unsigned char *table;       /* prediction state table */
    } bimod;
    struct {
      int l1size;             /* level-1 size, number of history regs */
      int l2size;             /* level-2 size, number of pred states */
      int shift_width;              /* amount of history in level-1 shift
regs */
      int xor;                      /* history xor address flag */
      int *shiftregs;               /* level-1 history table */
      unsigned char *l2table;     /* level-2 prediction state table */
    } two;
    struct {
      ffbpann *net;
    } ffbpann;
  } config;
};

/* branch predictor def */
struct bpred {
  enum bpred_class class;       /* type of predictor */
  struct {
    struct bpred_dir *bimod;     /* first direction predictor */
    struct bpred_dir *twolev;    /* second direction predictor */
    struct bpred_dir *meta;      /* meta predictor */
    struct bpred_dir *ffbpann;   /* ffbpann direction predictor */
  } dirpred;

  struct {
    int sets;                     /* num BTB sets */
    int assoc;                    /* BTB associativity */
    struct bpred_btb_ent *btb_data; /* BTB addr-prediction table */
  } btb;
```

```
  struct {
    int size;                   /* return-address stack size */
    int tos;                    /* top-of-stack */
    struct bpred_btb_ent *stack; /* return-address stack */
  } retstack;

  /* stats */
  SS_COUNTER_TYPE addr_hits;    /* num correct addr-
predictions */
  SS_COUNTER_TYPE dir_hits;     /* num correct dir-predictions
(incl addr) */
  SS_COUNTER_TYPE used_bimod;          /* num bimodal
predictions used (BPredComb) */
  SS_COUNTER_TYPE used_2lev;  /* num 2-level predictions
used (BPredComb) */
  SS_COUNTER_TYPE used_ffbpann; /* num ffbpann predictions
used (BPredFFBPANN)*/
  SS_COUNTER_TYPE jr_hits;      /* num correct addr-
predictions for JR's */
  SS_COUNTER_TYPE jr_seen;      /* num JR's seen */
  SS_COUNTER_TYPE misses;       /* num incorrect predictions
*/

  SS_COUNTER_TYPE lookups;      /* num lookups */
  SS_COUNTER_TYPE retstack_pops;   /* number of times a
value was popped */
  SS_COUNTER_TYPE retstack_pushes; /* number of times a
value was pushed */
};

/* branch predictor update information */
struct bpred_update {
  char *pdir1;        /* direction-1 predictor counter */
  char *pdir2;        /* direction-2 predictor counter */
  char *pmeta;         /* meta predictor counter */
  struct {            /* predicted directions */
    uint bimod  : 1;    /* bimodal predictor */
    uint twolev : 1;    /* 2-level predictor */
    uint ffbpann : 1;   /* ffbpann predictor */
    uint meta   : 1;    /* meta predictor (0..bimod / 1..2lev,ffbpann) */
  } dir;
};

/* create a branch predictor */
struct bpred *                  /* branch predictory instance
*/
```

```
bpred_create(enum bpred_class class,          /* type of predictor to
create */
             unsigned int bimod_size,   /* bimod table size */
             unsigned int l1size,/* level-1 table size */
             unsigned int l2size,/* level-2 table size */
             unsigned int meta_size,     /* meta predictor table size */
             unsigned int shift_width,   /* history register width */
             unsigned int xor,           /* history xor address flag */
             unsigned int btb_sets,      /* number of sets in BTB */
             unsigned int btb_assoc,     /* BTB associativity */
             unsigned int retstack_size);/* num entries in ret-addr
stack */

/* create a branch direction predictor */
struct bpred_dir *              /* branch direction predictor instance
*/
bpred_dir_create (
  enum bpred_class class,       /* type of predictor to create */
  unsigned int l1size,          /* level-1 table size */
  unsigned int l2size,          /* level-2 table size (if relevant) */
  unsigned int shift_width,     /* history register width */
  unsigned int  xor);    /* history xor address flag */

/* print branch predictor configuration */
void
bpred_config(struct bpred *pred,      /* branch predictor instance
*/
             FILE *stream);           /* output stream */

/* print predictor stats */
void
bpred_stats(struct bpred *pred,               /* branch predictor
instance */
            FILE *stream);            /* output stream */

/* register branch predictor stats */
void
bpred_reg_stats(struct bpred *pred,   /* branch predictor instance
*/
                struct stat_sdb_t *sdb);/* stats database */

/* reset stats after priming, if appropriate */
void bpred_after_priming(struct bpred *bpred);

/* probe a predictor for a next fetch address, the predictor is probed
   with branch address BADDR, the branch target is BTARGET
```

```
(used for
   static predictors), and OP is the instruction opcode (used to
simulate
   predecode bits; a pointer to the predictor state entry (or null for
jumps)
   is returned in *DIR_UPDATE_PTR (used for updating predictor
state),
   and the non-speculative top-of-stack is returned in
stack_recover_idx
   (used for recovering ret-addr stack after mis-predict).  */
SS_ADDR_TYPE                                    /* predicted branch
target addr */
bpred_lookup(struct bpred *pred,      /* branch predictor instance
*/
             SS_ADDR_TYPE baddr,  /* branch address */
             SS_ADDR_TYPE btarget,/* branch target if taken */
             enum ss_opcode op,                 /* opcode of
instruction */
             int r31p,                          /* is this using r31? */
             struct bpred_update *dir_update_ptr,/* predictor state
pointer */
             int *stack_recover_idx,     /* Non-speculative top-of-
stack;
                                          * used on mispredict
recovery */
           SS_INST_TYPE inst);       /* instruction used for
FFBPANN */

/* Speculative execution can corrupt the ret-addr stack.  So for
each
 * lookup we return the top-of-stack (TOS) at that point; a
mispredicted
 * branch, as part of its recovery, restores the TOS using this value
--
 * hopefully this uncorrupts the stack. */
void
bpred_recover(struct bpred *pred,     /* branch predictor instance
*/
              SS_ADDR_TYPE baddr, /* branch address */
              int stack_recover_idx);     /* Non-speculative top-of-
stack;
                                           * used on mispredict
recovery */

/* update the branch predictor, only useful for stateful predictors;
updates
```

```
  entry for instruction type OP at address BADDR.  BTB only gets
updated
  for branches which are taken.  Inst was determined to jump to
  address BTARGET and was taken if TAKEN is non-zero.
Predictor
  statistics are updated with result of prediction, indicated by
CORRECT and
  PRED_TAKEN, predictor state to be updated is indicated by
*DIR_UPDATE_PTR
  (may be NULL for jumps, which shouldn't modify state bits).
Note if
  bpred_update is done speculatively, branch-prediction may get
polluted. */
void
bpred_update(struct bpred *pred,      /* branch predictor instance
*/
             SS_ADDR_TYPE baddr,  /* branch address */
             SS_ADDR_TYPE btarget,/* resolved branch target */
             int taken,                   /* non-zero if branch was
taken */
             int pred_taken,              /* non-zero if branch was
pred taken */
             int correct,           /* was earlier prediction correct? */
             enum ss_opcode op,                /* opcode of
instruction */
             int r31p,                    /* is this using r31? */
             struct bpred_update *dir_update_ptr, /* predictor state
pointer */
           SS_INST_TYPE inst);       /* instruction for FFBPANN
update */



#ifdef foo0
/* OBSOLETE */
/* dump branch predictor state (for debug) */
void
bpred_dump(struct bpred *pred,               /* branch predictor
instance */
           FILE *stream);                /* output stream */
#endif

#endif /* BPRED_H */
```

bpred.c

```
/*
 * bpred.c - branch predictor routines
 *
 * This file is a part of the SimpleScalar tool suite written by
 * Todd M. Austin as a part of the Multiscalar Research Project.
 *
 * The tool suite is currently maintained by Doug Burger and Todd
M. Austin.
 *
 * Copyright (C) 1994, 1995, 1996, 1997 by Todd M. Austin
 *
 * This source file is distributed "as is" in the hope that it will be
 * useful.  The tool set comes with no warranty, and no author or
 * distributor accepts any responsibility for the consequences of its
 * use.
 *
 * Everyone is granted permission to copy, modify and redistribute
 * this tool set under the following conditions:
 *
 *    This source code is distributed for non-commercial use only.
 *    Please contact the maintainer for restrictions applying to
 *    commercial use.
 *
 *    Permission is granted to anyone to make or distribute copies
 *    of this source code, either as received or modified, in any
 *    medium, provided that all copyright notices, permission and
 *    nonwarranty notices are preserved, and that the distributor
 *    grants the recipient permission for further redistribution as
 *    permitted by this document.
 *
 *    Permission is granted to distribute this file in compiled
 *    or executable form under the same conditions that apply for
 *    source code, provided that either:
 *
 *    A. it is accompanied by the corresponding machine-readable
 *       source code,
 *    B. it is accompanied by a written offer, with no time limit,
 *       to give anyone a machine-readable copy of the
corresponding
 *       source code in return for reimbursement of the cost of
 *       distribution.  This written offer must permit verbatim
 *       duplication by anyone, or
 *    C. it is distributed by someone who received only the
 *       executable form, and is accompanied by a copy of the
 *       written offer of source code that they received concurrently.
 *
```

```
 * In other words, you are welcome to use, share and improve this
 * source file.  You are forbidden to forbid anyone else to use,
share
 * and improve what you give them.
 *
 * INTERNET: dburger@cs.wisc.edu
 * US Mail:  1210 W. Dayton Street, Madison, WI 53706
 *
 * $Id: bpred.c,v 1.1.1.1 1997/05/22 00:33:18 aklauser Exp $
 *
 * $Log: bpred.c,v $
 * Revision 1.1.1.1  1997/05/22 00:33:18  aklauser
 *
 * Revision 1.11  1997/05/01 20:23:00  skadron
 * BTB bug fixes; jumps no longer update direction state; non-
taken
 *    branches non longer update BTB
 *
 * Revision 1.10  1997/05/01 00:05:42  skadron
 * Separated BTB from direction-predictor
 *
 * Revision 1.9  1997/04/30  01:42:42  skadron
 * 1. Not aggressively returning the BTB target regardless of hit on
jump's,
 *    but instead returning just "taken" when it's a BTB miss yields
an
 *    apparent epsilon performance improvement for cc1 and perl.
 * 2. Bug fix: if no retstack, treat return's as any other jump
 *
 * Revision 1.8  1997/04/29  23:50:33  skadron
 * Added r31 info to distinguish between return-JRs and other JRs
for bpred
 *
 * Revision 1.7  1997/04/29  22:53:04  skadron
 * Hopefully bpred is now right: bpred now allocates entries only
for
 *    branches; on a BTB miss it still returns a direction; and it uses
a
 *    return-address stack.  Returns are not yet distinguished among
JR's
 *
 * Revision 1.6  1997/04/28  17:37:02  skadron
 * Bpred now allocates entries for any instruction instead of only
 *    branches; also added return-address stack
 *
 * Revision 1.5  1997/04/24  16:57:21  skadron
```

```
 * Bpred used to return no prediction if the indexing branch didn't
match
 *    in the BTB.  Now it can predict a direction even on a BTB
address
 *    conflict
 *
 * Revision 1.4  1997/03/27  16:31:52  skadron
 * Fixed bug: sim-outorder calls bpred_after_priming(), even if no
bpred
 *    exists.  Now we check for a null ptr.
 *
 * Revision 1.3  1997/03/25  16:16:33  skadron
 * Statistics now take account of priming: statistics report only
 *    post-prime info.
 *
 * Revision 1.2  1997/02/24  18:02:41  skadron
 * Fixed output format of a formula stat
 *
 * Revision 1.1  1997/02/16  22:23:54  skadron
 * Initial revision
 *
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <assert.h>

#include "misc.h"
#include "ss.h"
#include "bpred.h"

/* create a branch predictor */
struct bpred *                          /* branch predictory instance
*/
bpred_create(enum bpred_class class,          /* type of predictor to
create */
             unsigned int bimod_size,   /* bimod table size */
             unsigned int l1size,/* 2lev l1 table size */
             unsigned int l2size,/* 2lev l2 table size */
             unsigned int meta_size,    /* meta table size */
             unsigned int shift_width,  /* history register width */
             unsigned int xor,    /* history xor address flag */
             unsigned int btb_sets,     /* number of sets in BTB */
             unsigned int btb_assoc,    /* BTB associativity */
```

```
                unsigned int retstack_size) /* num entries in ret-addr
stack */
{
  struct bpred *pred;

  if (!(pred = calloc(1, sizeof(struct bpred))))
    fatal("out of virtual memory");

  pred->class = class;

  switch (class) {
  case BPredComb:
    /* bimodal component */
    pred->dirpred.bimod =
      bpred_dir_create(BPred2bit, bimod_size, 0, 0, 0);

    /* 2-level component */
    pred->dirpred.twolev =
      bpred_dir_create(BPred2Level, l1size, l2size, shift_width,
xor);

    /* metapredictor component */
    pred->dirpred.meta =
      bpred_dir_create(BPred2bit, meta_size, 0, 0, 0);

    break;

  case BPred2Level:
    pred->dirpred.twolev =
      bpred_dir_create(class, l1size, l2size, shift_width, xor);

    break;

  case BPredFFBPANN:
    /* ffbpann component */
    pred->dirpred.ffbpann =
      bpred_dir_create(BPredFFBPANN, l1size, l2size, shift_width,
xor);

    /* bimodal component */
    pred->dirpred.bimod =
      bpred_dir_create(BPred2bit, bimod_size, 0, 0, 0);


    /* metapredictor component */
    pred->dirpred.meta =
```

```
      bpred_dir_create(BPred2bit, meta_size, 0, 0, 0);

    break ;

  case BPred2bit:
    pred->dirpred.bimod =
      bpred_dir_create(class, bimod_size, 0, 0, 0);

  case BPredTaken:
  case BPredNotTaken:
    /* no other state */
    break;

  default:
    panic("bogus predictor class");
  }

  /* allocate ret-addr stack */
  switch (class) {
  case BPredComb:
  case BPred2Level:
  case BPredFFBPANN:
  case BPred2bit:
    {
      int i;

      /* allocate BTB */
      if (!btb_sets || (btb_sets & (btb_sets-1)) != 0)
          fatal("number of BTB sets must be non-zero and a power
of two");
      if (!btb_assoc || (btb_assoc & (btb_assoc-1)) != 0)
          fatal("BTB associativity must be non-zero and a power of
two");

      if (!(pred->btb.btb_data = calloc(btb_sets * btb_assoc,
                                        sizeof(struct
bpred_btb_ent))))
          fatal("cannot allocate BTB");

      pred->btb.sets = btb_sets;
      pred->btb.assoc = btb_assoc;

      if (pred->btb.assoc > 1)
          for (i=0; i < (pred->btb.assoc*pred->btb.sets); i++)
            {
              if (i % pred->btb.assoc != pred->btb.assoc - 1)
```

```
              pred->btb.btb_data[i].next = &pred->btb.btb_data[i+1];
            else
              pred->btb.btb_data[i].next = NULL;

            if (i % pred->btb.assoc != pred->btb.assoc - 1)
              pred->btb.btb_data[i+1].prev = &pred->btb.btb_data[i];
          }

      /* allocate retstack */
      if ((retstack_size & (retstack_size-1)) != 0)
          fatal("Return-address-stack size must be zero or a power of
two");

      pred->retstack.size = retstack_size;
      if (retstack_size)
          if (!(pred->retstack.stack = calloc(retstack_size,
                                              sizeof(struct
bpred_btb_ent))))
            fatal("cannot allocate return-address-stack");
      pred->retstack.tos = retstack_size - 1;

      break;
    }

  case BPredTaken:
  case BPredNotTaken:
    /* no other state */
    break;

  default:
    panic("bogus predictor class");
  }

  return pred;
}

/* create a branch direction predictor */
struct bpred_dir *              /* branch direction predictor instance
*/
bpred_dir_create (
  enum bpred_class class,       /* type of predictor to create */
  unsigned int l1size,          /* level-1 table size */
  unsigned int l2size,          /* level-2 table size (if relevant) */
  unsigned int shift_width,     /* history register width */
  unsigned int xor)       /* history xor address flag */
{
```

```
  struct bpred_dir *pred_dir;
  unsigned int cnt;
  int flipflop;
  int layers = 1;
  int i;
  int nodes[6];

  if (!(pred_dir = calloc(1, sizeof(struct bpred_dir))))
    fatal("out of virtual memory");

  pred_dir->class = class;

  cnt = -1;
  switch (class) {
  case BPred2Level:
    {
      if (!l1size || (l1size & (l1size-1)) != 0)
          fatal("level-1 size, `%d', must be non-zero and a power of
two",
                l1size);
      pred_dir->config.two.l1size = l1size;

      if (!l2size || (l2size & (l2size-1)) != 0)
          fatal("level-2 size, `%d', must be non-zero and a power of
two",
                l2size);
      pred_dir->config.two.l2size = l2size;

      if (!shift_width || shift_width > 30)
          fatal("shift register width, `%d', must be non-zero and
positive",
                shift_width);
      pred_dir->config.two.shift_width = shift_width;

      pred_dir->config.two.xor = xor;
      pred_dir->config.two.shiftregs = calloc(l1size, sizeof(int));
      if (!pred_dir->config.two.shiftregs)
          fatal("cannot allocate shift register table");

      pred_dir->config.two.l2table = calloc(l2size, sizeof(unsigned
char));
      if (!pred_dir->config.two.l2table)
          fatal("cannot allocate second level table");

      /* initialize counters to weakly this-or-that */
      flipflop = 1;
```

```
      for (cnt = 0; cnt < l2size; cnt++)
           {
            pred_dir->config.two.l2table[cnt] = flipflop;
            flipflop = 3 - flipflop;
           }

      break;
    }

  case BPred2bit:
    if (!l1size || (l1size & (l1size-1)) != 0)
      fatal("2bit table size, `%d', must be non-zero and a power of
two",
              l1size);
    pred_dir->config.bimod.size = l1size;
    if (!(pred_dir->config.bimod.table =
            calloc(l1size, sizeof(unsigned char))))
      fatal("cannot allocate 2bit storage");
    /* initialize counters to weakly this-or-that */
    flipflop = 1;
    for (cnt = 0; cnt < l1size; cnt++)
      {
           pred_dir->config.bimod.table[cnt] = flipflop;
           flipflop = 3 - flipflop;
      }

    break;

  case BPredFFBPANN:
    /* create ANN */
    nodes[0] = ffbpann_inputs ;     /* input nodes (baddr, opcode, rs,
rt) */
    nodes[1] = l1size;
    nodes[2] = l2size;
    nodes[3] = shift_width;
    nodes[4] = xor ;
    for(i=1;i<5;i++){
      if(nodes[i] != 0){   /* figure out how many "0" entries */
        layers++;
      }
    }
    nodes[layers] = 1;  /* output layer */
    layers++ ;          /* at least two layers, input and output */
    pred_dir->config.ffbpann.net=create_ffbpann(layers,nodes);
    if (!(pred_dir->config.ffbpann.table = calloc(2, sizeof(unsigned
char))))
```

```
      fatal("cannot allocate space for ffbpann");
    flipflop = 1 ;
    pred_dir->config.ffbpann.table[0] = flipflop;
    flipflop++;
    pred_dir->config.ffbpann.table[1] = flipflop;
/*    init_ffbpann(0,pred_dir->config.ffbpann.net);
*/  init_ffbpann(1,pred_dir->config.ffbpann.net);
    break;

  case BPredTaken:
  case BPredNotTaken:
    /* no other state */
    break;

  default:
    panic("bogus branch direction predictor class");
  }

  return pred_dir;
}

/* print branch direction predictor configuration */
void
bpred_dir_config(
  struct bpred_dir *pred_dir,   /* branch direction predictor instance
*/
  char name[],                  /* predictor name */
  FILE *stream)                     /* output stream */
{
  int i;
  switch (pred_dir->class) {
  case BPred2Level:
    fprintf(stream,
      "pred_dir: %s: 2-lvl: %d l1-sz, %d bits/ent, %s xor, %d l2-sz,
direct-mapped\n",
      name, pred_dir->config.two.l1size, pred_dir-
>config.two.shift_width,
      pred_dir->config.two.xor ? "" : "no", pred_dir-
>config.two.l2size);
    break;

  case BPred2bit:
    fprintf(stream, "pred_dir: %s: 2-bit: %d entries, direct-
mapped\n",
      name, pred_dir->config.bimod.size);
    break;
```

```
  case BPredFFBPANN:
    fprintf(stream, "pred_dir: %s: layers: %i, nodes:", name,
pred_dir->config.ffbpann.net->num_layers);
    for(i=0;i<pred_dir->config.ffbpann.net->num_layers;i++){
      fprintf(stream, " %i", pred_dir->config.ffbpann.net->layers[i]-
>nodes);
    }
    fprintf(stream, "\n");
    break;

  case BPredTaken:
    fprintf(stream, "pred_dir: %s: predict taken\n", name);
    break;

  case BPredNotTaken:
    fprintf(stream, "pred_dir: %s: predict not taken\n", name);
    break;

  default:
    panic("bogus branch direction predictor class");
  }
}

/* print branch predictor configuration */
void
bpred_config(struct bpred *pred,       /* branch predictor instance
*/
             FILE *stream)             /* output stream */
{
  switch (pred->class) {
  case BPredComb:
    bpred_dir_config (pred->dirpred.bimod, "bimod", stream);
    bpred_dir_config (pred->dirpred.twolev, "2lev", stream);
    bpred_dir_config (pred->dirpred.meta, "meta", stream);
    fprintf(stream, "btb: %d sets x %d associativity",
            pred->btb.sets, pred->btb.assoc);
    fprintf(stream, "ret_stack: %d entries", pred->retstack.size);
    break;

  case BPredFFBPANN:
    bpred_dir_config (pred->dirpred.bimod, "bimod", stream);
    bpred_dir_config (pred->dirpred.ffbpann, "ffbpann", stream);
    bpred_dir_config (pred->dirpred.meta, "meta", stream);
    fprintf(stream, "btb: %d sets x %d associativity",
          pred->btb.sets, pred->btb.assoc);
```

```
    fprintf(stream, "ret_stack: %d entries", pred->retstack.size);
    break;

  case BPred2Level:
    bpred_dir_config (pred->dirpred.twolev, "2lev", stream);
    fprintf(stream, "btb: %d sets x %d associativity",
            pred->btb.sets, pred->btb.assoc);
    fprintf(stream, "ret_stack: %d entries", pred->retstack.size);
    break;

  case BPred2bit:
    bpred_dir_config (pred->dirpred.bimod, "bimod", stream);
    fprintf(stream, "btb: %d sets x %d associativity",
            pred->btb.sets, pred->btb.assoc);
    fprintf(stream, "ret_stack: %d entries", pred->retstack.size);
    break;

  case BPredTaken:
    bpred_dir_config (pred->dirpred.bimod, "taken", stream);
    break;
  case BPredNotTaken:
    bpred_dir_config (pred->dirpred.bimod, "nottaken", stream);
    break;

  default:
    panic("bogus branch predictor class");
  }
}

/* print predictor stats */
void
bpred_stats(struct bpred *pred,                 /* branch predictor
instance */
            FILE *stream)               /* output stream */
{
  fprintf(stream, "pred: addr-prediction rate = %f\n",
          (double)pred->addr_hits/(double)(pred->addr_hits+pred-
>misses));
  fprintf(stream, "pred: dir-prediction rate = %f\n",
          (double)pred->dir_hits/(double)(pred->dir_hits+pred-
>misses));
}

/* register branch predictor stats */
void
bpred_reg_stats(struct bpred *pred,   /* branch predictor instance
```

```
*/
                    struct stat_sdb_t *sdb)/* stats database */
{
  char buf[512], buf1[512], *name;

  /* get a name for this predictor */
  switch (pred->class)
    {
    case BPredComb:
      name = "bpred_comb";
      break;
    case BPred2Level:
      name = "bpred_2lev";
      break;
    case BPred2bit:
      name = "bpred_bimod";
      break;
    case BPredTaken:
      name = "bpred_taken";
      break;
    case BPredNotTaken:
      name = "bpred_nottaken";
      break;
    case BPredFFBPANN:
      name = "bpred_ffbpann" ;
      break;
    default:
      panic("bogus branch predictor class");
    }

  sprintf(buf, "%s.lookups", name);
  stat_reg_counter(sdb, buf, "total number of bpred lookups",
                   &pred->lookups, 0, NULL);
  sprintf(buf, "%s.updates", name);
  sprintf(buf1, "%s.dir_hits + %s.misses", name, name);
  stat_reg_formula(sdb, buf, "total number of updates", buf1,
"%11.0f");
  sprintf(buf, "%s.addr_hits", name);
  stat_reg_counter(sdb, buf, "total number of address-predicted
hits",
                   &pred->addr_hits, 0, NULL);
  sprintf(buf, "%s.dir_hits", name);
  stat_reg_counter(sdb, buf,
                   "total number of direction-predicted hits "
                   "(includes addr-hits)",
                   &pred->dir_hits, 0, NULL);
```

```
  if (pred->class == BPredComb)
    {
      sprintf(buf, "%s.used_bimod", name);
      stat_reg_counter(sdb, buf,
                       "total number of bimodal predictions used",
                       &pred->used_bimod, 0, NULL);
      sprintf(buf, "%s.used_2lev", name);
      stat_reg_counter(sdb, buf,
                       "total number of 2-level predictions used",
                       &pred->used_2lev, 0, NULL);
    }
  if (pred->class == BPredFFBPANN)
    {
      sprintf(buf, "%s.used_bimod", name);
      stat_reg_counter(sdb, buf,
                  "total number of bimodal predictions used",
                   &pred->used_bimod, 0, NULL);
      sprintf(buf, "%s.used_ffbpann", name);
      stat_reg_counter(sdb, buf,
                  "total number of FFBPANN predictions used",
                  &pred->used_ffbpann, 0, NULL);
      dump_weights(pred->dirpred.ffbpann->config.ffbpann.net);
    }
  sprintf(buf, "%s.misses", name);
  stat_reg_counter(sdb, buf, "total number of misses", &pred-
>misses, 0, NULL);
  sprintf(buf, "%s.jr_hits", name);
  stat_reg_counter(sdb, buf,
                   "total number of address-predicted hits for JR's",
                   &pred->jr_hits, 0, NULL);
  sprintf(buf, "%s.jr_seen", name);
  stat_reg_counter(sdb, buf,
                   "total number of JR's seen",
                   &pred->jr_seen, 0, NULL);
  sprintf(buf, "%s.bpred_addr_rate", name);
  sprintf(buf1, "%s.addr_hits / %s.updates", name, name);
  stat_reg_formula(sdb, buf,
                   "branch address-prediction rate (i.e., addr-
hits/updates)",
                   buf1, "%9.4f");
  sprintf(buf, "%s.bpred_dir_rate", name);
  sprintf(buf1, "%s.dir_hits / %s.updates", name, name);
  stat_reg_formula(sdb, buf,
                  "branch direction-prediction rate (i.e., all-
hits/updates)",
                   buf1, "%9.4f");
```

```
  sprintf(buf, "%s.bpred_jr_rate", name);
  sprintf(buf1, "%s.jr_hits / %s.jr_seen", name, name);
  stat_reg_formula(sdb, buf,
                   "JR address-prediction rate (i.e., JR addr-hits/JRs
seen)",
                   buf1, "%9.4f");
  sprintf(buf, "%s.retstack_pushes", name);
  stat_reg_counter(sdb, buf,
                   "total number of address pushed onto ret-addr
stack",
                   &pred->retstack_pushes, 0, NULL);
  sprintf(buf, "%s.retstack_pops", name);
  stat_reg_counter(sdb, buf,
                   "total number of address popped off of ret-addr
stack",
                   &pred->retstack_pops, 0, NULL);

}

void bpred_after_priming(struct bpred *bpred)
{
  if (bpred == NULL)
    return;

  bpred->lookups = 0;
  bpred->addr_hits = 0;
  bpred->dir_hits = 0;
  bpred->used_bimod = 0;
  bpred->used_2lev = 0;
  bpred->jr_hits = 0;
  bpred->jr_seen = 0;
  bpred->misses = 0;
  bpred->retstack_pops = 0;
  bpred->retstack_pushes = 0;
}

#define BIMOD_HASH(PRED, ADDR)
                \
  ((((ADDR) >> 19) ^ ((ADDR) >> 3)) &                                  \
   ((PRED)->config.bimod.size-1))
    /* was: ((baddr >> 16) ^ baddr) & (pred->dirpred.bimod.size-1)
*/

/* predicts a branch direction */
char *                          /* pointer to counter */
bpred_dir_lookup (
```

```
  struct bpred_dir *pred_dir,   /* branch direction predictor instance
*/
  SS_ADDR_TYPE baddr,          /* branch address */
  SS_INST_TYPE inst)                  /* instruction for FFBPANN
*/
{
  unsigned char *p = NULL;
  int ok;
  double input[ffbpann_inputs];
  double *ptemp = NULL;

  /* Except for jumps, get a pointer to direction-prediction bits */
  switch (pred_dir->class) {
    case BPred2Level:
      {
          int l1index, l2index;

        /* traverse 2-level tables */
        l1index = (baddr >> 3) & (pred_dir->config.two.l1size - 1);
        l2index = pred_dir->config.two.shiftregs[l1index];
        if (pred_dir->config.two.xor)
            {
              l2index = l2index ^ (baddr>>3);
            }
          else
            {
              l2index =
                l2index | ((baddr>>3) << pred_dir-
>config.two.shift_width);
            }
        l2index = l2index & (pred_dir->config.two.l2size - 1);

        /* get a pointer to prediction state information */
        p = &pred_dir->config.two.l2table[l2index];
      }
      break;
    case BPredFFBPANN:
      if (!(ptemp = calloc(1, sizeof(double))))
        fatal("out of virtual memory");
      input[0] = (double) (baddr >> 3);
      input[1] = (double) SS_OPCODE(inst);
      input[2] = (double) RS;
      input[3] = (double) RT;
      ok = calc_ffbpann(input,pred_dir->config.ffbpann.net,ptemp);
      if(*ptemp < 0.5)
      {  /* nottaken */
```

```
        p = &pred_dir->config.ffbpann.table[0];
      }
      else
      { /* taken */
        p = &pred_dir->config.ffbpann.table[1];
      }
      free(ptemp);
      break;
    case BPred2bit:
      p = &pred_dir->config.bimod.table[BIMOD_HASH(pred_dir,
baddr)];
      break;
    case BPredTaken:
    case BPredNotTaken:
      break;
    default:
      panic("bogus branch direction predictor class");
    }

  return p;
}

/* probe a predictor for a next fetch address, the predictor is probed
   with branch address BADDR, the branch target is BTARGET
(used for
   static predictors), and OP is the instruction opcode (used to
simulate
   predecode bits; a pointer to the predictor state entry (or null for
jumps)
   is returned in *DIR_UPDATE_PTR (used for updating predictor
state),
   and the non-speculative top-of-stack is returned in
stack_recover_idx
   (used for recovering ret-addr stack after mis-predict).  */
SS_ADDR_TYPE                            /* predicted branch
target addr */
bpred_lookup(struct bpred *pred,      /* branch predictor instance
*/
             SS_ADDR_TYPE baddr,  /* branch address */
             SS_ADDR_TYPE btarget,/* branch target if taken */
             enum ss_opcode op,                 /* opcode of
instruction */
             int r31p,                      /* is this using r31? */
             struct bpred_update *dir_update_ptr,/* predictor state
pointer */
             int *stack_recover_idx,    /* Non-speculative top-of-
```

```
stack *
                                        * used on mispredict
recovery */
             SS_INST_TYPE inst)         /* instruction used for
FFBPANN */
{
  struct bpred_btb_ent *pbtb = NULL;
  int index, i;

  if (!dir_update_ptr)
    panic("no bpred update record");

  /* if this is not a branch, return not-taken */
  if (!(SS_OP_FLAGS(op) & F_CTRL))
    return 0;

  pred->lookups++;

  dir_update_ptr->pdir1 = NULL;
  dir_update_ptr->pdir2 = NULL;
  dir_update_ptr->pmeta = NULL;
  /* Except for jumps, get a pointer to direction-prediction bits */
  switch (pred->class) {
    case BPredComb:
      if ((SS_OP_FLAGS(op) & (F_CTRL|F_UNCOND)) !=
(F_CTRL|F_UNCOND))
          {
            char *bimod, *twolev, *meta;
            bimod = bpred_dir_lookup (pred->dirpred.bimod, baddr,
inst);
            twolev = bpred_dir_lookup (pred->dirpred.twolev, baddr,
inst);
            meta = bpred_dir_lookup (pred->dirpred.meta, baddr,
inst);
            dir_update_ptr->pmeta = meta;
            dir_update_ptr->dir.meta  = (*meta >= 2);
            dir_update_ptr->dir.bimod = (*bimod >= 2);
            dir_update_ptr->dir.twolev  = (*twolev >= 2);
            if (*meta >= 2)
              {
                dir_update_ptr->pdir1 = twolev;
                dir_update_ptr->pdir2 = bimod;
              }
            else
              {
                dir_update_ptr->pdir1 = bimod;
```

```
              dir_update_ptr->pdir2 = twolev;
            }
          }
      break;
    case BPred2Level:
      if ((SS_OP_FLAGS(op) & (F_CTRL|F_UNCOND)) !=
(F_CTRL|F_UNCOND))
          {
            dir_update_ptr->pdir1 =
              bpred_dir_lookup (pred->dirpred.twolev, baddr, inst);
          }
      break;
    case BPred2bit:
      if ((SS_OP_FLAGS(op) & (F_CTRL|F_UNCOND)) !=
(F_CTRL|F_UNCOND))
          {
            dir_update_ptr->pdir1 =
              bpred_dir_lookup (pred->dirpred.bimod, baddr, inst);
          }
      break;
    case BPredTaken:
      return btarget;
    case BPredNotTaken:
      if ((SS_OP_FLAGS(op) & (F_CTRL|F_UNCOND)) !=
(F_CTRL|F_UNCOND))
          {
            return baddr + sizeof(SS_INST_TYPE);
          }
      else
          {
            return btarget;
          }
    case BPredFFBPANN:
      if ((SS_OP_FLAGS(op) & (F_CTRL|F_UNCOND)) !=
(F_CTRL|F_UNCOND))
        {
          char *bimod, *ffbpann, *meta;
          bimod = bpred_dir_lookup (pred->dirpred.bimod, baddr,
inst);
          ffbpann = bpred_dir_lookup (pred->dirpred.ffbpann, baddr,
inst);
          meta = bpred_dir_lookup (pred->dirpred.meta, baddr, inst);
          dir_update_ptr->pmeta = meta;
          dir_update_ptr->dir.meta  = (*meta >= 2);
          dir_update_ptr->dir.bimod = (*bimod >= 2);
          dir_update_ptr->dir.ffbpann  = (*ffbpann >= 2);
```

```
        if (*meta >= 2)
          {
            dir_update_ptr->pdir1 = ffbpann;
            dir_update_ptr->pdir2 = bimod;
          }
        else
          {
            dir_update_ptr->pdir1 = bimod;
            dir_update_ptr->pdir2 = ffbpann;
          }
      }
    break;

  default:
    panic("bogus predictor class");
 }

 /*
  * We have a stateful predictor, and have gotten a pointer into the
  * direction predictor (except for jumps, for which the ptr is null)
  */

 /* record pre-pop TOS; if this branch is executed speculatively
  * and is squashed, we'll restore the TOS and hope the data
  * wasn't corrupted in the meantime. */
 if (pred->retstack.size)
   *stack_recover_idx = pred->retstack.tos;
 else
   *stack_recover_idx = 0;

 /* if this is a return, pop return-address stack */
 if (op == JR && r31p && pred->retstack.size)
   {
     SS_ADDR_TYPE target = pred->retstack.stack[pred-
>retstack.tos].target;
     pred->retstack.tos = (pred->retstack.tos + pred->retstack.size -
1)
                          % pred->retstack.size;
     pred->retstack_pops++;
     return target;
   }

 /* not a return. Get a pointer into the BTB */
 index = (baddr >> 3) & (pred->btb.sets - 1);

 if (pred->btb.assoc > 1)
```

```
    {
      index *= pred->btb.assoc;

      /* Now we know the set; look for a PC match */
      for (i = index; i < (index+pred->btb.assoc) ; i++)
          if (pred->btb.btb_data[i].addr == baddr)
            {
              /* match */
              pbtb = &pred->btb.btb_data[i];
              break;
            }
    }
  else
    {
      pbtb = &pred->btb.btb_data[index];
      if (pbtb->addr != baddr)
          pbtb = NULL;
    }

  /*
   * We now also have a pointer into the BTB for a hit, or NULL
otherwise
   */

  /* if this is a jump, ignore predicted direction; we know it's taken.
*/
  if ((SS_OP_FLAGS(op) & (F_CTRL|F_UNCOND)) ==
(F_CTRL|F_UNCOND))
    {
      return (pbtb ? pbtb->target : 1);
    }

  /* otherwise we have a conditional branch */
  if (pbtb == NULL)
    {
      /* BTB miss -- just return a predicted direction */
      return ((*(dir_update_ptr->pdir1) >= 2)
              ? /* taken */ 1
              : /* not taken */ 0);
    }
  else
    {
      /* BTB hit, so return target if it's a predicted-taken branch */
      return ((*(dir_update_ptr->pdir1) >= 2)
              ? /* taken */ pbtb->target
              : /* not taken */ 0);
```

```
    }
}

/* Speculative execution can corrupt the ret-addr stack.  So for
each
 * lookup we return the top-of-stack (TOS) at that point; a
mispredicted
 * branch, as part of its recovery, restores the TOS using this value
--
 * hopefully this uncorrupts the stack. */
void
bpred_recover(struct bpred *pred,      /* branch predictor instance
*/
              SS_ADDR_TYPE baddr, /* branch address */
              int stack_recover_idx)      /* Non-speculative top-of-
stack;
                                           * used on mispredict
recovery */
{
  if (pred == NULL)
    return;

  pred->retstack.tos = stack_recover_idx;
}

/* update the branch predictor, only useful for stateful predictors;
updates
   entry for instruction type OP at address BADDR.  BTB only gets
updated
   for branches which are taken.  Inst was determined to jump to
   address BTARGET and was taken if TAKEN is non-zero.
Predictor
   statistics are updated with result of prediction, indicated by
CORRECT and
   PRED_TAKEN, predictor state to be updated is indicated by
*DIR_UPDATE_PTR
   (may be NULL for jumps, which shouldn't modify state bits).
Note if
   bpred_update is done speculatively, branch-prediction may get
polluted. */
void
bpred_update(struct bpred *pred,       /* branch predictor instance
*/
             SS_ADDR_TYPE baddr,  /* branch address */
             SS_ADDR_TYPE btarget,/* resolved branch target */
             int taken,                    /* non-zero if branch was
```

```
taken */
                int pred_taken,                 /* non-zero if branch was
pred taken */
                int correct,            /* was earlier addr prediction ok? */
                enum ss_opcode op,                      /* opcode of
instruction */
                int r31p,                       /* is this using r31? */
                struct bpred_update *dir_update_ptr,/* predictor state
pointer */
           SS_INST_TYPE inst)        /* instruction for FFBPANN
update */
{
  struct bpred_btb_ent *pbtb = NULL;
  struct bpred_btb_ent *lruhead = NULL, *lruitem = NULL;
  int index, i;
  double input[ffbpann_inputs];
  double trgt[1];

/* FOR CREATING TRAINING DATA */
/*
  int target ;
  FILE *file;
  if(taken)
    target = 1;
  else
    target = 0 ;
  file = fopen("train.data", "a");
  fprintf(file,"%u %u %u %u %u\n", baddr, SS_OPCODE(inst),
RS, RT, target);
  fclose(file);
*/
/* DON'T USE FOR NORMAL RUNS */

  /* don't change bpred state for non-branch instructions or if this
   * is a stateless predictor*/
  if (!(SS_OP_FLAGS(op) & F_CTRL))
    return;

  /* Have a branch here */

  if (correct)
    pred->addr_hits++;

  if (!!pred_taken == !!taken)
    pred->dir_hits++;
  else
```

```
    pred->misses++;

  if (dir_update_ptr->dir.meta)
    if (pred->class == BPredFFBPANN)
      pred->used_ffbpann++;
    else                        /* We're using BPredComb */
      pred->used_2lev++;
  else
    pred->used_bimod++;

  /* keep stats about JR's; also, but don't change any bpred state for
JR's
   * which are returns unless there's no retstack */
  if (op == JR)
    {
      pred->jr_seen++;
      if (correct)
          pred->jr_hits++;

      if (r31p && pred->retstack.size)
          /* return that used the ret-addr stack; no further work to do
*/
          return;
    }

  /* Can exit now if this is a stateless predictor */
  if (pred->class == BPredNotTaken || pred->class == BPredTaken)
    return;

  /*
   * Now we know the branch didn't use the ret-addr stack, and that
this
   * is a stateful predictor
   */

  /* If this is a function call, push return-address onto return-
address
   * stack */
  if ((SS_OP_FLAGS(op) & (F_CTRL|F_CALL)) ==
(F_CTRL|F_CALL) && pred->retstack.size)
    {
      pred->retstack.tos = (pred->retstack.tos + 1)% pred-
>retstack.size;
      pred->retstack.stack[pred->retstack.tos].target =
        baddr + sizeof(SS_INST_TYPE);
      pred->retstack_pushes++;
```

```
    }

  /* update L1 table if appropriate */
  /* L1 table is updated unconditionally for combining predictor
too */
  if ((SS_OP_FLAGS(op) & (F_CTRL|F_UNCOND)) !=
(F_CTRL|F_UNCOND) &&
      (pred->class == BPred2Level || pred->class == BPredComb))
    {
      int l1index, shift_reg;

      /* also update appropriate L1 history register */
      l1index = (baddr >> 3) & (pred->dirpred.twolev-
>config.two.l1size - 1);
      shift_reg = (pred->dirpred.twolev-
>config.two.shiftregs[l1index] << 1) | (
!!taken);
      pred->dirpred.twolev->config.two.shiftregs[l1index] =
        shift_reg & ((1 << pred->dirpred.twolev-
>config.two.shift_width) - 1);
    }

  /* find BTB entry if it's a taken branch (don't allocate for non-
taken) */
  if (taken)
    {
      index = (baddr >> 3) & (pred->btb.sets - 1);

      if (pred->btb.assoc > 1)
        {
          index *= pred->btb.assoc;

          /* Now we know the set; look for a PC match; also identify
           * MRU and LRU items */
          for (i = index; i < (index+pred->btb.assoc) ; i++)
            {
              if (pred->btb.btb_data[i].addr == baddr)
                {
                  /* match */
                  assert(!pbtb);
                  pbtb = &pred->btb.btb_data[i];
                }

              dassert(pred->btb.btb_data[i].prev
                      != pred->btb.btb_data[i].next);
              if (pred->btb.btb_data[i].prev == NULL)
```

```
            {
              /* this is the head of the lru list, ie current MRU item */
              dassert(lruhead == NULL);
              lruhead = &pred->btb.btb_data[i];
            }
          if (pred->btb.btb_data[i].next == NULL)
            {
              /* this is the tail of the lru list, ie the LRU item */
              dassert(lruitem == NULL);
              lruitem = &pred->btb.btb_data[i];
            }
        }
      dassert(lruhead && lruitem);

      if (!pbtb)
        /* missed in BTB; choose the LRU item in this set as the
victim */
        pbtb = lruitem;
      /* else hit, and pbtb points to matching BTB entry */

      /* Update LRU state: selected item, whether selected
because it
       * matched or because it was LRU and selected as a victim,
becomes
       * MRU */
      if (pbtb != lruhead)
        {
          /* this splices out the matched entry... */
          if (pbtb->prev)
            pbtb->prev->next = pbtb->next;
          if (pbtb->next)
            pbtb->next->prev = pbtb->prev;
          /* ...and this puts the matched entry at the head of the list
*/
          pbtb->next = lruhead;
          pbtb->prev = NULL;
          lruhead->prev = pbtb;
          dassert(pbtb->prev || pbtb->next);
          dassert(pbtb->prev != pbtb->next);
        }
      /* else pbtb is already MRU item; do nothing */
    }
  else
    pbtb = &pred->btb.btb_data[index];
}
```

```
  /*
   * Now 'p' is a possibly null pointer into the direction prediction
table,
   * and 'pbtb' is a possibly null pointer into the BTB (either to a
   * matched-on entry or a victim which was LRU in its set)
   */

  /* update state (but not for jumps) */
  if (dir_update_ptr->pdir1 && !((pred->class ==
BPredFFBPANN) && (dir_update_ptr->dir.meta)) )
    {
      if (taken)
        {
          if (*dir_update_ptr->pdir1 < 3)
            ++*dir_update_ptr->pdir1;
        }
      else
        { /* not taken */
          if (*dir_update_ptr->pdir1 > 0)
            --*dir_update_ptr->pdir1;
        }
    }

  /* combining predictor also updates second predictor and meta
predictor */
  /* second direction predictor */
  if (dir_update_ptr->pdir2 && !((!dir_update_ptr->dir.meta) &&
(pred->class == BPredFFBPANN)) )
    {
      if (taken)
        {
          if (*dir_update_ptr->pdir2 < 3)
            ++*dir_update_ptr->pdir2;
        }
      else
        { /* not taken */
          if (*dir_update_ptr->pdir2 > 0)
            --*dir_update_ptr->pdir2;
        }
    }

  /* meta predictor */
  if (dir_update_ptr->pmeta)
    {
      if ((dir_update_ptr->dir.bimod != dir_update_ptr->dir.twolev
&&
```

```
          pred->class == BPredComb ) ||
        (dir_update_ptr->dir.bimod != dir_update_ptr->dir.ffbpann
&&
          pred->class == BPredFFBPANN ) )
       {
        /* we only update meta predictor if directions were different
*/
        if ((dir_update_ptr->dir.twolev == (unsigned int)taken &&
             pred->class == BPredComb ) ||
             (dir_update_ptr->dir.ffbpann == (unsigned int)taken &&
             pred->class == BPredFFBPANN ) )
          {
           /* 2-level or FFBPANN predictor was correct */
           if (*dir_update_ptr->pmeta < 3)
             ++*dir_update_ptr->pmeta;
          }
        else
          {
           /* bimodal predictor was correct */
           if (*dir_update_ptr->pmeta > 0)
             --*dir_update_ptr->pmeta;
          }
       }
   }

  /* update BTB (but only for taken branches) */
  if (pbtb)
    {
     /* update current information */
     dassert(taken);

     if (pbtb->addr == baddr)
       {
        if (!correct)
          pbtb->target = btarget;
       }
     else
       {
        /* enter a new branch in the table */
        pbtb->addr = baddr;
        pbtb->op = op;
        pbtb->target = btarget;
       }
    }

  /* update FFBPANN if we predicted wrong */
```

```
  if((pred->class == BPredFFBPANN) && (dir_update_ptr-
>dir.meta) && (!!taken != !!pred_taken) )   /* Then update the
FFBPANN */
    {
      input[0] = (double) (baddr >> 3);
      input[1] = (double) SS_OPCODE(inst);
      input[2] = (double) RS;
      input[3] = (double) RT;
      trgt[1] = (double) pred_taken ;

      learn_ffbpann(input,pred->dirpred.ffbpann-
>config.ffbpann.net,trgt);
    }


}
```

sim-outorder.c

```
/*
 * sim-outorder.c - sample out-of-order issue perf simulator
implementation
 *
 * This file is a part of the SimpleScalar tool suite written by
 * Todd M. Austin as a part of the Multiscalar Research Project.
 *
 * The tool suite is currently maintained by Doug Burger and Todd
M. Austin.
 *
 * Copyright (C) 1994, 1995, 1996, 1997 by Todd M. Austin
 *
 * This source file is distributed "as is" in the hope that it will be
 * useful.  The tool set comes with no warranty, and no author or
 * distributor accepts any responsibility for the consequences of its
 * use.
 *
 * Everyone is granted permission to copy, modify and redistribute
 * this tool set under the following conditions:
 *
 *    This source code is distributed for non-commercial use only.
 *    Please contact the maintainer for restrictions applying to
 *    commercial use.
 *
 *    Permission is granted to anyone to make or distribute copies
 *    of this source code, either as received or modified, in any
```

```
 *    medium, provided that all copyright notices, permission and
 *    nonwarranty notices are preserved, and that the distributor
 *    grants the recipient permission for further redistribution as
 *    permitted by this document.
 *
 *    Permission is granted to distribute this file in compiled
 *    or executable form under the same conditions that apply for
 *    source code, provided that either:
 *
 *    A. it is accompanied by the corresponding machine-readable
 *       source code,
 *    B. it is accompanied by a written offer, with no time limit,
 *       to give anyone a machine-readable copy of the
corresponding
 *       source code in return for reimbursement of the cost of
 *       distribution.  This written offer must permit verbatim
 *       duplication by anyone, or
 *    C. it is distributed by someone who received only the
 *       executable form, and is accompanied by a copy of the
 *       written offer of source code that they received concurrently.
 *
 * In other words, you are welcome to use, share and improve this
 * source file.  You are forbidden to forbid anyone else to use,
share
 * and improve what you give them.
 *
 * INTERNET: dburger@cs.wisc.edu
 * US Mail:  1210 W. Dayton Street, Madison, WI 53706
 *
 * $Id: sim-outorder.c,v 1.4 1997/04/16 22:10:23 taustin Exp
taustin $
 *
 * $Log: sim-outorder.c,v $
 * Revision 1.4  1997/04/16  22:10:23  taustin
 * added -commit:width support (from kskadron)
 * fixed "bad l2 D-cache parms" fatal string
 *
 * Revision 1.3  1997/03/11  17:17:06  taustin
 * updated copyright
 * `-pcstat' option support added
 * long/int tweaks made for ALPHA target support
 * better defaults defined for caches/TLBs
 * "mstate" command supported added for DLite!
 * supported added for non-GNU C compilers
 * buglet fixed in speculative trace generation
 * multi-level cache hierarchy now supported
```

```
 * two-level predictor supported added
 * I/D-TLB supported added
 * many comments added
 * options package supported added
 * stats package support added
 * resource configuration options extended
 * pipetrace support added
 * DLite! support added
 * writeback throttling now supported
 * decode and issue B/W now decoupled
 * new and improved (and more precise) memory scheduler added
 * cruft for TLB paper removed
 *
 * Revision 1.1  1996/12/05  18:52:32  taustin
 * Initial revision
 *
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <assert.h>
#include <signal.h>

#include "misc.h"
#include "ss.h"
#include "regs.h"
#include "memory.h"
#include "cache.h"
#include "loader.h"
#include "syscall.h"
#include "bpred.h"
#include "resource.h"
#include "bitmap.h"
#include "options.h"
#include "stats.h"
#include "ptrace.h"
#include "dlite.h"
#include "sim.h"

/*
 * This file implements a very detailed out-of-order issue
superscalar
 * processor with a two-level memory system and speculative
execution support.
```

```
 * This simulator is a performance simulator, tracking the latency
of all
 * pipeline operations.
 */


/*
 * simulator options
 */

/* pipeline trace range and output filename */
static int ptrace_nelt = 0;
static char *ptrace_opts[2];

/* instruction fetch queue size (in insts) */
static int ruu_ifq_size;

/* extra branch mis-prediction latency */
static int ruu_branch_penalty;

/* speed of front-end of machine relative to execution core */
static int fetch_speed;

/* branch predictor type
{nottaken|taken|perfect|bimod|2lev|ffbpann} */
static char *pred_type;

/* bimodal predictor config (<table_size>) */
static int bimod_nelt = 1;
static int bimod_config[1] =
  { /* bimod tbl size */2048 };

/* ffbpann predictor config (<meta_table_size> <hl1nodes>
<hl2nodes> <hl3nodes> <hl4nodes>) */
static int ffbpann_nelt = 5;
static int ffbpann_config[5] =
  { /* meta_table_size */1024, /* hidden layer nodes*/4, 3, 2, 0 };

/* 2-level predictor config (<l1size> <l2size> <hist_size> <xor>)
*/
static int twolev_nelt = 4;
static int twolev_config[4] =
  { /* l1size */1, /* l2size */1024, /* hist */8, /* xor */FALSE};

/* combining predictor config (<meta_table_size> */
static int comb_nelt = 1;
```

```
static int comb_config[1] =
  { /* meta_table_size */1024 };

/* return address stack (RAS) size */
static int ras_size = 8;

/* BTB predictor config (<num_sets> <associativity>) */
static int btb_nelt = 2;
static int btb_config[2] =
  { /* nsets */512, /* assoc */4 };

/* instruction decode B/W (insts/cycle) */
static int ruu_decode_width;

/* instruction issue B/W (insts/cycle) */
static int ruu_issue_width;

/* run pipeline with in-order issue */
static int ruu_inorder_issue;

/* issue instructions down wrong execution paths */
static int ruu_include_spec = TRUE;

/* instruction commit B/W (insts/cycle) */
static int ruu_commit_width;

/* register update unit (RUU) size */
static int RUU_size = 8;

/* load/store queue (LSQ) size */
static int LSQ_size = 4;

/* l1 data cache config, i.e., {<config>|none} */
static char *cache_dl1_opt;

/* l1 data cache hit latency (in cycles) */
static int cache_dl1_lat;

/* l2 data cache config, i.e., {<config>|none} */
static char *cache_dl2_opt;

/* l2 data cache hit latency (in cycles) */
static int cache_dl2_lat;

/* l1 instruction cache config, i.e., {<config>|dl1|dl2|none} */
static char *cache_il1_opt;
```

```
/* l1 instruction cache hit latency (in cycles) */
static int cache_il1_lat;

/* l2 instruction cache config, i.e., {<config>|dl1|dl2|none} */
static char *cache_il2_opt;

/* l2 instruction cache hit latency (in cycles) */
static int cache_il2_lat;

/* flush caches on system calls */
static int flush_on_syscalls;

/* convert 64-bit inst addresses to 32-bit inst equivalents */
static int compress_icache_addrs;

/* memory access latency (<first_chunk> <inter_chunk>) */
static int mem_nelt = 2;
static int mem_lat[2] =
  { /* lat to first chunk */18, /* lat between remaining chunks */2 };

/* memory access bus width (in bytes) */
static int mem_bus_width;

/* instruction TLB config, i.e., {<config>|none} */
static char *itlb_opt;

/* data TLB config, i.e., {<config>|none} */
static char *dtlb_opt;

/* inst/data TLB miss latency (in cycles) */
static int tlb_miss_lat;

/* total number of integer ALU's available */
static int res_ialu;

/* total number of integer multiplier/dividers available */
static int res_imult;

/* total number of memory system ports available (to CPU) */
static int res_memport;

/* total number of floating point ALU's available */
static int res_fpalu;

/* total number of floating point multiplier/dividers available */
```

```
static int res_fpmult;

/* text-based stat profiles */
#define MAX_PCSTAT_VARS 8
static int pcstat_nelt = 0;
static char *pcstat_vars[MAX_PCSTAT_VARS];

/* operate in backward-compatible bugs mode (for testing only) */
static int bugcompat_mode;

/*
 * functional unit resource configuration
 */

/* resource pool indices, NOTE: update these if you change
FU_CONFIG */
#define FU_IALU_INDEX                   0
#define FU_IMULT_INDEX                  1
#define FU_MEMPORT_INDEX                2
#define FU_FPALU_INDEX                  3
#define FU_FPMULT_INDEX                         4

/* resource pool definition, NOTE: update FU_*_INDEX defs if
you change this */
struct res_desc fu_config[] = {
  {
    "integer-ALU",
    4,
    0,
    {
      { IntALU, 1, 1 }
    }
  },
  {
    "integer-MULT/DIV",
    1,
    0,
    {
      { IntMULT, 3, 1 },
      { IntDIV, 20, 19 }
    }
  },
  {
    "memory-port",
    2,
    0,
```

```
    {
      { RdPort, 1, 1 },
      { WrPort, 1, 1 }
    }
  },
  {
    "FP-adder",
    4,
    0,
    {
      { FloatADD, 2, 1 },
      { FloatCMP, 2, 1 },
      { FloatCVT, 2, 1 }
    }
  },
  {
    "FP-MULT/DIV",
    1,
    0,
    {
      { FloatMULT, 4, 1 },
      { FloatDIV, 12, 12 },
      { FloatSQRT, 24, 24 }
    }
  },
};


/*
 * simulator stats
 */

/* total number of instructions committed */
static SS_COUNTER_TYPE sim_num_insn = 0;

/* total number of instructions executed */
static SS_COUNTER_TYPE sim_total_insn = 0;

/* total number of memory references committed */
static SS_COUNTER_TYPE sim_num_refs = 0;

/* total number of memory references executed */
static SS_COUNTER_TYPE sim_total_refs = 0;

/* total number of loads committed */
static SS_COUNTER_TYPE sim_num_loads = 0;
```

```
/* total number of loads executed */
static SS_COUNTER_TYPE sim_total_loads = 0;

/* total number of branches committed */
static SS_COUNTER_TYPE sim_num_branches = 0;

/* total number of branches executed */
static SS_COUNTER_TYPE sim_total_branches = 0;

/* cycle counter */
static SS_TIME_TYPE sim_cycle = 0;


/*
 * simulator state variables
 */

/* instruction sequence counter, used to assign unique id's to insts
*/
static unsigned int inst_seq = 0;

/* pipetrace instruction sequence counter */
static unsigned int ptrace_seq = 0;

/* speculation mode, non-zero when mis-speculating, i.e.,
executing
   instructions down the wrong path, thus state recovery will
eventually have
   to occur that resets processor register and memory state back to
the last
   precise state */
static int spec_mode = FALSE;

/* cycles until fetch issue resumes */
static unsigned ruu_fetch_issue_delay = 0;

/* perfect prediction enabled */
static int pred_perfect = FALSE;

/* speculative bpred-update enabled */
static char *bpred_spec_opt;
static enum { spec_ID, spec_WB, spec_CT } bpred_spec_update;

/* level 1 instruction cache, entry level instruction cache */
static struct cache *cache_il1;
```

```
/* level 1 instruction cache */
static struct cache *cache_il2;

/* level 1 data cache, entry level data cache */
static struct cache *cache_dl1;

/* level 2 data cache */
static struct cache *cache_dl2;

/* instruction TLB */
static struct cache *itlb;

/* data TLB */
static struct cache *dtlb;

/* branch predictor */
static struct bpred *pred;

/* functional unit resource pool */
static struct res_pool *fu_pool = NULL;

/* text-based stat profiles */
static struct stat_stat_t *pcstat_stats[MAX_PCSTAT_VARS];
static SS_COUNTER_TYPE
pcstat_lastvals[MAX_PCSTAT_VARS];
static struct stat_stat_t *pcstat_sdists[MAX_PCSTAT_VARS];

/* wedge all stat values into a SS_COUNTER_TYPE */
#define STATVAL(STAT)
        \
  ((STAT)->sc == sc_int
        \
   ? (SS_COUNTER_TYPE)*((STAT)->variant.for_int.var)
        \
   : ((STAT)->sc == sc_uint
        \
     ? (SS_COUNTER_TYPE)*((STAT)->variant.for_uint.var)
        \
     : ((STAT)->sc == sc_counter
        \
         ? *((STAT)->variant.for_counter.var)
        \
         : (panic("bad stat class"), 0))))
```

```
/* memory access latency, assumed to not cross a page boundary */
static unsigned int                     /* total latency of access */
mem_access_latency(int blk_sz)          /* block size accessed
*/
{
  int chunks = (blk_sz + (mem_bus_width - 1)) / mem_bus_width;

  assert(chunks > 0);

  return (/* first chunk latency */mem_lat[0] +
          (/* remainder chunk latency */mem_lat[1] * (chunks -
1)));
}


/*
 * cache miss handlers
 */

/* l1 data cache l1 block miss handler function */
static unsigned int                     /* latency of block access */
dl1_access_fn(enum mem_cmd cmd,         /* access cmd, Read
or Write */
              SS_ADDR_TYPE baddr, /* block address to access */
              int bsize,            /* size of block to access */
              struct cache_blk *blk,    /* ptr to block in upper level
*/
              SS_TIME_TYPE now)         /* time of access */
{
  unsigned int lat;

  if (cache_dl2)
    {
      /* access next level of data cache hierarchy */
      lat = cache_access(cache_dl2, cmd, baddr, NULL, bsize,
                         /* now */now, /* pudata */NULL, /* repl
addr */NULL);
      if (cmd == Read)
          return lat;
      else
          {
            /* FIXME: unlimited write buffers */
            return 0;
          }
    }
  else
```

```
    {
      /* access main memory */
      if (cmd == Read)
          return mem_access_latency(bsize);
      else
          {
            /* FIXME: unlimited write buffers */
            return 0;
          }
    }
}

/* l2 data cache block miss handler function */
static unsigned int                     /* latency of block access */
dl2_access_fn(enum mem_cmd cmd,         /* access cmd, Read
or Write */
              SS_ADDR_TYPE baddr, /* block address to access */
              int bsize,            /* size of block to access */
              struct cache_blk *blk,    /* ptr to block in upper level
*/
              SS_TIME_TYPE now)             /* time of access */
{
  /* this is a miss to the lowest level, so access main memory */
  if (cmd == Read)
    return mem_access_latency(bsize);
  else
    {
      /* FIXME: unlimited write buffers */
      return 0;
    }
}

/* l1 inst cache l1 block miss handler function */
static unsigned int                     /* latency of block access */
il1_access_fn(enum mem_cmd cmd,         /* access cmd, Read
or Write */
              SS_ADDR_TYPE baddr, /* block address to access */
              int bsize,            /* size of block to access */
              struct cache_blk *blk,    /* ptr to block in upper level
*/
              SS_TIME_TYPE now)             /* time of access */
{
  unsigned int lat;

if (cache_il2)
    {
```

```
      /* access next level of inst cache hierarchy */
      lat = cache_access(cache_il2, cmd, baddr, NULL, bsize,
                         /* now */now, /* pudata */NULL, /* repl
addr */NULL);
      if (cmd == Read)
          return lat;
      else
          panic("writes to instruction memory not supported");
    }
  else
    {
      /* access main memory */
      if (cmd == Read)
          return mem_access_latency(bsize);
      else
          panic("writes to instruction memory not supported");
    }
}

/* l2 inst cache block miss handler function */
static unsigned int                     /* latency of block access */
il2_access_fn(enum mem_cmd cmd,         /* access cmd, Read
or Write */
              SS_ADDR_TYPE baddr, /* block address to access */
              int bsize,          /* size of block to access */
              struct cache_blk *blk,    /* ptr to block in upper level
*/
              SS_TIME_TYPE now)         /* time of access */
{
  /* this is a miss to the lowest level, so access main memory */
  if (cmd == Read)
    return mem_access_latency(bsize);
  else
    panic("writes to instruction memory not supported");
}


/*
 * TLB miss handlers
 */

/* inst cache block miss handler function */
static unsigned int                     /* latency of block access */
itlb_access_fn(enum mem_cmd cmd,/* access cmd, Read or Write
*/
               SS_ADDR_TYPE baddr,/* block address to access */
```

```
		int bsize,		/* size of block to access */
		struct cache_blk *blk,	/* ptr to block in upper level
*/
		SS_TIME_TYPE now)	/* time of access */
{
  SS_ADDR_TYPE *phy_page_ptr = (SS_ADDR_TYPE *)blk-
>user_data;

  /* no real memory access, however, should have user data space
attached */
  assert(phy_page_ptr);

  /* fake translation, for now... */
  *phy_page_ptr = 0;

  /* return tlb miss latency */
  return tlb_miss_lat;
}

/* data cache block miss handler function */
static unsigned int			/* latency of block access */
dtlb_access_fn(enum mem_cmd cmd,		/* access cmd, Read
or Write */
		SS_ADDR_TYPE baddr,/* block address to access */
		int bsize,		/* size of block to access */
		struct cache_blk *blk,	/* ptr to block in upper level
*/
		SS_TIME_TYPE now)	/* time of access */
{
  SS_ADDR_TYPE *phy_page_ptr = (SS_ADDR_TYPE *)blk-
>user_data;

  /* no real memory access, however, should have user data space
attached */
  assert(phy_page_ptr);

  /* fake translation, for now... */
  *phy_page_ptr = 0;

  /* return tlb miss latency */
  return tlb_miss_lat;
}


/* register simulator-specific options */
void
```

```
sim_reg_options(struct opt_odb_t *odb)
{
  opt_reg_header(odb,
"sim-outorder: This simulator implements a very detailed out-of-
order issue\n"
"superscalar processor with a two-level memory system and
speculative\n"
"execution support.  This simulator is a performance simulator,
tracking the\n"
"latency of all pipeline operations.\n"
                 );

  /* trace options */

  opt_reg_string_list(odb, "-ptrace",
                "generate pipetrace, i.e., <fname|stdout|stderr>
<range>",
                ptrace_opts, /* arr_sz */2, &ptrace_nelt, /* default
*/NULL,
                /* !print */FALSE, /* format */NULL, /* !accrue
*/FALSE);

  opt_reg_note(odb,
"  Pipetrace range arguments are formatted as follows:\n"
"\n"
"    {{@|#}<start>}:{{@|#|+}<end>}\n"
"\n"
"  Both ends of the range are optional, if neither are specified, the
entire\n"
"  execution is traced.  Ranges that start with a `@' designate an
address\n"
"  range to be traced, those that start with an `#' designate a cycle
count\n"
"  range.  All other range values represent an instruction count
range.  The\n"
"  second argument, if specified with a `+', indicates a value
relative\n"
"  to the first argument, e.g., 1000:+100 == 1000:1100.  Program
symbols may\n"
"  be used in all contexts.\n"
"\n"
"    Examples:   -ptrace FOO.trc #0:#1000\n"
"                -ptrace BAR.trc @2000:\n"
"                -ptrace BLAH.trc :1500\n"
"                -ptrace UXXE.trc :\n"
"                -ptrace FOOBAR.trc @main:+278\n"
```

```
                );

  /* ifetch options */

  opt_reg_int(odb, "-fetch:ifqsize", "instruction fetch queue size (in
insts)",
              &ruu_ifq_size, /* default */4,
              /* print */TRUE, /* format */NULL);

  opt_reg_int(odb, "-fetch:mplat", "extra branch mis-prediction
latency",
              &ruu_branch_penalty, /* default */3,
              /* print */TRUE, /* format */NULL);

  opt_reg_int(odb, "-fetch:speed",
              "speed of front-end of machine relative to execution
core",
              &fetch_speed, /* default */1,
              /* print */TRUE, /* format */NULL);

  /* branch predictor options */

  opt_reg_note(odb,
"  Branch predictor configuration examples for 2-level
predictor:\n"
"    Configurations:   N, M, W, X\n"
"      N   # entries in first level (# of shift register(s))\n"
"      W   width of shift register(s)\n"
"      M   # entries in 2nd level (# of counters, or other FSM)\n"
"      X   (yes-1/no-0) xor history and address for 2nd level index\n"
"    Sample predictors:\n"
"      GAg     : 1, W, 2^W, 0\n"
"      GAp     : 1, W, M (M > 2^W), 0\n"
"      PAg     : N, W, 2^W, 0\n"
"      PAp     : N, W, M (M == 2^(N+W)), 0\n"
"      gshare  : 1, W, 2^W, 1\n"
"  Predictor `comb' combines a bimodal and a 2-level predictor.\n"
"\n"
"  Predictor `ffbpann' combines a bimodal and a feed-forward
artificial\n"
"  neural network predictor.  The configuration for the FFBPANN
is:\n"
"    Meta, HL1Nodes, HL2Nodes, NL3Nodes, HL4Nodes\n"
"      Meta       size of meta table\n"
"      HL1Nodes   number of nodes in Hidden Layer 1\n"
"      HL2Nodes   number of nodes in Hidden Layer 2\n"
```

```
"    HL3Nodes   number of nodes in Hidden Layer 3\n"
"    HL4Nodes   number of nodes in Hidden Layer 4\n"
"   There are four inputs to the neural network (the input layer):\n"
"    The branch address, the op code, the rs value and the rt
value.\n"
"    There is one output (the output layer).\n"
"   Therefore the neural network can have from 2 (input and
output) to\n"
"   6 (up to 4 hidden) layers, of user specified sizes.\n"
            );

  opt_reg_string(odb, "-bpred",
                  "branch predictor type
{nottaken|taken|perfect|bimod|2lev|comb|ffbpann}",
             &pred_type, /* default */"bimod",
             /* print */TRUE, /* format */NULL);

  opt_reg_int_list(odb, "-bpred:bimod",
                    "bimodal predictor config (<table size>)",
                    bimod_config, bimod_nelt, &bimod_nelt,
                    /* default */bimod_config,
                    /* print */TRUE, /* format */NULL, /* !accrue
*/FALSE);

  opt_reg_int_list(odb, "-bpred:ffbpann",
                "ffbpann config (<meta_table_size> <hl1nodes>
<hl2nodes> <hl3nodes> <hl4nodes>)",
                ffbpann_config, ffbpann_nelt, &ffbpann_nelt,
                /* default */ffbpann_config,
                /* print */TRUE, /* format */NULL, /* !accrue
*/FALSE);

  opt_reg_int_list(odb, "-bpred:2lev",
                "2-level predictor config "
                      "(<l1size> <l2size> <hist_size> <xor>)",
                twolev_config, twolev_nelt, &twolev_nelt,
                      /* default */twolev_config,
                /* print */TRUE, /* format */NULL, /* !accrue
*/FALSE);

  opt_reg_int_list(odb, "-bpred:comb",
                    "combining predictor config
(<meta_table_size>)",
                    comb_config, comb_nelt, &comb_nelt,
                    /* default */comb_config,
                    /* print */TRUE, /* format */NULL, /* !accrue
```

```
*/FALSE);

  opt_reg_int(odb, "-bpred:ras",
              "return address stack size (0 for no return stack)",
              &ras_size, /* default */ras_size,
              /* print */TRUE, /* format */NULL);

  opt_reg_int_list(odb, "-bpred:btb",
                   "BTB config (<num_sets> <associativity>)",
                   btb_config, btb_nelt, &btb_nelt,
                   /* default */btb_config,
                   /* print */TRUE, /* format */NULL, /* !accrue
*/FALSE);

  opt_reg_string(odb, "-bpred:spec_update",
                 "speculative predictors update in {ID|WB} (default
non-spec)",
                 &bpred_spec_opt, /* default */NULL,
                 /* print */TRUE, /* format */NULL);

  /* decode options */

  opt_reg_int(odb, "-decode:width",
              "instruction decode B/W (insts/cycle)",
              &ruu_decode_width, /* default */4,
              /* print */TRUE, /* format */NULL);

  /* issue options */

  opt_reg_int(odb, "-issue:width",
              "instruction issue B/W (insts/cycle)",
              &ruu_issue_width, /* default */4,
              /* print */TRUE, /* format */NULL);

  opt_reg_flag(odb, "-issue:inorder", "run pipeline with in-order
issue",
               &ruu_inorder_issue, /* default */FALSE,
               /* print */TRUE, /* format */NULL);

  opt_reg_flag(odb, "-issue:wrongpath",
               "issue instructions down wrong execution paths",
               &ruu_include_spec, /* default */TRUE,
               /* print */TRUE, /* format */NULL);

  /* commit options */
```

```
  opt_reg_int(odb, "-commit:width",
              "instruction commit B/W (insts/cycle)",
              &ruu_commit_width, /* default */4,
              /* print */TRUE, /* format */NULL);

  /* register scheduler options */

  opt_reg_int(odb, "-ruu:size",
              "register update unit (RUU) size",
              &RUU_size, /* default */16,
              /* print */TRUE, /* format */NULL);

  /* memory scheduler options  */

  opt_reg_int(odb, "-lsq:size",
              "load/store queue (LSQ) size",
              &LSQ_size, /* default */8,
              /* print */TRUE, /* format */NULL);

  /* cache options */

  opt_reg_string(odb, "-cache:dl1",
                 "l1 data cache config, i.e., {<config>|none}",
                 &cache_dl1_opt, "dl1:128:32:4:l",
                 /* print */TRUE, NULL);

  opt_reg_note(odb,
"  The cache config parameter <config> has the following
format:\n"
"\n"
"    <name>:<nsets>:<bsize>:<assoc>:<repl>\n"
"\n"
"    <name>   - name of the cache being defined\n"
"    <nsets>  - number of sets in the cache\n"
"    <bsize>  - block size of the cache\n"
"    <assoc>  - associativity of the cache\n"
"    <repl>   - block replacement strategy, 'l'-LRU, 'f'-FIFO, 'r'-
random\n"
"\n"
"    Examples:   -cache:dl1 dl1:4096:32:1:l\n"
"                -dtlb dtlb:128:4096:32:r\n"
               );

  opt_reg_int(odb, "-cache:dl1lat",
              "l1 data cache hit latency (in cycles)",
              &cache_dl1_lat, /* default */1,
```

```
                /* print */TRUE, /* format */NULL);

  opt_reg_string(odb, "-cache:dl2",
                 "l2 data cache config, i.e., {<config>|none}",
                 &cache_dl2_opt, "ul2:1024:64:4:l",
                 /* print */TRUE, NULL);

  opt_reg_int(odb, "-cache:dl2lat",
              "l2 data cache hit latency (in cycles)",
              &cache_dl2_lat, /* default */6,
              /* print */TRUE, /* format */NULL);

  opt_reg_string(odb, "-cache:il1",
                 "l1 inst cache config, i.e.,
{<config>|dl1|dl2|none}",
                 &cache_il1_opt, "il1:512:32:1:l",
                 /* print */TRUE, NULL);

  opt_reg_note(odb,
"  Cache levels can be unified by pointing a level of the instruction
cache\n"
"  hierarchy at the data cache hiearchy using the \"dl1\" and \"dl2\"
cache\n"
"  configuration arguments.  Most sensible combinations are
supported, e.g.,\n"
"\n"
"    A unified l2 cache (il2 is pointed at dl2):\n"
"      -cache:il1 il1:128:64:1:l -cache:il2 dl2\n"
"      -cache:dl1 dl1:256:32:1:l -cache:dl2 ul2:1024:64:2:l\n"
"\n"
"    Or, a fully unified cache hierarchy (il1 pointed at dl1):\n"
"      -cache:il1 dl1\n"
"      -cache:dl1 ul1:256:32:1:l -cache:dl2 ul2:1024:64:2:l\n"
              );

  opt_reg_int(odb, "-cache:il1lat",
              "l1 instruction cache hit latency (in cycles)",
              &cache_il1_lat, /* default */1,
              /* print */TRUE, /* format */NULL);

  opt_reg_string(odb, "-cache:il2",
                 "l2 instruction cache config, i.e.,
{<config>|dl2|none}",
                 &cache_il2_opt, "dl2",
                 /* print */TRUE, NULL);
```

```
  opt_reg_int(odb, "-cache:il2lat",
              "l2 instruction cache hit latency (in cycles)",
              &cache_il2_lat, /* default */6,
              /* print */TRUE, /* format */NULL);

  opt_reg_flag(odb, "-cache:flush", "flush caches on system calls",
               &flush_on_syscalls, /* default */FALSE, /* print
*/TRUE, NULL);

  opt_reg_flag(odb, "-cache:icompress",
               "convert 64-bit inst addresses to 32-bit inst
equivalents",
               &compress_icache_addrs, /* default */FALSE,
               /* print */TRUE, NULL);

  /* mem options */
  opt_reg_int_list(odb, "-mem:lat",
                   "memory access latency (<first_chunk>
<inter_chunk>)",
                   mem_lat, mem_nelt, &mem_nelt, mem_lat,
                   /* print */TRUE, /* format */NULL, /* !accrue
*/FALSE);

  opt_reg_int(odb, "-mem:width", "memory access bus width (in
bytes)",
              &mem_bus_width, /* default */8,
              /* print */TRUE, /* format */NULL);

  /* TLB options */

  opt_reg_string(odb, "-tlb:itlb",
                 "instruction TLB config, i.e., {<config>|none}",
                 &itlb_opt, "itlb:16:4096:4:l", /* print */TRUE,
NULL);

  opt_reg_string(odb, "-tlb:dtlb",
                 "data TLB config, i.e., {<config>|none}",
                 &dtlb_opt, "dtlb:32:4096:4:l", /* print */TRUE,
NULL);

  opt_reg_int(odb, "-tlb:lat",
              "inst/data TLB miss latency (in cycles)",
              &tlb_miss_lat, /* default */30,
              /* print */TRUE, /* format */NULL);

  /* resource configuration */
```

```
  opt_reg_int(odb, "-res:ialu",
              "total number of integer ALU's available",
              &res_ialu, /* default
*/fu_config[FU_IALU_INDEX].quantity,
              /* print */TRUE, /* format */NULL);

  opt_reg_int(odb, "-res:imult",
              "total number of integer multiplier/dividers available",
              &res_imult, /* default
*/fu_config[FU_IMULT_INDEX].quantity,
              /* print */TRUE, /* format */NULL);

  opt_reg_int(odb, "-res:memport",
              "total number of memory system ports available (to
CPU)",
              &res_memport, /* default
*/fu_config[FU_MEMPORT_INDEX].quantity,
              /* print */TRUE, /* format */NULL);

  opt_reg_int(odb, "-res:fpalu",
              "total number of floating point ALU's available",
              &res_fpalu, /* default
*/fu_config[FU_FPALU_INDEX].quantity,
              /* print */TRUE, /* format */NULL);

  opt_reg_int(odb, "-res:fpmult",
              "total number of floating point multiplier/dividers
available",
              &res_fpmult, /* default
*/fu_config[FU_FPMULT_INDEX].quantity,
              /* print */TRUE, /* format */NULL);

  opt_reg_string_list(odb, "-pcstat",
                      "profile stat(s) against text addr's (mult uses
ok)",
                      pcstat_vars, MAX_PCSTAT_VARS,
&pcstat_nelt, NULL,
                      /* !print */FALSE, /* format */NULL, /* accrue
*/TRUE);

  opt_reg_flag(odb, "-bugcompat",
               "operate in backward-compatible bugs mode (for
testing only)",
               &bugcompat_mode, /* default */FALSE, /* print
*/TRUE, NULL);
```

```
}

/* check simulator-specific option values */
void
sim_check_options(struct opt_odb_t *odb,        /* options database
*/
                  int argc, char **argv)        /* command line
arguments */
{
  char name[128], c;
  int nsets, bsize, assoc;

  if (ruu_ifq_size < 1 || (ruu_ifq_size & (ruu_ifq_size - 1)) != 0)
    fatal("inst fetch queue size must be positive > 0 and a power of
two");

  if (ruu_branch_penalty < 1)
    fatal("mis-prediction penalty must be at least 1 cycle");

  if (fetch_speed < 1)
    fatal("front-end speed must be positive and non-zero");

  if (!mystricmp(pred_type, "perfect"))
    {
      /* perfect predictor */
      pred = NULL;
      pred_perfect = TRUE;
    }
  else if (!mystricmp(pred_type, "taken"))
    {
      /* static predictor, not taken */
      pred = bpred_create(BPredTaken, 0, 0, 0, 0, 0, 0, 0, 0, 0);
    }
  else if (!mystricmp(pred_type, "nottaken"))
    {
      /* static predictor, taken */
      pred = bpred_create(BPredNotTaken, 0, 0, 0, 0, 0, 0, 0, 0, 0);
    }
  else if (!mystricmp(pred_type, "bimod"))
    {
      /* bimodal predictor, bpred_create() checks BTB_SIZE */
      if (bimod_nelt != 1)
          fatal("bad bimod predictor config (<table_size>)");
      if (btb_nelt != 2)
          fatal("bad btb config (<num_sets> <associativity>)");
```

```
    /* bimodal predictor, bpred_create() checks BTB_SIZE */
    pred = bpred_create(BPred2bit,
                        /* bimod table size */bimod_config[0],
                        /* 2lev l1 size */0,
                        /* 2lev l2 size */0,
                        /* meta table size */0,
                        /* history reg size */0,
                        /* history xor address */0,
                        /* btb sets */btb_config[0],
                        /* btb assoc */btb_config[1],
                        /* ret-addr stack size */ras_size);
  }
 else if (!mystricmp(pred_type, "ffbpann"))
  {
    /* ffbpann predictor - combination bimodal and ffbpann */
    if (bimod_nelt != 1)
      fatal("bad bimod predictor config (<table_size>)");
    if (btb_nelt != 2)
      fatal("bad btb config (<num_sets> <associativity>)");
    if (ffbpann_nelt != 5)
      fatal("bad ffbpann predictor config (<meta_table_size>
<hl1nodes> <hl2nodes> <hl3nodes>
<hl4nodes>)");

    /* bimodal predictor, bpred_create() checks BTB_SIZE */
    pred = bpred_create(BPredFFBPANN,
                    /* bimod table size */bimod_config[0],
                    /* # of nodes in hidden layer1
*/ffbpann_config[1],
                    /* # of nodes in hidden layer2
*/ffbpann_config[2],
                    /* meta table size */ffbpann_config[0],
                    /* # of nodes in hidden layer3
*/ffbpann_config[3],
                    /* # of nodes in hidden layer4
*/ffbpann_config[4],
                    /* btb sets */btb_config[0],
                    /* btb assoc */btb_config[1],
                    /* ret-addr stack size */ras_size);
  }
 else if (!mystricmp(pred_type, "2lev"))
  {
    /* 2-level adaptive predictor, bpred_create() checks args */
    if (twolev_nelt != 4)
        fatal("bad 2-level pred config (<l1size> <l2size>
<hist_size> <xor>)");
```

```
    if (btb_nelt != 2)
        fatal("bad btb config (<num_sets> <associativity>)");

    pred = bpred_create(BPred2Level,
                        /* bimod table size */0,
                        /* 2lev l1 size */twolev_config[0],
                        /* 2lev l2 size */twolev_config[1],
                        /* meta table size */0,
                        /* history reg size */twolev_config[2],
                        /* history xor address */twolev_config[3],
                        /* btb sets */btb_config[0],
                        /* btb assoc */btb_config[1],
                        /* ret-addr stack size */ras_size);
  }
 else if (!mystricmp(pred_type, "comb"))
  {
    /* combining predictor, bpred_create() checks args */
    if (twolev_nelt != 4)
        fatal("bad 2-level pred config (<l1size> <l2size>
<hist_size> <xor>)");
    if (bimod_nelt != 1)
        fatal("bad bimod predictor config (<table_size>)");
    if (comb_nelt != 1)
        fatal("bad combining predictor config
(<meta_table_size>)");
    if (btb_nelt != 2)
        fatal("bad btb config (<num_sets> <associativity>)");

    pred = bpred_create(BPredComb,
                        /* bimod table size */bimod_config[0],
                        /* l1 size */twolev_config[0],
                        /* l2 size */twolev_config[1],
                        /* meta table size */comb_config[0],
                        /* history reg size */twolev_config[2],
                        /* history xor address */twolev_config[3],
                        /* btb sets */btb_config[0],
                        /* btb assoc */btb_config[1],
                        /* ret-addr stack size */ras_size);
  }
 else
  fatal("cannot parse predictor type `%s'", pred_type);

 if (!bpred_spec_opt)
  bpred_spec_update = spec_CT;
 else if (!mystricmp(bpred_spec_opt, "ID"))
  bpred_spec_update = spec_ID;
```

```
  else if (!mystricmp(bpred_spec_opt, "WB"))
    bpred_spec_update = spec_WB;
  else
    fatal("bad speculative update stage specifier, use {ID|WB}");

  if (ruu_decode_width < 1 || (ruu_decode_width &
(ruu_decode_width-1)) != 0)
    fatal("issue width must be positive non-zero and a power of
two");

  if (ruu_issue_width < 1 || (ruu_issue_width & (ruu_issue_width-
1)) != 0)
    fatal("issue width must be positive non-zero and a power of
two");

  if (ruu_commit_width < 1)
    fatal("commit width must be positive non-zero");

  if (RUU_size < 2 || (RUU_size & (RUU_size-1)) != 0)
    fatal("RUU size must be a positive number > 1 and a power of
two");

  if (LSQ_size < 2 || (LSQ_size & (LSQ_size-1)) != 0)
    fatal("LSQ size must be a positive number > 1 and a power of
two");

  /* use a level 1 D-cache? */
  if (!mystricmp(cache_dl1_opt, "none"))
    {
      cache_dl1 = NULL;

      /* the level 2 D-cache cannot be defined */
      if (strcmp(cache_dl2_opt, "none"))
          fatal("the l1 data cache must defined if the l2 cache is
defined");
      cache_dl2 = NULL;
    }
  else /* dl1 is defined */
    {
      if (sscanf(cache_dl1_opt, "%[^:]:%d:%d:%d:%c",
                 name, &nsets, &bsize, &assoc, &c) != 5)
          fatal("bad l1 D-cache parms:
<name>:<nsets>:<bsize>:<assoc>:<repl>");
      cache_dl1 = cache_create(name, nsets, bsize, /* balloc
*/FALSE,
                               /* usize */0, assoc,
```

```
cache_char2policy(c),
                                dl1_access_fn, /* hit lat
*/cache_dl1_lat);

    /* is the level 2 D-cache defined? */
    if (!mystricmp(cache_dl2_opt, "none"))
        cache_dl2 = NULL;
    else
        {
         if (sscanf(cache_dl2_opt, "%[^:]:%d:%d:%d:%c",
                    name, &nsets, &bsize, &assoc, &c) != 5)
           fatal("bad l2 D-cache parms: "
                 "<name>:<nsets>:<bsize>:<assoc>:<repl>");
         cache_dl2 = cache_create(name, nsets, bsize, /* balloc
*/FALSE,
                                  /* usize */0, assoc,
cache_char2policy(c),
                                  dl2_access_fn, /* hit lat
*/cache_dl2_lat);
        }
   }

  /* use a level 1 I-cache? */
  if (!mystricmp(cache_il1_opt, "none"))
    {
     cache_il1 = NULL;

     /* the level 2 I-cache cannot be defined */
     if (strcmp(cache_il2_opt, "none"))
         fatal("the l1 inst cache must defined if the l2 cache is
defined");
     cache_il2 = NULL;
    }
  else if (!mystricmp(cache_il1_opt, "dl1"))
    {
     if (!cache_dl1)
         fatal("I-cache l1 cannot access D-cache l1 as it's
undefined");
     cache_il1 = cache_dl1;

     /* the level 2 I-cache cannot be defined */
     if (strcmp(cache_il2_opt, "none"))
         fatal("the l1 inst cache must defined if the l2 cache is
defined");
     cache_il2 = NULL;
    }
```

```
  else if (!mystricmp(cache_il1_opt, "dl2"))
    {
      if (!cache_dl2)
          fatal("I-cache l1 cannot access D-cache l2 as it's
undefined");
      cache_il1 = cache_dl2;

      /* the level 2 I-cache cannot be defined */
      if (strcmp(cache_il2_opt, "none"))
          fatal("the l1 inst cache must defined if the l2 cache is
defined");
      cache_il2 = NULL;
    }
  else /* il1 is defined */
    {
      if (sscanf(cache_il1_opt, "%[^:]:%d:%d:%d:%c",
                 name, &nsets, &bsize, &assoc, &c) != 5)
          fatal("bad l1 I-cache parms:
<name>:<nsets>:<bsize>:<assoc>:<repl>");
      cache_il1 = cache_create(name, nsets, bsize, /* balloc
*/FALSE,
                               /* usize */0, assoc,
cache_char2policy(c),
                               il1_access_fn, /* hit lat */cache_il1_lat);

      /* is the level 2 D-cache defined? */
      if (!mystricmp(cache_il2_opt, "none"))
          cache_il2 = NULL;
      else if (!mystricmp(cache_il2_opt, "dl2"))
          {
            if (!cache_dl2)
              fatal("I-cache l2 cannot access D-cache l2 as it's
undefined");
            cache_il2 = cache_dl2;
          }
      else
          {
            if (sscanf(cache_il2_opt, "%[^:]:%d:%d:%d:%c",
                       name, &nsets, &bsize, &assoc, &c) != 5)
              fatal("bad l2 I-cache parms: "
                    "<name>:<nsets>:<bsize>:<assoc>:<repl>");
            cache_il2 = cache_create(name, nsets, bsize, /* balloc
*/FALSE,
                                     /* usize */0, assoc,
cache_char2policy(c),
                                     il2_access_fn, /* hit lat
```

```
*/cache_il2_lat);
          }
    }

  /* use an I-TLB? */
  if (!mystricmp(itlb_opt, "none"))
    itlb = NULL;
  else
    {
      if (sscanf(itlb_opt, "%[^:]:%d:%d:%d:%c",
                 name, &nsets, &bsize, &assoc, &c) != 5)
        fatal("bad TLB parms:
<name>:<nsets>:<page_size>:<assoc>:<repl>");
      itlb = cache_create(name, nsets, bsize, /* balloc */FALSE,
                          /* usize */sizeof(SS_ADDR_TYPE),
assoc,
                          cache_char2policy(c), itlb_access_fn,
                          /* hit latency */1);
    }

  /* use a D-TLB? */
  if (!mystricmp(dtlb_opt, "none"))
    dtlb = NULL;
  else
    {
      if (sscanf(dtlb_opt, "%[^:]:%d:%d:%d:%c",
                 name, &nsets, &bsize, &assoc, &c) != 5)
        fatal("bad TLB parms:
<name>:<nsets>:<page_size>:<assoc>:<repl>");
      dtlb = cache_create(name, nsets, bsize, /* balloc */FALSE,
                          /* usize */sizeof(SS_ADDR_TYPE),
assoc,
                          cache_char2policy(c), dtlb_access_fn,
                          /* hit latency */1);
    }

  if (cache_dl1_lat < 1)
    fatal("l1 data cache latency must be greater than zero");

  if (cache_dl2_lat < 1)
    fatal("l2 data cache latency must be greater than zero");

  if (cache_il1_lat < 1)
    fatal("l1 instruction cache latency must be greater than zero");

  if (cache_il2_lat < 1)
```

```
    fatal("l2 instruction cache latency must be greater than zero");

  if (mem_nelt != 2)
    fatal("bad memory access latency (<first_chunk>
<inter_chunk>)");

  if (mem_lat[0] < 1 || mem_lat[1] < 1)
    fatal("all memory access latencies must be greater than zero");

  if (mem_bus_width < 1 || (mem_bus_width & (mem_bus_width-
1)) != 0)
    fatal("memory bus width must be positive non-zero and a power
of two");

  if (tlb_miss_lat < 1)
    fatal("TLB miss latency must be greater than zero");

  if (res_ialu < 1)
    fatal("number of integer ALU's must be greater than zero");
  if (res_ialu > MAX_INSTS_PER_CLASS)
    fatal("number of integer ALU's must be <=
MAX_INSTS_PER_CLASS");
  fu_config[FU_IALU_INDEX].quantity = res_ialu;

  if (res_imult < 1)
    fatal("number of integer multiplier/dividers must be greater than
zero");
  if (res_imult > MAX_INSTS_PER_CLASS)
    fatal("number of integer mult/div's must be <=
MAX_INSTS_PER_CLASS");
  fu_config[FU_IMULT_INDEX].quantity = res_imult;

  if (res_memport < 1)
    fatal("number of memory system ports must be greater than
zero");
  if (res_memport > MAX_INSTS_PER_CLASS)
    fatal("number of memory system ports must be <=
MAX_INSTS_PER_CLASS");
  fu_config[FU_MEMPORT_INDEX].quantity = res_memport;

  if (res_fpalu < 1)
    fatal("number of floating point ALU's must be greater than
zero");
  if (res_fpalu > MAX_INSTS_PER_CLASS)
    fatal("number of floating point ALU's must be <=
MAX_INSTS_PER_CLASS");
```

```
  fu_config[FU_FPALU_INDEX].quantity = res_fpalu;

  if (res_fpmult < 1)
    fatal("number of floating point multiplier/dividers must be >
zero");
  if (res_fpmult > MAX_INSTS_PER_CLASS)
    fatal("number of FP mult/div's must be <=
MAX_INSTS_PER_CLASS");
  fu_config[FU_FPMULT_INDEX].quantity = res_fpmult;
}

/* print simulator-specific configuration information */
void
sim_aux_config(FILE *stream)            /* output stream */
{
  /* nada */
}

/* register simulator-specific statistics */
void
sim_reg_stats(struct stat_sdb_t *sdb)   /* stats database */
{
  int i;

  /* register baseline stats */
  stat_reg_counter(sdb, "sim_num_insn",
                   "total number of instructions committed",
                   &sim_num_insn, 0, NULL);
  stat_reg_counter(sdb, "sim_num_refs",
                   "total number of loads and stores committed",
                   &sim_num_refs, 0, NULL);
  stat_reg_counter(sdb, "sim_num_loads",
                   "total number of loads committed",
                   &sim_num_loads, 0, NULL);
  stat_reg_formula(sdb, "sim_num_stores",
                   "total number of stores committed",
                   "sim_num_refs - sim_num_loads", NULL);
  stat_reg_counter(sdb, "sim_num_branches",
                   "total number of branches committed",
                   &sim_num_branches, /* initial value */0, /*
format */NULL);
  stat_reg_int(sdb, "sim_elapsed_time",
               "total simulation time in seconds",
               (int *)&sim_elapsed_time, 0, NULL);
  stat_reg_formula(sdb, "sim_inst_rate",
                   "simulation speed (in insts/sec)",
```

```
                    "sim_num_insn / sim_elapsed_time", NULL);

  stat_reg_counter(sdb, "sim_total_insn",
                    "total number of instructions executed",
                    &sim_total_insn, 0, NULL);
  stat_reg_counter(sdb, "sim_total_refs",
                    "total number of loads and stores executed",
                    &sim_total_refs, 0, NULL);
  stat_reg_counter(sdb, "sim_total_loads",
                    "total number of loads executed",
                    &sim_total_loads, 0, NULL);
  stat_reg_formula(sdb, "sim_total_stores",
                    "total number of stores executed",
                    "sim_total_refs - sim_total_loads", NULL);
  stat_reg_counter(sdb, "sim_total_branches",
                    "total number of branches executed",
                    &sim_total_branches, /* initial value */0, /*
format */NULL);

  /* register performance stats */
  stat_reg_counter(sdb, "sim_cycle",
                    "total simulation time in cycles",
                    &sim_cycle, /* initial value */0, /* format
*/NULL);
  stat_reg_formula(sdb, "sim_IPC",
                    "instructions per cycle",
                    "sim_num_insn / sim_cycle", /* format */NULL);
  stat_reg_formula(sdb, "sim_CPI",
                    "cycles per instruction",
                    "sim_cycle / sim_num_insn", /* format */NULL);
  stat_reg_formula(sdb, "sim_exec_BW",
                    "total instructions (mis-spec + committed) per
cycle",
                    "sim_total_insn / sim_cycle", /* format */NULL);
  stat_reg_formula(sdb, "sim_IPB",
                    "instruction per branch",
                    "sim_num_insn / sim_num_branches", /* format
*/NULL);

  /* register predictor stats */
  if (pred)
    bpred_reg_stats(pred, sdb);

  /* register cache stats */
  if (cache_il1
      && (cache_il1 != cache_dl1 && cache_il1 != cache_dl2))
```

```
  cache_reg_stats(cache_il1, sdb);
 if (cache_il2
    && (cache_il2 != cache_dl1 && cache_il2 != cache_dl2))
  cache_reg_stats(cache_il2, sdb);
 if (cache_dl1)
  cache_reg_stats(cache_dl1, sdb);
 if (cache_dl2)
  cache_reg_stats(cache_dl2, sdb);
 if (itlb)
  cache_reg_stats(itlb, sdb);
 if (dtlb)
  cache_reg_stats(dtlb, sdb);

 for (i=0; i<pcstat_nelt; i++)
  {
    char buf[512], buf1[512];
    struct stat_stat_t *stat;

    /* track the named statistical variable by text address */

    /* find it... */
    stat = stat_find_stat(sdb, pcstat_vars[i]);
    if (!stat)
        fatal("cannot locate any statistic named `%s'",
pcstat_vars[i]);

    /* stat must be an integral type */
    if (stat->sc != sc_int && stat->sc != sc_uint && stat->sc !=
sc_counter)
        fatal("`-pcstat' statistical variable `%s' is not an integral
type",
              stat->name);

    /* register this stat */
    pcstat_stats[i] = stat;
    pcstat_lastvals[i] = STATVAL(stat);

    /* declare the sparce text distribution */
    sprintf(buf, "%s_by_pc", stat->name);
    sprintf(buf1, "%s (by text address)", stat->desc);
    pcstat_sdists[i] = stat_reg_sdist(sdb, buf, buf1,
                                      /* initial value */0,
                                      /* print format
*/(PF_COUNT|PF_PDF),
                                      /* format */"0x%lx %lu
%.2f",
```

```
                                        /* print fn */NULL);
    }
}

/* forward declarations */
static void ruu_init(void);
static void lsq_init(void);
static void rslink_init(int nlinks);
static void eventq_init(void);
static void readyq_init(void);
static void cv_init(void);
static void tracer_init(void);
static void fetch_init(void);

/* default register state accessor, used by DLite */
static char *                           /* err str, NULL for
no err */
simoo_reg_obj(enum dlite_access_t at,           /* access type
*/
              enum dlite_reg_t rt,      /* reg bank to probe
*/
              int reg,                  /* register number */
              union dlite_reg_val_t *val);      /* input, output */

/* default memory state accessor, used by DLite */
static char *                           /* err str, NULL for
no err */
simoo_mem_obj(enum dlite_access_t at,           /* access type
*/
              SS_ADDR_TYPE addr,        /* address to access */
              char *p,                  /* input/output buffer
*/
              int nbytes);              /* size of access */

/* default machine state accessor, used by DLite */
static char *                           /* err str, NULL for
no err */
simoo_mstate_obj(FILE *stream,                  /* output
stream */
                 char *cmd);            /* optional command
string */

/* total RS links allocated at program start */
#define MAX_RS_LINKS            4096

/* initialize the simulator */
```

```
void
sim_init(void)
{
  sim_num_insn = 0;
  sim_num_refs = 0;

  /* initialize here, so symbols can be loaded */
  if (ptrace_nelt == 2)
    {
      /* generate a pipeline trace */
      ptrace_open(/* fname */ptrace_opts[0], /* range
*/ptrace_opts[1]);
    }
  else if (ptrace_nelt == 0)
    {
      /* no pipetracing */;
    }
  else
    fatal("bad pipetrace args, use: <fname|stdout|stderr> <range>");

  /* decode all instructions */
  {
    SS_ADDR_TYPE addr;
    SS_INST_TYPE inst;

    if (OP_MAX > 255)
      fatal("cannot do fast decoding, too many opcodes");

    debug("sim: decoding text segment...");
    for (addr=ld_text_base;
           addr < (ld_text_base+ld_text_size);
           addr += SS_INST_SIZE)
      {
          inst = __UNCHK_MEM_ACCESS(SS_INST_TYPE,
addr);
          inst.a = (inst.a & ~0xff) | (unsigned
int)SS_OP_ENUM(SS_OPCODE(inst));
          __UNCHK_MEM_ACCESS(SS_INST_TYPE, addr) =
inst;
      }
  }

  /* initialize the simulation engine */
  fu_pool = res_create_pool("fu-pool", fu_config,
N_ELT(fu_config));
  rslink_init(MAX_RS_LINKS);
```

```
  tracer_init();
  fetch_init();
  cv_init();
  eventq_init();
  readyq_init();
  ruu_init();
  lsq_init();

  /* initialize the DLite debugger */
  dlite_init(simoo_reg_obj, simoo_mem_obj, simoo_mstate_obj);
}

/* dump simulator-specific auxiliary simulator statistics */
void
sim_aux_stats(FILE *stream)             /* output stream */
{
  /* nada */
}

/* un-initialize the simulator */
void
sim_uninit(void)
{
  if (ptrace_nelt > 0)
    ptrace_close();
}


/*
 * processor core definitions and declarations
 */

/* inst tag type, used to tag an operation instance in the RUU */
typedef unsigned int INST_TAG_TYPE;

/* inst sequence type, used to order instructions in the ready list, if
   this rolls over the ready list order temporarily will get messed up,
   but execution will continue and complete correctly */
typedef unsigned int INST_SEQ_TYPE;


/* total input dependencies possible */
#define MAX_IDEPS               3

/* total output dependencies possible */
#define MAX_ODEPS               2
```

```
/* a register update unit (RUU) station, this record is contained in
the
   processors RUU, which serves as a collection of ordered
reservations
   stations.  The reservation stations capture register results and
await
   the time when all operands are ready, at which time the
instruction is
   issued to the functional units; the RUU is an order circular
queue, in which
   instructions are inserted in fetch (program) order, results are
stored in
   the RUU buffers, and later when an RUU entry is the oldest entry
in the
   machines, it and its instruction's value is retired to the
architectural
   register file in program order, NOTE: the RUU and LSQ share
the same
   structure, this is useful because loads and stores are split into two
   operations: an effective address add and a load/store, the add is
inserted
   into the RUU and the load/store inserted into the LSQ, allowing
the add
   to wake up the load/store when effective address computation
has finished */
struct RUU_station {
  /* inst info */
  SS_INST_TYPE IR;                  /* instruction bits */
  enum ss_opcode op;                /* decoded instruction opcode */
  SS_ADDR_TYPE PC, next_PC, pred_PC;    /* inst PC, next PC,
predicted PC */
  int in_LSQ;                     /* non-zero if op is in LSQ */
  int ea_comp;                    /* non-zero if op is an addr comp */
  int recover_inst;               /* start of mis-speculation? */
  int stack_recover_idx;                  /* non-speculative TOS for
RSB pred */
  struct bpred_update dir_update;       /* bpred direction update info
*/
  int spec_mode;                  /* non-zero if issued in spec_mode
*/
  SS_ADDR_TYPE addr;                /* effective address for ld/st's
*/
  INST_TAG_TYPE tag;                /* RUU slot tag, increment to
                                       squash operation */
  INST_SEQ_TYPE seq;                /* instruction sequence, used
```

```
to
                                        sort the ready list and tag
inst */
  unsigned int ptrace_seq;              /* pipetrace sequence number
*/

  /* instruction status */
  int queued;                   /* operands ready and queued */
  int issued;                  /* operation is/was executing */
  int completed;                 /* operation has completed
execution */

  /* output operand dependency list, these lists are used to
     limit the number of associative searches into the RUU when
     instructions complete and need to wake up dependent insts */
  int onames[MAX_ODEPS];              /* output logical names
(NA=unused) */
  struct RS_link *odep_list[MAX_ODEPS]; /* chains to
consuming operations */

  /* input dependent links, the output chains rooted above use these
     fields to mark input operands as ready, when all these fields
have
     been set non-zero, the RUU operation has all of its register
     operands, it may commence execution as soon as all of its
memory
     operands are known to be read (see lsq_refresh() for details on
     enforcing memory dependencies) */
  int idep_ready[MAX_IDEPS];          /* input operand ready? */
};

/* non-zero if all register operands are ready, update with
MAX_IDEPS */
#define OPERANDS_READY(RS)                                      \
  ((RS)->idep_ready[0] && (RS)->idep_ready[1] && (RS)-
>idep_ready[2])

/* register update unit, combination of reservation stations and
reorder
   buffer device, organized as a circular queue */
static struct RUU_station *RUU;         /* register update unit */
static int RUU_head, RUU_tail;          /* RUU head and tail
pointers */
static int RUU_num;                   /* num entries currently in RUU
*/
```

```
/* allocate and initialize register update unit (RUU) */
static void
ruu_init(void)
{
  RUU = calloc(RUU_size, sizeof(struct RUU_station));
  if (!RUU)
    fatal("out of virtual memory");

  RUU_num = 0;
  RUU_head = RUU_tail = 0;
}

/* dump the contents of the RUU */
static void
ruu_dumpent(struct RUU_station *rs,                    /* ptr to RUU
station */
            int index,                                 /* entry index */
            FILE *stream,                              /* output stream */
            int header)                                /* print header? */
{
  if (header)
    fprintf(stream, "idx: %2d: opcode: %s, inst: `",
            index, SS_OP_NAME(rs->op));
  else
    fprintf(stream, "       opcode: %s, inst: `",
            SS_OP_NAME(rs->op));
  ss_print_insn(rs->IR, rs->PC, stream);
  fprintf(stream, "'\n");
  fprintf(stream, "         PC: 0x%08x, NPC: 0x%08x (pred_PC:
0x%08x)\n",
          rs->PC, rs->next_PC, rs->pred_PC);
  fprintf(stream, "         in_LSQ: %s, ea_comp: %s, recover_inst:
%s\n",
          rs->in_LSQ ? "t" : "f",
          rs->ea_comp ? "t" : "f",
          rs->recover_inst ? "t" : "f");
  fprintf(stream, "         spec_mode: %s, addr: 0x%08x, tag:
0x%08x\n",
          rs->spec_mode ? "t" : "f", rs->addr, rs->tag);
  fprintf(stream, "         seq: 0x%08x, ptrace_seq: 0x%08x\n",
          rs->seq, rs->ptrace_seq);
  fprintf(stream, "         queued: %s, issued: %s, completed: %s\n",
          rs->queued ? "t" : "f",
          rs->issued ? "t" : "f",
          rs->completed ? "t" : "f");
  fprintf(stream, "         operands ready: %s\n",
```

```
            OPERANDS_READY(rs) ? "t" : "f");
}

/* dump the contents of the RUU */
static void
ruu_dump(FILE *stream)                                  /* output
stream */
{
  int num, head;
  struct RUU_station *rs;

  fprintf(stream, "** RUU state **\n");
  fprintf(stream, "RUU_head: %d, RUU_tail: %d\n", RUU_head,
RUU_tail);
  fprintf(stream, "RUU_num: %d\n", RUU_num);

  num = RUU_num;
  head = RUU_head;
  while (num)
    {
      rs = &RUU[head];
      ruu_dumpent(rs, rs - RUU, stream, /* header */TRUE);
      head = (head + 1) % RUU_size;
      num--;
    }
}

/*
 * load/store queue (LSQ): holds loads and stores in program order,
indicating
 * status of load/store access:
 *
 *   - issued: address computation complete, memory access in
progress
 *   - completed: memory access has completed, stored value
available
 *   - squashed: memory access was squashed, ignore this entry
 *
 * loads may execute when:
 *   1) register operands are ready, and
 *   2) memory operands are ready (no earlier unresolved store)
 *
 * loads are serviced by:
 *   1) previous store at same address in LSQ (hit latency), or
 *   2) data cache (hit latency + miss latency)
 *
```

```
 * stores may execute when:
 *   1) register operands are ready
 *
 * stores are serviced by:
 *   1) depositing store value into the load/store queue
 *   2) writing store value to the store buffer (plus tag check) at
commit
 *   3) writing store buffer entry to data cache when cache is free
 *
 * NOTE: the load/store queue can bypass a store value to a load in
the same
 *   cycle the store executes (using a bypass network), thus stores
complete
 *   in effective zero time after their effective address is known
 */
static struct RUU_station *LSQ;         /* load/store queue */
static int LSQ_head, LSQ_tail;          /* LSQ head and tail pointers
*/
static int LSQ_num;                     /* num entries currently in LSQ
*/

/*
 * input dependencies for stores in the LSQ:
 *   idep #0 - operand input (value that is store'd)
 *   idep #1 - effective address input (address of store operation)
 */
#define STORE_OP_INDEX                  0
#define STORE_ADDR_INDEX                1

#define STORE_OP_READY(RS)              ((RS)-
>idep_ready[STORE_OP_INDEX])
#define STORE_ADDR_READY(RS)            ((RS)-
>idep_ready[STORE_ADDR_INDEX])

/* allocate and initialize the load/store queue (LSQ) */
static void
lsq_init(void)
{
  LSQ = calloc(LSQ_size, sizeof(struct RUU_station));
  if (!LSQ)
    fatal("out of virtual memory");

  LSQ_num = 0;
  LSQ_head = LSQ_tail = 0;
}
```

```
/* dump the contents of the RUU */
static void
lsq_dump(FILE *stream)                                  /* output
stream */
{
  int num, head;
  struct RUU_station *rs;

  fprintf(stream, "** LSQ state **\n");
  fprintf(stream, "LSQ_head: %d, LSQ_tail: %d\n", LSQ_head,
LSQ_tail);
  fprintf(stream, "LSQ_num: %d\n", LSQ_num);

  num = LSQ_num;
  head = LSQ_head;
  while (num)
    {
      rs = &LSQ[head];
      ruu_dumpent(rs, rs - LSQ, stream, /* header */TRUE);
      head = (head + 1) % LSQ_size;
      num--;
    }
}


/*
 * RS_LINK defs and decls
 */

/* a reservation station link: this structure links elements of a RUU
   reservation station list; used for ready instruction queue, event
queue, and
   output dependency lists; each RS_LINK node contains a pointer
to the RUU
   entry it references along with an instance tag, the RS_LINK is
only valid if
   the instruction instance tag matches the instruction RUU entry
instance tag;
   this strategy allows entries in the RUU can be squashed and
reused without
   updating the lists that point to it, which significantly improves
the
   performance of (all to frequent) squash events */
struct RS_link {
  struct RS_link *next;               /* next entry in list */
  struct RUU_station *rs;              /* referenced RUU resv station
```

```
*/
  INST_TAG_TYPE tag;                /* inst instance sequence
number */
  union {
    SS_TIME_TYPE when;               /* time stamp of entry (for
eventq) */
    INST_SEQ_TYPE seq;               /* inst sequence */
    int opnum;                     /* input/output operand number */
  } x;
};

/* RS link free list, grab RS_LINKs from here, when needed */
static struct RS_link *rslink_free_list;

/* NULL value for an RS link */
#define RSLINK_NULL_DATA                { NULL, NULL, 0 }
static struct RS_link RSLINK_NULL = RSLINK_NULL_DATA;

/* NULL value for an RS link */
#define RSLINK_INIT(RSL, RS)
          \
  ((RSL).next = NULL, (RSL).rs = (RS), (RSL).tag = (RS)->tag)

/* non-zero if RS link is NULL */
#define RSLINK_IS_NULL(LINK)          ((LINK)->rs ==
NULL)

/* non-zero if RS link is to a valid (non-squashed) entry */
#define RSLINK_VALID(LINK)            ((LINK)->tag ==
(LINK)->rs->tag)

/* extra RUU reservation station pointer */
#define RSLINK_RS(LINK)               ((LINK)->rs)

/* get a new RS link record */
#define RSLINK_NEW(DST, RS)
          \
  { struct RS_link *n_link;
          \
    if (!rslink_free_list)                                        \
      panic("out of rs links");
          \
    n_link = rslink_free_list;
          \
    rslink_free_list = rslink_free_list->next;
          \
```

```
    n_link->next = NULL;
          \
    n_link->rs = (RS); n_link->tag = n_link->rs->tag;
          \
    (DST) = n_link;
          \
  }

/* free an RS link record */
#define RSLINK_FREE(LINK)
          \
  {  struct RS_link *f_link = (LINK);
          \
     f_link->rs = NULL; f_link->tag = 0;
          \
     f_link->next = rslink_free_list;
          \
     rslink_free_list = f_link;
          \
  }

/* FIXME: could this be faster!!! */
/* free an RS link list */
#define RSLINK_FREE_LIST(LINK)
                \
  {  struct RS_link *fl_link, *fl_link_next;
          \
     for (fl_link=(LINK); fl_link; fl_link=fl_link_next)
          \
       {                                                          \
          fl_link_next = fl_link->next;
          \
          RSLINK_FREE(fl_link);
          \
       }                                                          \
  }

/* initialize the free RS_LINK pool */
static void
rslink_init(int nlinks)                 /* total number of RS_LINK
available */
{
  int i;
  struct RS_link *link;

  rslink_free_list = NULL;
```

```
  for (i=0; i<nlinks; i++)
    {
      link = calloc(1, sizeof(struct RS_link));
      if (!link)
          fatal("out of virtual memory");
      link->next = rslink_free_list;
      rslink_free_list = link;
    }
}

/* service all functional unit release events, this function is called
   once per cycle, and it used to step the BUSY timers attached to
each
   functional unit in the function unit resource pool, as long as a
functional
   unit's BUSY count is > 0, it cannot be issued an operation */
static void
ruu_release_fu(void)
{
  int i;

  /* walk all resource units, decrement busy counts by one */
  for (i=0; i<fu_pool->num_resources; i++)
    {
      /* resource is released when BUSY hits zero */
      if (fu_pool->resources[i].busy > 0)
          fu_pool->resources[i].busy--;
    }
}



/*
 * the execution unit event queue implementation follows, the
event queue
 * indicates which instruction will complete next, the writeback
handler
 * drains this queue
 */

/* pending event queue, sorted from soonest to latest event (in
time), NOTE:
   RS_LINK nodes are used for the event queue list so that it need
not be
   updated during squash events */
static struct RS_link *event_queue;
```

```
/* initialize the event queue structures */
static void
eventq_init(void)
{
  event_queue = NULL;
}

/* dump the contents of the event queue */
static void
eventq_dump(FILE *stream)                    /* output stream */
{
  struct RS_link *ev;

  fprintf(stream, "** event queue state **\n");

  for (ev = event_queue; ev != NULL; ev = ev->next)
    {
      /* is event still valid? */
      if (RSLINK_VALID(ev))
          {
            struct RUU_station *rs = RSLINK_RS(ev);

            fprintf(stream, "idx: %2d: @ %.0f\n",
                    rs - (rs->in_LSQ ? LSQ : RUU), (double)ev-
>x.when);
            ruu_dumpent(rs, rs - (rs->in_LSQ ? LSQ : RUU),
                        stream, /* !header */FALSE);
          }
    }
}

/* insert an event for RS into the event queue, event queue is sorted
from
   earliest to latest event, event and associated side-effects will be
   apparent at the start of cycle WHEN */
static void
eventq_queue_event(struct RUU_station *rs, SS_TIME_TYPE
when)
{
  struct RS_link *prev, *ev, *new_ev;

  if (rs->completed)
    panic("event completed");

  if (when <= sim_cycle)
    panic("event occurred in the past");
```

```
  /* get a free event record */
  RSLINK_NEW(new_ev, rs);
  new_ev->x.when = when;

  /* locate insertion point */
  for (prev=NULL, ev=event_queue;
       ev && ev->x.when < when;
       prev=ev, ev=ev->next);

  if (prev)
    {
      /* insert middle or end */
      new_ev->next = prev->next;
      prev->next = new_ev;
    }
  else
    {
      /* insert at beginning */
      new_ev->next = event_queue;
      event_queue = new_ev;
    }
}

/* return the next event that has already occurred, returns NULL
when no
   remaining events or all remaining events are in the future */
static struct RUU_station *
eventq_next_event(void)
{
  struct RS_link *ev;

  if (event_queue && event_queue->x.when <= sim_cycle)
    {
      /* unlink and return first event on priority list */
      ev = event_queue;
      event_queue = event_queue->next;

      /* event still valid? */
      if (RSLINK_VALID(ev))
          {
            struct RUU_station *rs = RSLINK_RS(ev);

            /* reclaim event record */
            RSLINK_FREE(ev);
```

```
            /* event is valid, return resv station */
            return rs;
          }
      else
          {
            /* reclaim event record */
            RSLINK_FREE(ev);

            /* receiving inst was squashed, return next event */
            return eventq_next_event();
          }
    }
  else
    {
      /* no event or no event is ready */
      return NULL;
    }
}


/*
 * the ready instruction queue implementation follows, the ready
instruction
 * queue indicates which instruction have all of there *register*
dependencies
 * satisfied, instruction will issue when 1) all memory
dependencies for
 * the instruction have been satisfied (see lsq_refresh() for details
on how
 * this is accomplished) and 2) resources are available; ready queue
is fully
 * constructed each cycle before any operation is issued from it --
this
 * ensures that instruction issue priorities are properly observed;
NOTE:
 * RS_LINK nodes are used for the event queue list so that it need
not be
 * updated during squash events
 */

/* the ready instruction queue */
static struct RS_link *ready_queue;

/* initialize the event queue structures */
static void
readyq_init(void)
```

```
{
  ready_queue = NULL;
}

/* dump the contents of the ready queue */
static void
readyq_dump(FILE *stream)                       /* output stream */
{
  struct RS_link *link;

  fprintf(stream, "** ready queue state **\n");

  for (link = ready_queue; link != NULL; link = link->next)
    {
      /* is entry still valid? */
      if (RSLINK_VALID(link))
          {
            struct RUU_station *rs = RSLINK_RS(link);

            ruu_dumpent(rs, rs - (rs->in_LSQ ? LSQ : RUU),
                        stream, /* header */TRUE);
          }
    }
}

/* insert ready node into the ready list using ready instruction
scheduling
   policy; currently the following scheduling policy is enforced:

     memory and long latency operands, and branch instructions first

   then

     all other instructions, oldest instructions first

  this policy works well because branches pass through the
machine quicker
  which works to reduce branch misprediction latencies, and very
long latency
  instructions (such loads and multiplies) get priority since they are
very
  likely on the program's critical path */
static void
readyq_enqueue(struct RUU_station *rs)                  /* RS to
enqueue */
{
```

```
  struct RS_link *prev, *node, *new_node;

  /* node is now queued */
  if (rs->queued)
    panic("node is already queued");
  rs->queued = TRUE;

  /* get a free ready list node */
  RSLINK_NEW(new_node, rs);
  new_node->x.seq = rs->seq;

  /* locate insertion point */
  if (rs->in_LSQ || SS_OP_FLAGS(rs->op) &
(F_LONGLAT|F_CTRL))
    {
      /* insert loads/stores and long latency ops at the head of the
queue */
      prev = NULL;
      node = ready_queue;
    }
  else
    {
      /* otherwise insert in program order (earliest seq first) */
      for (prev=NULL, node=ready_queue;
           node && node->x.seq < rs->seq;
           prev=node, node=node->next);
    }

  if (prev)
    {
      /* insert middle or end */
      new_node->next = prev->next;
      prev->next = new_node;
    }
  else
    {
      /* insert at beginning */
      new_node->next = ready_queue;
      ready_queue = new_node;
    }
}


/*
 * the create vector maps a logical register to a creator in the RUU
(and
```

```
 * specific output operand) or the architected register file (if
RS_link
 * is NULL)
 */

/* an entry in the create vector */
struct CV_link {
  struct RUU_station *rs;             /* creator's reservation station */
  int odep_num;                       /* specific output operand */
};

/* a NULL create vector entry */
static struct CV_link CVLINK_NULL = { NULL, 0 };

/* get a new create vector link */
#define CVLINK_INIT(CV, RS,ONUM)      ((CV).rs = (RS),
(CV).odep_num = (ONUM))

/* size of the create vector (one entry per architected register) */
#define CV_BMAP_SZ
(BITMAP_SIZE(SS_TOTAL_REGS))

/* the create vector, NOTE: speculative copy on write storage
provided
   for fast recovery during wrong path execute (see tracer_recover()
for
   details on this process */
static BITMAP_TYPE(SS_TOTAL_REGS, use_spec_cv);
static struct CV_link create_vector[SS_TOTAL_REGS];
static struct CV_link spec_create_vector[SS_TOTAL_REGS];

/* these arrays shadow the create vector an indicate when a register
was
   last created */
static SS_TIME_TYPE create_vector_rt[SS_TOTAL_REGS];
static SS_TIME_TYPE
spec_create_vector_rt[SS_TOTAL_REGS];

/* read a create vector entry */
#define CREATE_VECTOR(N)
(BITMAP_SET_P(use_spec_cv, CV_BMAP_SZ, (N))\
                                    ? spec_create_vector[N]             \
                                    : create_vector[N])

/* read a create vector timestamp entry */
#define CREATE_VECTOR_RT(N)
```

```
(BITMAP_SET_P(use_spec_cv, CV_BMAP_SZ, (N))\
                                        ? spec_create_vector_rt[N]          \
                                        : create_vector_rt[N])

/* set a create vector entry */
#define SET_CREATE_VECTOR(N, L) (spec_mode
\
                                        ? (BITMAP_SET(use_spec_cv,
CV_BMAP_SZ, (N)),\
                                           spec_create_vector[N] = (L))
\
                                        : (create_vector[N] = (L)))

/* initialize the create vector */
static void
cv_init(void)
{
  int i;

  /* initially all registers are valid in the architected register file,
     i.e., the create vector entry is CVLINK_NULL */
  for (i=0; i<SS_TOTAL_REGS; i++)
    {
      create_vector[i] = CVLINK_NULL;
      create_vector_rt[i] = 0;
      spec_create_vector[i] = CVLINK_NULL;
      spec_create_vector_rt[i] = 0;
    }

  /* all create vector entries are non-speculative */
  BITMAP_CLEAR_MAP(use_spec_cv, CV_BMAP_SZ);
}

/* dependency index names */
static char *dep_names[SS_TOTAL_REGS] = {
  "n/a", "$r1", "$r2", "$r3", "$r4", "$r5", "$r6", "$r7", "$r8",  "$r9",
  "$r10", "$r11", "$r12", "$r13", "$r14", "$r15", "$r16", "$r17",
"$r18",
  "$r19", "$r20", "$r21", "$r22", "$r23", "$r24", "$r25", "$r26",
"$r27",
  "$r28",  "$r29", "$r30", "$r31",
  "$f0", "$f1", "$f2", "$f3", "$f4", "$f5", "$f6", "$f7", "$f8",  "$f9",
  "$f10", "$f11", "$f12", "$f13", "$f14", "$f15", "$f16", "$f17",
"$f18",
  "$f19", "$f20", "$f21", "$f22", "$f23", "$f24", "$f25", "$f26",
"$f27",
```

```
  "$f28",  "$f29", "$f30", "$f31",
  "$hi", "$lo", "$fcc", "$tmp",
  "n/a", "n/a"
};

/* dump the contents of the create vector */
static void
cv_dump(FILE *stream)                                   /* output
stream */
{
  int i;
  struct CV_link ent;

  fprintf(stream, "** create vector state **\n");

  for (i=0; i<SS_TOTAL_REGS; i++)
    {
      ent = CREATE_VECTOR(i);
      if (!ent.rs)
          fprintf(stream, "[%4s]: from architected reg file\n",
dep_names[i]);
      else
          fprintf(stream, "[%4s]: from %s, idx: %d\n",
              dep_names[i], (ent.rs->in_LSQ ? "LSQ" : "RUU"),
                  ent.rs - (ent.rs->in_LSQ ? LSQ : RUU));
    }
}


/*
 *  RUU_COMMIT() - instruction retirement pipeline stage
 */

/* this function commits the results of the oldest completed entries
from the
   RUU and LSQ to the architected reg file, stores in the LSQ will
commit
   their store data to the data cache at this point as well */
static void
ruu_commit(void)
{
  int i, lat, events, committed = 0;

  /* all values must be retired to the architected reg file in program
order */
  while (RUU_num > 0 && committed < ruu_commit_width)
```

```
    {
      struct RUU_station *rs = &(RUU[RUU_head]);

      if (!rs->completed)
        {
          /* at least RUU entry must be complete */
          break;
        }

      /* default commit events */
      events = 0;

      /* load/stores must retire load/store queue entry as well */
      if (RUU[RUU_head].ea_comp)
        {
          /* load/store, retire head of LSQ as well */
          if (LSQ_num <= 0 || !LSQ[LSQ_head].in_LSQ)
            panic("RUU out of sync with LSQ");

          /* load/store operation must be complete */
          if (!LSQ[LSQ_head].completed)
            {
              /* load/store operation is not yet complete */
              break;
            }

          if ((SS_OP_FLAGS(LSQ[LSQ_head].op) &
(F_MEM|F_STORE))
              == (F_MEM|F_STORE))
            {
              struct res_template *fu;


              /* stores must retire their store value to the cache at
commit,
                 try to get a store port (functional unit allocation) */
              fu = res_get(fu_pool,
SS_OP_FUCLASS(LSQ[LSQ_head].op));
              if (fu)
                {
                  /* reserve the functional unit */
                  if (fu->master->busy)
                    panic("functional unit already in use");

                  /* schedule functional unit release event */
                  fu->master->busy = fu->issuelat;
```

```
                    /* go to the data cache */
                    if (cache_dl1)
                      {
                        /* commit store value to D-cache */
                        lat =
                            cache_access(cache_dl1, Write,
(LSQ[LSQ_head].addr&~3),
                                         NULL, 4, sim_cycle, NULL,
NULL);
                        if (lat > cache_dl1_lat)
                            events |= PEV_CACHEMISS;
                      }

                    /* all loads and stores must to access D-TLB */
                    if (dtlb)
                      {
                        /* access the D-TLB */
                        lat =
                            cache_access(dtlb, Read,
(LSQ[LSQ_head].addr & ~3),
                                         NULL, 4, sim_cycle, NULL,
NULL);
                        if (lat > 1)
                            events |= PEV_TLBMISS;
                      }
                  }
                else
                  {
                    /* no store ports left, cannot continue to commit
insts */
                    break;
                  }
              }

            /* invalidate load/store operation instance */
            LSQ[LSQ_head].tag++;

            /* indicate to pipeline trace that this instruction retired */
            ptrace_newstage(LSQ[LSQ_head].ptrace_seq,
PST_COMMIT, events);
            ptrace_endinst(LSQ[LSQ_head].ptrace_seq);

            /* commit head of LSQ as well */
            LSQ_head = (LSQ_head + 1) % LSQ_size;
            LSQ_num--;
```

```
        }

      if (pred
          && bpred_spec_update == spec_CT
          && (SS_OP_FLAGS(rs->op) & F_CTRL))
        {
          SS_INST_TYPE inst = rs->IR;

          bpred_update(pred,
                       /* branch address */rs->PC,
                       /* actual target address */rs->next_PC,
                  /* taken? */rs->next_PC != (rs->PC +
                                           sizeof(SS_INST_TYPE)),
                  /* pred taken? */rs->pred_PC != (rs->PC +
                                               sizeof(SS_INST_TYPE)),
                  /* correct pred? */rs->pred_PC == rs->next_PC,
                  /* opcode */rs->op,
                       /* jump through R31? */(RS) == 31,
                  /* dir predictor update pointer */&rs->dir_update,
                  /* instruction for FFBPANN */ inst);

        }

      /* invalidate RUU operation instance */
      RUU[RUU_head].tag++;

      /* indicate to pipeline trace that this instruction retired */
      ptrace_newstage(RUU[RUU_head].ptrace_seq,
PST_COMMIT, events);
      ptrace_endinst(RUU[RUU_head].ptrace_seq);

      /* commit head entry of RUU */
      RUU_head = (RUU_head + 1) % RUU_size;
      RUU_num--;

      /* one more instruction committed to architected state */
      committed++;

      for (i=0; i<MAX_ODEPS; i++)
        {
          if (rs->odep_list[i])
            panic ("retired instruction has odeps\n");
        }
    }
}
```

```
/*
 *  RUU_RECOVER() - squash mispredicted microarchitecture
state
 */

/* recover processor microarchitecture state back to point of the
   mis-predicted branch at RUU[BRANCH_INDEX] */
static void
ruu_recover(int branch_index)                    /* index of
mis-pred branch */
{
  int i, RUU_index = RUU_tail, LSQ_index = LSQ_tail;
  int RUU_prev_tail = RUU_tail, LSQ_prev_tail = LSQ_tail;

  /* recover from the tail of the RUU towards the head until the
branch index
     is reached, this direction ensures that the LSQ can be
synchronized with
     the RUU */

  /* go to first element to squash */
  RUU_index = (RUU_index + (RUU_size-1)) % RUU_size;
  LSQ_index = (LSQ_index + (LSQ_size-1)) % LSQ_size;

  /* traverse to older insts until the mispredicted branch is
encountered */
  while (RUU_index != branch_index)
    {
      /* the RUU should not drain since the mispredicted branch will
remain */
      if (!RUU_num)
          panic("empty RUU");

      /* should meet up with the tail first */
      if (RUU_index == RUU_head)
          panic("RUU head and tail broken");

      /* is this operation an effective addr calc for a load or store? */
      if (RUU[RUU_index].ea_comp)
          {
            /* should be at least one load or store in the LSQ */
            if (!LSQ_num)
              panic("RUU and LSQ out of sync");

            /* recover any resources consumed by the load or store
```

```
operation */
          for (i=0; i<MAX_ODEPS; i++)
            {
              RSLINK_FREE_LIST(LSQ[LSQ_index].odep_list[i]);
              /* blow away the consuming op list */
              LSQ[LSQ_index].odep_list[i] = NULL;
            }

          /* squash this LSQ entry */
          LSQ[LSQ_index].tag++;

          /* indicate in pipetrace that this instruction was squashed
*/
          ptrace_endinst(LSQ[LSQ_index].ptrace_seq);

          /* go to next earlier LSQ slot */
          LSQ_prev_tail = LSQ_index;
          LSQ_index = (LSQ_index + (LSQ_size-1)) % LSQ_size;
          LSQ_num--;
        }

      /* recover any resources used by this RUU operation */
      for (i=0; i<MAX_ODEPS; i++)
          {
            RSLINK_FREE_LIST(RUU[RUU_index].odep_list[i]);
            /* blow away the consuming op list */
            RUU[RUU_index].odep_list[i] = NULL;
          }

      /* squash this RUU entry */
      RUU[RUU_index].tag++;

      /* indicate in pipetrace that this instruction was squashed */
      ptrace_endinst(RUU[RUU_index].ptrace_seq);

      /* go to next earlier slot in the RUU */
      RUU_prev_tail = RUU_index;
      RUU_index = (RUU_index + (RUU_size-1)) % RUU_size;
      RUU_num--;
    }

  /* reset head/tail pointers to point to the mis-predicted branch */
  RUU_tail = RUU_prev_tail;
  LSQ_tail = LSQ_prev_tail;

  /* revert create vector back to last precise create vector state,
```

```
NOTE:
    this is accomplished by resetting all the copied-on-write bits in
the
    USE_SPEC_CV bit vector */
  BITMAP_CLEAR_MAP(use_spec_cv, CV_BMAP_SZ);

  /* FIXME: could reset functional units at squash time */
}


/*
 *  RUU_WRITEBACK() - instruction result writeback pipeline
stage
 */

/* forward declarations */
static void tracer_recover(void);

/* writeback completed operation results from the functional units
to RUU,
   at this point, the output dependency chains of completing
instructions
   are also walked to determine if any dependent instruction now
has all
   of its register operands, if so the (nearly) ready instruction is
inserted
   into the ready instruction queue */
static void
ruu_writeback(void)
{
  int i;
  struct RUU_station *rs;

  /* service all completed events */
  while ((rs = eventq_next_event()))
    {
      /* RS has completed execution and (possibly) produced a result
*/
      if (!OPERANDS_READY(rs) || rs->queued || !rs->issued || rs-
>completed)
          panic("inst completed and !ready, !issued, or completed");

      /* operation has completed */
      rs->completed = TRUE;

      /* does this operation reveal a mis-predicted branch? */
```

```
    if (rs->recover_inst)
        {
          if (rs->in_LSQ)
            panic("mis-predicted load or store?!?!?");

          /* recover processor state and reinit fetch to correct path
*/
          ruu_recover(rs - RUU);
          tracer_recover();
          bpred_recover(pred, rs->PC, rs->stack_recover_idx);

          /* stall fetch until I-fetch and I-decode recover */
          ruu_fetch_issue_delay = ruu_branch_penalty;

          /* continue writeback of the branch/control instruction */
        }

    /* if we speculatively update branch-predictor, do it here */
    if (pred
          && bpred_spec_update == spec_WB
          && !rs->in_LSQ
          && (SS_OP_FLAGS(rs->op) & F_CTRL))
        {
          SS_INST_TYPE inst = rs->IR;

          bpred_update(pred,
                       /* branch address */rs->PC,
                       /* actual target address */rs->next_PC,
                       /* taken? */rs->next_PC != (rs->PC +

sizeof(SS_INST_TYPE)),
                       /* pred taken? */rs->pred_PC != (rs->PC +

        sizeof(SS_INST_TYPE)),
                       /* correct pred? */rs->pred_PC == rs-
>next_PC,
                       /* opcode */rs->op,
                       /* jump through R31? */(RS) == 31,
                       /* dir predictor update pointer */&rs-
>dir_update,
                  /* instruction for FFBPANN */ inst);

        }

    /* entered writeback stage, indicate in pipe trace */
    ptrace_newstage(rs->ptrace_seq, PST_WRITEBACK,
```

```
                    rs->recover_inst ? PEV_MPDETECT : 0);

    /* broadcast results to consuming operations, this is more
efficiently
       accomplished by walking the output dependency chains of
the
          completed instruction */
    for (i=0; i<MAX_ODEPS; i++)
        {
         if (rs->onames[i] != NA)
           {
             struct CV_link link;
             struct RS_link *olink, *olink_next;

             if (rs->spec_mode)
                 {
                  /* update the speculative create vector, future
operations
                     get value from later creator or architected reg file
*/
                  link = spec_create_vector[rs->onames[i]];
                  if (/* !NULL */link.rs
                     && /* refs RS */(link.rs == rs &&
link.odep_num == i))
                    {
                     /* the result can now be read from a physical
register,
                           indicate this as so */
                     spec_create_vector[rs->onames[i]] =
CVLINK_NULL;
                     spec_create_vector_rt[rs->onames[i]] =
sim_cycle;
                    }
                  /* else, creator invalidated or there is another
creator */
                 }
             else
                 {
                  /* update the non-speculative create vector, future
                     operations get value from later creator or
architected
                     reg file */
                  link = create_vector[rs->onames[i]];
                  if (/* !NULL */link.rs
                     && /* refs RS */(link.rs == rs &&
link.odep_num == i))
```

```
                    {
                      /* the result can now be read from a physical
register,
                         indicate this as so */
                      create_vector[rs->onames[i]] =
CVLINK_NULL;
                      create_vector_rt[rs->onames[i]] = sim_cycle;
                    }
                  /* else, creator invalidated or there is another
creator */
                }

              /* walk output list, queue up ready operations */
              for (olink=rs->odep_list[i]; olink; olink=olink_next)
                  {
                   if (RSLINK_VALID(olink))
                    {
                      if (olink->rs->idep_ready[olink->x.opnum])
                          panic("output dependence already
satisfied");

                      /* input is now ready */
                      olink->rs->idep_ready[olink->x.opnum] =
TRUE;

                      /* are all the register operands of target ready?
*/
                      if (OPERANDS_READY(olink->rs))
                          {
                            /* yes! enqueue instruction as ready,
NOTE: stores
                               complete at dispatch, so no need to
enqueue
                               them */
                            if (!olink->rs->in_LSQ
                               || ((SS_OP_FLAGS(olink->rs-
>op)&(F_MEM|F_STORE))
                                   == (F_MEM|F_STORE)))
                              readyq_enqueue(olink->rs);
                            /* else, ld op, issued when no mem conflict
*/
                          }
                    }

                  /* grab link to next element prior to free */
                  olink_next = olink->next;
```

```
                    /* free dependence link element */
                    RSLINK_FREE(olink);
                  }
                /* blow away the consuming op list */
                rs->odep_list[i] = NULL;

              } /* if not NA output */

            } /* for all outputs */

  } /* for all writeback events */

}


/*
 *  LSQ_REFRESH() - memory access dependence
checker/scheduler
 */

/* this function locates ready instructions whose memory
dependencies have
  been satisfied, this is accomplished by walking the LSQ for
loads, looking
  for blocking memory dependency condition (e.g., earlier store
with an
  unknown address) */
#define MAX_STD_UNKNOWNS              64
static void
lsq_refresh(void)
{
 int i, j, index, n_std_unknowns;
 SS_ADDR_TYPE std_unknowns[MAX_STD_UNKNOWNS];

 /* scan entire queue for ready loads: scan from oldest instruction
    (head) until we reach the tail or an unresolved store, after which
no
    other instruction will become ready */
 for (i=0, index=LSQ_head, n_std_unknowns=0;
      i < LSQ_num;
      i++, index=(index + 1) % LSQ_size)
   {
     /* terminate search for ready loads after first unresolved store,
          as no later load could be resolved in its presence */
     if (/* store? */
```

```
          (SS_OP_FLAGS(LSQ[index].op) &
(F_MEM|F_STORE)) == (F_MEM|F_STORE))
        {
          if (!STORE_ADDR_READY(&LSQ[index]))
            {
              /* FIXME: a later STD + STD known could hide the
STA unknown */
              /* sta unknown, blocks all later loads, stop search */
              break;
            }
          else if (!OPERANDS_READY(&LSQ[index]))
            {
              /* sta known, but std unknown, may block a later store,
record
                 this address for later referral, we use an array here
because
                 for most simulations the number of entries to
search will be
                 very small */
              if (n_std_unknowns == MAX_STD_UNKNOWNS)
                fatal("STD unknown array overflow, increase
MAX_STD_UNKNOWNS");
              std_unknowns[n_std_unknowns++] = LSQ[index].addr;
            }
          else /* STORE_ADDR_READY() &&
OPERANDS_READY() */
            {
              /* a later STD known hides an earlier STD unknown */
              for (j=0; j<n_std_unknowns; j++)
                {
                  if (std_unknowns[j] == /* STA/STD known
*/LSQ[index].addr)
                    std_unknowns[j] = /* bogus addr */0;
                }
            }
        }

    if (/* load? */
          ((SS_OP_FLAGS(LSQ[index].op) &
(F_MEM|F_LOAD)) == (F_MEM|F_LOAD))
          && /* queued? */!LSQ[index].queued
          && /* waiting? */!LSQ[index].issued
          && /* completed? */!LSQ[index].completed
          && /* regs ready?
*/OPERANDS_READY(&LSQ[index]))
        {
```

```
            /* no STA unknown conflict (because we got to this
check), check for
               a STD unknown conflict */
            for (j=0; j<n_std_unknowns; j++)
              {
               /* found a relevant STD unknown? */
               if (std_unknowns[j] == LSQ[index].addr)
                    break;
              }
            if (j == n_std_unknowns)
              {
               /* no STA or STD unknown conflicts, put load on
ready queue */
               readyq_enqueue(&LSQ[index]);
              }
          }
    }
}


/*
 *  RUU_ISSUE() - issue instructions to functional units
 */

/* attempt to issue all operations in the ready queue; insts in the
ready
   instruction queue have all register dependencies satisfied, this
function
   must then 1) ensure the instructions memory dependencies have
been satisfied
   (see lsq_refresh() for details on this process) and 2) a function
unit
   is available in this cycle to commence execution of the operation;
if all
   goes well, the function unit is allocated, a writeback event is
scheduled,
   and the instruction begins execution */
static void
ruu_issue(void)
{
  int i, load_lat, tlb_lat, n_issued;
  struct RS_link *node, *next_node;
  struct res_template *fu;

  /* FIXME: could be a little more efficient when scanning the
ready queue */
```

```
  /* copy and then blow away the ready list, NOTE: the ready list is
     always totally reclaimed each cycle, and instructions that are
not
     issue are explicitly reinserted into the ready instruction queue,
     this management strategy ensures that the ready instruction
queue
     is always properly sorted */
  node = ready_queue;
  ready_queue = NULL;

  /* visit all ready instructions (i.e., insts whose register input
     dependencies have been satisfied, stop issue when no more
instructions
     are available or issue bandwidth is exhausted */
  for (n_issued=0;
       node && n_issued < ruu_issue_width;
       node = next_node)
    {
      next_node = node->next;

      /* still valid? */
      if (RSLINK_VALID(node))
          {
            struct RUU_station *rs = RSLINK_RS(node);

            /* issue operation, both reg and mem deps have been
satisfied */
            if (!OPERANDS_READY(rs) || !rs->queued
                || rs->issued || rs->completed)
              panic("issued inst !ready, issued, or completed");

            /* node is now un-queued */
            rs->queued = FALSE;

            if (rs->in_LSQ
                && ((SS_OP_FLAGS(rs->op) & (F_MEM|F_STORE))
== (F_MEM|F_STORE)))
              {
                /* stores complete in effectively zero time, result is
                     written into the load/store queue, the actual store
into
                     the memory system occurs when the instruction is
retired
                     (see ruu_commit()) */
                rs->issued = TRUE;
```

```
              rs->completed = TRUE;

              if (rs->onames[0] || rs->onames[1])
                  panic("store creates result");

              if (rs->recover_inst)
                  panic("mis-predicted store");

              /* entered execute stage, indicate in pipe trace */
              ptrace_newstage(rs->ptrace_seq, PST_WRITEBACK,
0);

              /* one more inst issued */
              n_issued++;
            }
          else
            {
              /* issue the instruction to a functional unit */
              if (SS_OP_FUCLASS(rs->op) != NA)
                  {
                    fu = res_get(fu_pool, SS_OP_FUCLASS(rs->op));
                    if (fu)
                      {
                        /* got one! issue inst to functional unit */
                        rs->issued = TRUE;

                        /* reserve the functional unit */
                        if (fu->master->busy)
                            panic("functional unit already in use");

                        /* schedule functional unit release event */
                        fu->master->busy = fu->issuelat;

                        /* schedule a result writeback event */
                        if (rs->in_LSQ
                              && ((SS_OP_FLAGS(rs->op) &
(F_MEM|F_LOAD))
                                  == (F_MEM|F_LOAD)))
                            {
                              int events = 0;

                              /* for loads, determine cache access
latency:
                                 first scan LSQ to see if a store forward is
                                 possible, if not, access the data cache */
                              load_lat = 0;
```

```
                    i = (rs - LSQ);
                    if (i != LSQ_head)
                      {
                        for (;;)
                            {
                              /* go to next earlier LSQ entry */
                              i = (i + (LSQ_size-1)) % LSQ_size;

                              /* FIXME: not dealing with
partials! */
                              if ((SS_OP_FLAGS(LSQ[i].op) &
F_STORE)
                                  && (LSQ[i].addr == rs->addr))
                                {
                                  /* hit in the LSQ */
                                  load_lat = 1;
                                  break;
                                }

                              /* scan finished? */
                              if (i == LSQ_head)
                                break;
                            }
                      }

                    /* was the value store forwared from the
LSQ? */
                    if (!load_lat)
                      {
                        /* no! go to the data cache */
                        if (cache_dl1
                              /* valid address? */
                              && (rs->addr >= ld_data_base
                                  && rs->addr < ld_stack_base))
                            {
                              /* access the cache if non-faulting
*/
                              load_lat =
                                cache_access(cache_dl1, Read,
                                             (rs->addr & ~3),
NULL, 4,
                                             sim_cycle, NULL,
NULL);
                              if (load_lat > cache_dl1_lat)
                                events |= PEV_CACHEMISS;
                            }
```

```
                    else
                        {
                          /* no caches defined, just use op
latency */
                          load_lat = fu->oplat;
                        }
                    }

                  /* all loads and stores must to access D-
TLB */
                  if (dtlb
                    /* valid address? */
                    && (rs->addr >= ld_data_base
                        && rs->addr < ld_stack_base))
                    {
                      /* access the D-DLB, NOTE: this code
will
                         initiate speculative TLB misses */
                      tlb_lat =
                        cache_access(dtlb, Read, (rs->addr
& ~3),
                                     NULL, 4, sim_cycle,
NULL, NULL);
                      if (tlb_lat > 1)
                        events |= PEV_TLBMISS;

                      /* D-cache/D-TLB accesses occur in
parallel */
                      load_lat = MAX(tlb_lat, load_lat);
                    }

                  /* use computed cache access latency */
                  eventq_queue_event(rs, sim_cycle +
load_lat);

                  /* entered execute stage, indicate in pipe
trace */
                  ptrace_newstage(rs->ptrace_seq,
PST_EXECUTE,
                                  ((rs->ea_comp ?
PEV_AGEN : 0)
                                   | events));
                }
              else /* !load && !store */
                {
                  /* use deterministic functional unit latency
```

```
*/
                                eventq_queue_event(rs, sim_cycle + fu-
>oplat);

                                /* entered execute stage, indicate in pipe
trace */
                                ptrace_newstage(rs->ptrace_seq,
PST_EXECUTE,
                                                rs->ea_comp ? PEV_AGEN
: 0);
                              }

                            /* one more inst issued */
                            n_issued++;
                          }
                        else /* no functional unit */
                          {
                            /* insufficient functional unit resources, put
operation
                               back onto the ready list, we'll try to issue it
                               again next cycle */
                            readyq_enqueue(rs);
                          }
                      }
                  else /* does not require a functional unit! */
                      {
                        /* FIXME: need better solution for these */
                        /* the instruction does not need a functional unit */
                        rs->issued = TRUE;

                        /* schedule a result event */
                        eventq_queue_event(rs, sim_cycle + 1);

                        /* entered execute stage, indicate in pipe trace */
                        ptrace_newstage(rs->ptrace_seq, PST_EXECUTE,
                                        rs->ea_comp ? PEV_AGEN : 0);

                        /* one more inst issued */
                        n_issued++;
                      }
                } /* !store */

            }
        /* else, RUU entry was squashed */

        /* reclaim ready list entry, NOTE: this is done whether or not
```

```
the
        instruction issued, since the instruction was once again
reinserted
        into the ready queue if it did not issue, this ensures that the
ready
        queue is always properly sorted */
      RSLINK_FREE(node);
    }

  /* put any instruction not issued back into the ready queue, go
through
     normal channels to ensure instruction stay ordered correctly */
  for (; node; node = next_node)
    {
      next_node = node->next;

      /* still valid? */
      if (RSLINK_VALID(node))
        {
            struct RUU_station *rs = RSLINK_RS(node);

          /* node is now un-queued */
          rs->queued = FALSE;

            /* not issued, put operation back onto the ready list, we'll
try to
               issue it again next cycle */
          readyq_enqueue(rs);
        }
      /* else, RUU entry was squashed */

      /* reclaim ready list entry, NOTE: this is done whether or not
the
        instruction issued, since the instruction was once again
reinserted
        into the ready queue if it did not issue, this ensures that the
ready
        queue is always properly sorted */
      RSLINK_FREE(node);
    }
}


/*
 * routines for generating on-the-fly instruction traces with support
 * for control and data misspeculation modeling
```

```
 */

/* integer register file */
#define R_BMAP_SZ       (BITMAP_SIZE(SS_NUM_REGS))
static BITMAP_TYPE(SS_NUM_REGS, use_spec_R);
static SS_WORD_TYPE spec_regs_R[SS_NUM_REGS];

/* floating point register file */
#define F_BMAP_SZ       (BITMAP_SIZE(SS_NUM_REGS))
static BITMAP_TYPE(SS_NUM_REGS, use_spec_F);
static union regs_FP spec_regs_F;

 /* miscellaneous registers */
static int use_spec_HI;
static SS_WORD_TYPE spec_regs_HI;
static int use_spec_LO;
static SS_WORD_TYPE spec_regs_LO;
static int use_spec_FCC;
static int spec_regs_FCC;

/* dump speculative register state */
static void
rspec_dump(FILE *stream)                        /* output stream */
{
  int i;

  fprintf(stream, "** speculative register contents **\n");

  fprintf(stream, "spec_mode: %s\n", spec_mode ? "t" : "f");

  /* dump speculative integer regs */
  for (i=0; i<SS_NUM_REGS; i++)
    {
      if (BITMAP_SET_P(use_spec_R, R_BMAP_SZ, i))
          {
            /* speculative state */
            fprintf(stream, "[%4s]: %12d/0x%08x\n", dep_names[i],
                    spec_regs_R[i], spec_regs_R[i]);
          }
    }

  /* dump speculative FP regs */
  for (i=0; i<SS_NUM_REGS; i++)
    {
      if (BITMAP_SET_P(use_spec_F, F_BMAP_SZ, i))
          {
```

```
          /* speculative state */
          fprintf(stream,
                  "[%4s]: %12d/0x%08x/%f ([%4s] as double:
%f)\n\n",
                  dep_names[i+32],
                  spec_regs_F.l[i], spec_regs_F.l[i],
spec_regs_F.f[i],
                  dep_names[i+32],
                  spec_regs_F.d[i >> 1]);
        }
    }

  /* dump speculative miscellaneous regs */
  if (use_spec_HI)
    fprintf(stream, "[ $hi]: %12d/0x%08x\n", spec_regs_HI,
spec_regs_HI);
  if (use_spec_LO)
    fprintf(stream, "[ $lo]: %12d/0x%08x\n", spec_regs_LO,
spec_regs_LO);
  if (use_spec_FCC)
    fprintf(stream, "[$fcc]: 0x%08x\n", spec_regs_FCC);
}


/* speculative memory hash table size, NOTE: this must be a
power-of-two */
#define STORE_HASH_SIZE                 32

/* speculative memory hash table definition, accesses go through
this hash
   table when accessing memory in speculative mode, the hash
table flush the
   table when recovering from mispredicted branches */
struct spec_mem_ent {
  struct spec_mem_ent *next;            /* ptr to next hash table
bucket */
  SS_ADDR_TYPE addr;                    /* virtual address of
spec state */
  unsigned int data[2];                 /* spec buffer, up to 8 bytes
*/
};

/* speculative memory hash table */
static struct spec_mem_ent *store_htable[STORE_HASH_SIZE];

/* speculative memory hash table bucket free list */
```

```
static struct spec_mem_ent *bucket_free_list = NULL;


/* program counter */
static SS_ADDR_TYPE pred_PC;
static SS_ADDR_TYPE recover_PC;

/* fetch unit next fetch address */
static SS_ADDR_TYPE fetch_regs_PC;
static SS_ADDR_TYPE fetch_pred_PC;

/* IFETCH -> DISPATCH instruction queue definition */
struct fetch_rec {
  SS_INST_TYPE IR;                  /* inst register */
  SS_ADDR_TYPE regs_PC, pred_PC;        /* current PC,
predicted next PC */
  struct bpred_update dir_update;       /* bpred direction update info
*/
  int stack_recover_idx;                /* branch predictor RSB
index */
  unsigned int ptrace_seq;              /* print trace sequence id */
};
static struct fetch_rec *fetch_data;    /* IFETCH -> DISPATCH
inst queue */
static int fetch_num;                   /* num entries in IF -> DIS
queue */
static int fetch_tail, fetch_head;        /* head and tail pointers of
queue */

/* recover instruction trace generator state to precise state state
immediately
   before the first mis-predicted branch; this is accomplished by
resetting
   all register value copied-on-write bitmasks are reset, and the
speculative
   memory hash table is cleared */
static void
tracer_recover(void)
{
  int i;
  struct spec_mem_ent *ent, *ent_next;

  /* better be in mis-speculative trace generation mode */
  if (!spec_mode)
    panic("cannot recover unless in speculative mode");
```

```
  /* reset to non-speculative trace generation mode */
  spec_mode = FALSE;

  /* reset copied-on-write register bitmasks back to non-speculative
state */
  BITMAP_CLEAR_MAP(use_spec_R, R_BMAP_SZ);
  BITMAP_CLEAR_MAP(use_spec_F, F_BMAP_SZ);
  use_spec_HI = FALSE;
  use_spec_LO = FALSE;
  use_spec_FCC = FALSE;

  /* reset memory state back to non-speculative state */
  /* FIXME: could version stamps be used here?!?!? */
  for (i=0; i<STORE_HASH_SIZE; i++)
    {
      /* release all hash table buckets */
      for (ent=store_htable[i]; ent; ent=ent_next)
          {
            ent_next = ent->next;
            ent->next = bucket_free_list;
            bucket_free_list = ent;
          }
      store_htable[i] = NULL;
    }

  /* if pipetracing, indicate squash of instructions in the inst fetch
queue */
  if (ptrace_active)
    {
      while (fetch_num != 0)
          {
            /* squash the next instruction from the IFETCH ->
DISPATCH queue */
            ptrace_endinst(fetch_data[fetch_head].ptrace_seq);

            /* consume instruction from IFETCH -> DISPATCH
queue */
            fetch_head = (fetch_head+1) & (ruu_ifq_size - 1);
            fetch_num--;
          }
    }

  /* reset IFETCH state */
  fetch_num = 0;
  fetch_tail = fetch_head = 0;
  fetch_pred_PC = fetch_regs_PC = recover_PC;
```

```
}

/* initialize the speculative instruction state generator state */
static void
tracer_init(void)
{
  int i;

  /* initially in non-speculative mode */
  spec_mode = FALSE;

  /* register state is from non-speculative state buffers */
  BITMAP_CLEAR_MAP(use_spec_R, R_BMAP_SZ);
  BITMAP_CLEAR_MAP(use_spec_F, F_BMAP_SZ);
  use_spec_HI = FALSE;
  use_spec_LO = FALSE;
  use_spec_FCC = FALSE;

  /* memory state is from non-speculative memory pages */
  for (i=0; i<STORE_HASH_SIZE; i++)
    store_htable[i] = NULL;
}


/* speculative memory hash table address hash function */
#define HASH_ADDR(ADDR)
                    \
  ((((ADDR) >> 24)^((ADDR) >> 16)^((ADDR) >> 8)^(ADDR))
& (STORE_HASH_SIZE-1))

/* this functional provides a layer of mis-speculated state over the
   non-speculative memory state, when in mis-speculation trace
generation mode,
   the simulator will call this function to access memory, instead of
the
   non-speculative memory access interfaces defined in memory.h;
when storage
   is written, an entry is allocated in the speculative memory hash
table,
   future reads and writes while in mis-speculative trace generation
mode will
   access this buffer instead of non-speculative memory state; when
the trace
   generator transitions back to non-speculative trace generation
mode,
   tracer_recover() clears this table */
```

```
static void
spec_mem_access(enum mem_cmd cmd,               /* Read or
Write access cmd */
                SS_ADDR_TYPE addr,              /* virtual
address of access */
                void *p,                        /* input/output buffer
*/
                int nbytes)                     /* number of bytes to
access */
{
  int index;
  struct spec_mem_ent *ent, *prev;

  /* FIXME: partially overlapping writes are not combined... */
  /* FIXME: partially overlapping reads are not handled correctly...
*/

  /* check alignments, even speculative this test should always pass
*/
  if ((nbytes & (nbytes-1)) != 0 || (addr & (nbytes-1)) != 0)
    {
      /* no can do */
      return;
    }

  /* check permissions */
  if (!((addr >= ld_text_base && addr <
(ld_text_base+ld_text_size)
           && cmd == Read)
          || (addr >= ld_data_base && addr < ld_stack_base)))
    {
      /* no can do */
      return;
    }

  /* has this memory state been copied on mis-speculative write? */
  index = HASH_ADDR(addr);
  for (prev=NULL,ent=store_htable[index]; ent; prev=ent,ent=ent-
>next)
    {
      if (ent->addr == addr)
          {
            /* reorder chains to speed access into hash table */
            if (prev != NULL)
              {
                /* not at head of list, relink the hash table entry at front
```

```
*/
                prev->next = ent->next;
              ent->next = store_htable[index];
              store_htable[index] = ent;
                }
              break;
            }
    }

  /* no, if it is a write, allocate a hash table entry to hold the data */
  if (!ent && cmd == Write)
    {
      /* try to get an entry from the free list, if available */
      if (!bucket_free_list)
            {
              /* otherwise, call calloc() to get the needed storage */
              bucket_free_list = calloc(1, sizeof(struct spec_mem_ent));
              if (!bucket_free_list)
                fatal("out of virtual memory");
            }
      ent = bucket_free_list;
      bucket_free_list = bucket_free_list->next;

      if (!bugcompat_mode)
            {
              /* insert into hash table */
              ent->next = store_htable[index];
              store_htable[index] = ent;
              ent->addr = addr;
              ent->data[0] = 0; ent->data[1] = 0;
            }
    }

  /* handle the read or write to speculative or non-speculative
storage */
  switch (nbytes)
    {
    case 1:
      if (cmd == Read)
            {
              if (ent)
                {
                  /* read from mis-speculated state buffer */
                  *((unsigned char *)p) = *((unsigned char *)(&ent-
>data[0]));
                }
```

```
            else
              {
                /* read from non-speculative memory state */
                *((unsigned char *)p) = MEM_READ_BYTE(addr);
              }
          }
        else
          {
            /* always write into mis-speculated state buffer */
            *((unsigned char *)(&ent->data[0])) = *((unsigned char
*)p);
          }
        break;
      case 2:
        if (cmd == Read)
          {
            if (ent)
              {
                /* read from mis-speculated state buffer */
                *((unsigned short *)p) = *((unsigned short *)(&ent-
>data[0]));
              }
            else
              {
                /* read from non-speculative memory state */
                *((unsigned short *)p) = MEM_READ_HALF(addr);
              }
          }
        else
          {
            /* always write into mis-speculated state buffer */
            *((unsigned short *)&ent->data[0]) = *((unsigned short
*)p);
          }
        break;
      case 4:
        if (cmd == Read)
          {
            if (ent)
              {
                /* read from mis-speculated state buffer */
                *((unsigned int *)p) = *((unsigned int *)&ent-
>data[0]);
              }
            else
              {
```

```
            /* read from non-speculative memory state */
            *((unsigned int *)p) = MEM_READ_WORD(addr);
          }
        }
    else
        {
         /* always write into mis-speculated state buffer */
         *((unsigned int *)&ent->data[0]) = *((unsigned int *)p);
        }
    break;
  case 8:
    if (cmd == Read)
        {
         if (ent)
           {
            /* read from mis-speculated state buffer */
            *((unsigned int *)p) = *((unsigned int *)&ent-
>data[0]);
            *(((unsigned int *)p)+1) = *((unsigned int *)&ent-
>data[1]);
           }
         else
           {
            /* read from non-speculative memory state */
            *((unsigned int *)p) = MEM_READ_WORD(addr);
            *(((unsigned int *)p)+1) =
                MEM_READ_WORD(addr +
sizeof(SS_WORD_TYPE));
           }
        }
    else
        {
         /* always write into mis-speculated state buffer */
         *((unsigned int *)&ent->data[0]) = *((unsigned int *)p);
        }
    break;
  default:
    panic("access size not supported in mis-speculative mode");
  }
}

/* dump speculative memory state */
static void
mspec_dump(FILE *stream)                    /* output stream */
{
 int i;
```

```
  struct spec_mem_ent *ent;

  fprintf(stream, "** speculative memory contents **\n");

  fprintf(stream, "spec_mode: %s\n", spec_mode ? "t" : "f");

  for (i=0; i<STORE_HASH_SIZE; i++)
    {
      /* dump contents of all hash table buckets */
      for (ent=store_htable[i]; ent; ent=ent->next)
          {
            fprintf(stream, "[0x%08x]: %12.0f/0x%08x:%08x\n",
                    ent->addr, (double)(*((double *)ent->data)),
                    *((unsigned int *)&ent->data[0]),
                    *(((unsigned int *)&ent->data[0]) + 1));
          }
    }
}

/* default memory state accessor, used by DLite */
static char *                                   /* err str, NULL for
no err */
simoo_mem_obj(enum dlite_access_t at,                /* access type
*/
              SS_ADDR_TYPE addr,            /* address to access */
              char *p,                      /* input/output buffer
*/
              int nbytes)                   /* size of access */
{
  char *errstr;
  enum mem_cmd cmd;

  if (at == at_read)
    cmd = Read;
  else if (at == at_write)
    cmd = Write;
  else
    panic("bogus access type");

  errstr = mem_valid(cmd, addr, nbytes, /* declare */FALSE);
  if (errstr)
    return errstr;

  /* else, no error, access memory */
  if (spec_mode)
    spec_mem_access(cmd, addr, p, nbytes);
```

```
  else
    mem_access(cmd, addr, p, nbytes);

  /* no error */
  return NULL;
}


/*
 *  RUU_DISPATCH() - decode instructions and allocate RUU
and LSQ resources
 */

/* link RS onto the output chain number of whichever operation
will next
   create the architected register value IDEP_NAME */
static INLINE void
ruu_link_idep(struct RUU_station *rs,                    /* rs station to
link */
              int idep_num,                              /* input dependence
number */
              int idep_name)                             /* input register name
*/
{
  struct CV_link head;
  struct RS_link *link;

  /* any dependence? */
  if (idep_name == NA)
    {
      /* no input dependence for this input slot, mark operand as
ready */
      rs->idep_ready[idep_num] = TRUE;
      return;
    }

  /* locate creator of operand */
  head = CREATE_VECTOR(idep_name);

  /* any creator? */
  if (!head.rs)
    {
      /* no active creator, use value available in architected reg file,
         indicate the operand is ready for use */
      rs->idep_ready[idep_num] = TRUE;
      return;
```

```
    }
  /* else, creator operation will make this value sometime in the
future */

  /* indicate value will be created sometime in the future, i.e.,
operand
     is not yet ready for use */
  rs->idep_ready[idep_num] = FALSE;

  /* link onto creator's output list of dependant operand */
  RSLINK_NEW(link, rs); link->x.opnum = idep_num;
  link->next = head.rs->odep_list[head.odep_num];
  head.rs->odep_list[head.odep_num] = link;
}

/* make RS the creator of architected register ODEP_NAME */
static INLINE void
ruu_install_odep(struct RUU_station *rs,      /* creating RUU
station */
                 int odep_num,                /* output
operand number */
                 int odep_name)               /* output
register name */
{
  struct CV_link cv;

  /* any dependence? */
  if (odep_name == NA)
    {
      /* no value created */
      rs->onames[odep_num] = NA;
      return;
    }
  /* else, create a RS_NULL terminated output chain in create
vector */

  /* record output name, used to update create vector at completion
*/
  rs->onames[odep_num] = odep_name;

  /* initialize output chain to empty list */
  rs->odep_list[odep_num] = NULL;

  /* indicate this operation is latest creator of ODEP_NAME */
  CVLINK_INIT(cv, rs, odep_num);
  SET_CREATE_VECTOR(odep_name, cv);
```

```
}


/*
 * configure the instruction decode engine
 */

/* general register dependence decoders */
#define DGPR(N)                         (N)
#define DCGPR(N)            (SS_COMP_OP != SS_COMP_NOP
? (N) : 0)
#define DGPR_D(N)           ((N)&~1)

/* floating point register dependence decoders */
#define DFPR_L(N)           (((N)+32)&~1)
#define DFPR_F(N)           (((N)+32)&~1)
#define DFPR_D(N)           (((N)+32)&~1)

/* miscellaneous register dependence decoders */
#define DHI                 (0+32+32)
#define DLO                 (1+32+32)
#define DFCC                (2+32+32)
#define DTMP                        (3+32+32)
#define DNA                 (0)


/*
 * configure the execution engine
 */

/* next program counter */
#define SET_NPC(EXPR)         (next_PC = (EXPR))

/* target program counter */
#undef  SET_TPC
#define SET_TPC(EXPR)               (target_PC = (EXPR))

/* current program counter */
#define CPC                 (regs_PC)
#define SET_CPC(EXPR)         (regs_PC = (EXPR))

/* general purpose register accessors, NOTE: speculative copy on
write storage
   provided for fast recovery during wrong path execute (see
tracer_recover()
   for details on this process */
```

```
#define GPR(N)                (BITMAP_SET_P(use_spec_R,
R_BMAP_SZ, (N))\
                                  ? spec_regs_R[N]                \
                                  : regs_R[N])
#define SET_GPR(N,EXPR)       (spec_mode                          \
                                  ? (BITMAP_SET(use_spec_R,
R_BMAP_SZ, (N)),\
                                     spec_regs_R[N] = (EXPR))
\
                                  : (regs_R[N] = (EXPR)))

/* floating point register accessors, NOTE: speculative copy on
write storage
   provided for fast recovery during wrong path execute (see
tracer_recover()
   for details on this process */
#define FPR_L(N)              (BITMAP_SET_P(use_spec_F,
F_BMAP_SZ, ((N)&~1))\
                                  ? spec_regs_F.l[(N)]            \
                                  : regs_F.l[(N)])
#define SET_FPR_L(N,EXPR)     (spec_mode                          \
                                  ?
(BITMAP_SET(use_spec_F,F_BMAP_SZ,((N)&~1)),\
                                     spec_regs_F.l[(N)] = (EXPR))
\
                                  : (regs_F.l[(N)] = (EXPR)))
#define FPR_F(N)              (BITMAP_SET_P(use_spec_F,
F_BMAP_SZ, ((N)&~1))\
                                  ? spec_regs_F.f[(N)]            \
                                  : regs_F.f[(N)])
#define SET_FPR_F(N,EXPR)     (spec_mode                          \
                                  ?
(BITMAP_SET(use_spec_F,F_BMAP_SZ,((N)&~1)),\
                                     spec_regs_F.f[(N)] = (EXPR))
\
                                  : (regs_F.f[(N)] = (EXPR)))
#define FPR_D(N)              (BITMAP_SET_P(use_spec_F,
F_BMAP_SZ, ((N)&~1))\
                                  ? spec_regs_F.d[(N) >> 1]       \
                                  : regs_F.d[(N) >> 1])
#define SET_FPR_D(N,EXPR)     (spec_mode
\
                                  ?
(BITMAP_SET(use_spec_F,F_BMAP_SZ,((N)&~1)),\
                                     spec_regs_F.d[(N) >> 1] =
(EXPR))  \
```

```
                                        : (regs_F.d[(N) >> 1] = (EXPR)))

/* miscellanous register accessors, NOTE: speculative copy on
write storage
   provided for fast recovery during wrong path execute (see
tracer_recover()
   for details on this process */
#define HI                  (use_spec_HI ? spec_regs_HI : regs_HI)
#define SET_HI(EXPR)          (spec_mode                          \
                                       ? (use_spec_HI=TRUE,                \
                                          spec_regs_HI = (EXPR))           \
                                       : (regs_HI = (EXPR)))
#define LO                  (use_spec_LO ? spec_regs_LO :
regs_LO)
#define SET_LO(EXPR)          (spec_mode                          \
                                       ? (use_spec_LO=TRUE,
\
                                          spec_regs_LO = (EXPR))
\
                                       : (regs_LO = (EXPR)))
#define FCC                 (use_spec_FCC ? spec_regs_FCC :
regs_FCC)
#define SET_FCC(EXPR)         (spec_mode                          \
                                       ? (use_spec_FCC=TRUE,
\
                                          spec_regs_FCC = (EXPR))
\
                                       : (regs_FCC = (EXPR)))

/* precise architected memory state accessor macros, NOTE:
speculative copy on
   write storage provided for fast recovery during wrong path
execute (see
   tracer_recover() for details on this process */
#define __READ_SPECMEM(DST_T, SRC_N, SRC)
                \
  (addr = (SRC),
         \
   (spec_mode
         \
    ? spec_mem_access(Read, addr, SYMCAT(&temp_,SRC_N),
                \
                      sizeof(SYMCAT(temp_,SRC_N)))
         \
    : mem_access(Read, addr, SYMCAT(&temp_,SRC_N),
         \
```

```
                sizeof(SYMCAT(temp_,SRC_N))),
        \
  ((unsigned int)((DST_T)SYMCAT(temp_,SRC_N))))

#define READ_WORD(SRC)
                \
  __READ_SPECMEM(unsigned int, uint, (SRC))
#define READ_UNSIGNED_HALF(SRC)
                \
  __READ_SPECMEM(unsigned int, ushort, (SRC))
#define READ_SIGNED_HALF(SRC)
                \
  __READ_SPECMEM(signed int, short, (SRC))
#define READ_UNSIGNED_BYTE(SRC)
                \
  __READ_SPECMEM(unsigned int, uchar, (SRC))
#define READ_SIGNED_BYTE(SRC)
                \
  __READ_SPECMEM(signed int, char, (SRC))

#define __WRITE_SPECMEM(SRC, DST_T, DST_N, DST)
                        \
  (addr = (DST),
        \
   SYMCAT(temp_,DST_N) = (DST_T)((unsigned int)(SRC)),
                \
   (spec_mode
        \
    ? spec_mem_access(Write, addr, SYMCAT(&temp_,DST_N),
        \
                        sizeof(SYMCAT(temp_,DST_N)))
        \
    : mem_access(Write, addr, SYMCAT(&temp_,DST_N),
        \
                sizeof(SYMCAT(temp_,DST_N)))))

#define WRITE_WORD(SRC, DST)
        \
  __WRITE_SPECMEM((SRC), unsigned int, uint, (DST))
#define WRITE_HALF(SRC, DST)
        \
  __WRITE_SPECMEM((SRC), unsigned short, ushort, (DST))
#define WRITE_BYTE(SRC, DST)
        \
  __WRITE_SPECMEM((SRC), unsigned char, uchar, (DST))
```

```
/* system call handler macro */
#define SYSCALL(INST)
        \
  (/* only execute system calls in non-speculative mode */
        \
   (spec_mode ? panic("speculative syscall") : (void) 0),
        \
   ss_syscall(mem_access, INST))

/* non-expression inst implementations */
#define
DEFINST(OP,MSK,NAME,OPFORM,RES,CLASS,O1,O2,I1,I2,I
3,EXPR)
#define DEFLINK(OP,MSK,NAME,MASK,SHIFT)
#define CONNECT(OP)
#define IMPL
#include "ss.def"
#undef DEFINST
#undef DEFLINK
#undef CONNECT
#undef IMPL


/* default register state accessor, used by DLite */
static char *                           /* err str, NULL for
no err */
simoo_reg_obj(enum dlite_access_t at,           /* access type
*/
              enum dlite_reg_t rt,      /* reg bank to probe
*/
              int reg,                  /* register number */
              union dlite_reg_val_t *val)       /* input, output */
{
  if (reg < 0 || reg >= SS_NUM_REGS)
    return "register number out of range";

  if (at == at_read || at == at_write)
    {
      switch (rt)
        {
        case rt_gpr:
          if (at == at_read)
            val->as_word = GPR(reg);
          else
            SET_GPR(reg, val->as_word);
          break;
```

```
case rt_lpr:
  if (at == at_read)
    val->as_word = FPR_L(reg);
  else
    SET_FPR_L(reg, val->as_word);
  break;
case rt_fpr:
  if (at == at_read)
    val->as_float = FPR_F(reg);
  else
    SET_FPR_F(reg, val->as_float);
  break;
case rt_dpr:
  /* 1/2 as many regs in this mode */
  if (reg < 0 || reg >= SS_NUM_REGS/2)
    return "register number out of range";

  if (at == at_read)
    val->as_double = FPR_D(reg * 2);
  else
    SET_FPR_D(reg * 2, val->as_double);
  break;
case rt_hi:
  if (at == at_read)
    val->as_word = HI;
  else
    SET_HI(val->as_word);
  break;
case rt_lo:
  if (at == at_read)
    val->as_word = LO;
  else
    SET_LO(val->as_word);
  break;
case rt_FCC:
  if (at == at_read)
    val->as_condition = FCC;
  else
    SET_FCC(val->as_condition);
  break;
case rt_PC:
  if (at == at_read)
    val->as_address = regs_PC;
  else
    regs_PC = val->as_address;
  break;
```

```
          default:
            panic("bogus register bank");
          }
    }
  else
    panic("bogus access type");

  /* no error */
  return NULL;
}

/* non-zero for a valid address */
#define VALID_ADDR(ADDR) (SS_DATA_BASE <= (ADDR)
&& (ADDR) <= SS_STACK_BASE)

/* the last operation that ruu_dispatch() attempted to dispatch, for
   implementing in-order issue */
static struct RS_link last_op = RSLINK_NULL_DATA;

/* dispatch instructions from the IFETCH -> DISPATCH queue:
instructions are
   first decoded, then they allocated RUU (and LSQ for load/stores)
resources
   and input and output dependence chains are updated accordingly
*/
static void
ruu_dispatch(void)
{
  int i;
  int n_dispatched;                 /* total insts dispatched */
  SS_INST_TYPE inst;                  /* actual instruction bits */
  enum ss_opcode op;                /* decoded opcode enum */
  int out1, out2, in1, in2, in3;       /* output/input register names */
  SS_ADDR_TYPE next_PC, target_PC;      /* actual next/target
PC address */
  SS_ADDR_TYPE addr;                  /* effective address, if
load/store */
  struct RUU_station *rs;             /* RUU station being allocated
*/
  struct RUU_station *lsq;            /* LSQ station for ld/st's */
  struct bpred_update *dir_update_ptr;          /* branch predictor dir
update ptr */
  int stack_recover_idx;                  /* bpred retstack recovery
index */
  unsigned int pseq;                      /* pipetrace sequence number
*/
```

```
  int is_write;                 /* store? */
  int made_check;               /* used to ensure DLite entry
*/
  int br_taken, br_pred_taken;  /* if br, taken?  predicted
taken? */
  int fetch_redirected = FALSE;
  unsigned int temp_uint;       /* temp variable for spec
mem access */
  signed short temp_short;      /* " ditto " */
  unsigned short temp_ushort;   /* " ditto " */
  signed char temp_char;        /* " ditto " */
  unsigned char temp_uchar;     /* " ditto " */

  made_check = FALSE;
  n_dispatched = 0;
  while (/* instruction decode B/W left? */
         n_dispatched < (ruu_decode_width * fetch_speed)
         /* RUU and LSQ not full? */
         && RUU_num < RUU_size && LSQ_num < LSQ_size
         /* insts still available from fetch unit? */
         && fetch_num != 0
         /* on an acceptable trace path */
         && (ruu_include_spec || !spec_mode))
    {
      /* if issuing in-order, block until last op issues if inorder issue
*/
      if (ruu_inorder_issue
          && (last_op.rs && RSLINK_VALID(&last_op)
              && !OPERANDS_READY(last_op.rs)))
        {
          /* stall until last operation is ready to issue */
          break;
        }

      /* get the next instruction from the IFETCH -> DISPATCH
queue */
      inst = fetch_data[fetch_head].IR;
      regs_PC = fetch_data[fetch_head].regs_PC;
      pred_PC = fetch_data[fetch_head].pred_PC;
      dir_update_ptr = &(fetch_data[fetch_head].dir_update);
      stack_recover_idx = fetch_data[fetch_head].stack_recover_idx;
      pseq = fetch_data[fetch_head].ptrace_seq;

      /* decode the inst */
      op = SS_OPCODE(inst);
```

```
      /* compute default next_PC */
      next_PC = regs_PC + sizeof(SS_INST_TYPE);

      /* drain RUU for TRAPs and system calls */
      if (SS_OP_FLAGS(op) & F_TRAP)
        {
          if (RUU_num != 0)
            break;

          /* else, syscall is only instruction in the machine, at this
             point we should not be in (mis-)speculative mode */
          if (spec_mode)
            panic("drained and speculative");
        }

      /* maintain $r0 semantics (in spec and non-spec space) */
      regs_R[0] = 0; spec_regs_R[0] = 0;

      /* default effective address (none) and access */
      addr = 0; is_write = FALSE;

      /* more decoding and execution */
      switch (op)
        {

/* the divide-by-zero instruction check macro is redefined because
there
   is no portable and reliable way to hide integer divide-by-zero
faults */
#undef  DIV0
#define DIV0(N)                 (spec_mode
          \
                                ? /* do nothing */(void)0
          \
                                : (((N) == 0) ? IFAIL("divide by 0") :
(void)0))

/* the divide operator semantics are also redefined because there
   are no portable and reliable way to hide integer divide-by-zero
faults */
#undef  IDIV
#define IDIV(A, B)     (((B) == 0) ? 0 : ((A) / (B)))

#undef  IMOD
#define IMOD(A, B)  (((B) == 0) ? 0 : ((A) % (B)))
```

```
#undef  FDIV
#define FDIV(A, B)    (((B) == 0) ? 0 : ((A) / (B)))

#undef  FINT
#define FINT(A)                 (isnan(A) ? 0 : ((int)(A)))

/* decode and execute the instruction */
/* the following macro wraps the instruction check failure
declaration
   with a test to see if the trace generator is in non-speculative
   mode, if so the instruction fault is declared, otherwise, the error
   is shunted because instruction faults need to be masked on the
   mis-speculated instruction paths */
#undef  IFAIL
#define IFAIL(S)
        \
  (spec_mode ? /* ignore */(void)0 : /* declare */(void)fatal(S))

#define
DEFINST(OP,MSK,NAME,OPFORM,RES,CLASS,O1,O2,I1,I2,I
3,EXPR)        \
        case OP:
        \
          out1 = O1; out2 = O2;
        \
          in1 = I1; in2 = I2; in3 = I3;
        \
          EXPR;
        \
          break;
#define DEFLINK(OP,MSK,NAME,MASK,SHIFT)
                        \
        case OP:
        \
          /* could speculatively decode a bogus inst */
        \
          op = NOP;
        \
          out1 = NA; out2 = NA;
        \
          in1 = NA; in2 = NA; in3 = NA;
        \
          /* no EXPR */
        \
          break;
#define CONNECT(OP)
```

```
#include "ss.def"
#undef DEFINST
#undef DEFLINK
#undef CONNECT
          default:
            /* can speculatively decode a bogus inst */
            op = NOP;
            out1 = NA; out2 = NA;
            in1 = NA; in2 = NA; in3 = NA;
            /* no EXPR */
          }
      /* operation sets next_PC */

      /* update memory access stats */
      if (SS_OP_FLAGS(op) & F_MEM)
          {
            sim_total_refs++;
            if (!spec_mode)
              sim_num_refs++;

            if (SS_OP_FLAGS(op) & F_STORE)
              is_write = TRUE;
            else
              {
                sim_total_loads++;
                if (!spec_mode)
                     sim_num_loads++;
              }
          }

      br_taken = (next_PC != (regs_PC + sizeof(SS_INST_TYPE)));
      br_pred_taken = (pred_PC != (regs_PC +
sizeof(SS_INST_TYPE)));

      if ((pred_PC != next_PC && pred_perfect)
            || ((SS_OP_FLAGS(op) & (F_CTRL|F_DIRJMP)) ==
(F_CTRL|F_DIRJMP)
                && target_PC != pred_PC && br_pred_taken))
          {
            /* Either 1) we're simulating perfect prediction and are in
a
            mis-predict state and need to patch up, or 2) We're not
simulating
            perfect prediction, we've predicted the branch taken, but
our
            predicted target doesn't match the computed target (i.e.,
```

```
          mis-fetch).  Just update the PC values and do a fetch
squash.
          This is just like calling fetch_squash() except we pre-
anticipate
          the updates to the fetch values at the end of this function.
If
          case #2, also charge a mispredict penalty for redirecting
fetch */
           fetch_pred_PC = fetch_regs_PC = next_PC;
           /* was: if (pred_perfect) */
#if 1
           if (pred_perfect)
#endif
             pred_PC = next_PC;

           fetch_head = (ruu_ifq_size-1);
           fetch_num = 1;
           fetch_tail = 0;

           if (!pred_perfect)
             ruu_fetch_issue_delay = ruu_branch_penalty;
#if 0
           else
             {
               /* only 1 cycle bubble */
               ruu_fetch_issue_delay = 1;
             }
#endif

           fetch_redirected = TRUE;
         }

     /* is this a NOP */
     if (op != NOP)
         {
           /* for load/stores:
                idep #0     - store operand (value that is store'ed)
                idep #1, #2 - eff addr computation inputs (addr of
access)

              resulting RUU/LSQ operation pair:
                RUU (effective address computation operation):
                     idep #0, #1 - eff addr computation inputs (addr of
access)
                LSQ (memory access operation):
                     idep #0     - operand input (value that is store'd)
```

```
                    idep #1     - eff addr computation result (from RUU
op)

              effective address computation is transfered via the
reserved
              name DTMP
             */

            /* fill in RUU reservation station */
            rs = &RUU[RUU_tail];

            rs->IR = inst;
            rs->op = op;
            rs->PC = regs_PC;
            rs->next_PC = next_PC; rs->pred_PC = pred_PC;
            rs->in_LSQ = FALSE;
            rs->ea_comp = FALSE;
            rs->recover_inst = FALSE;
          rs->dir_update = *dir_update_ptr;
            rs->stack_recover_idx = stack_recover_idx;
            rs->spec_mode = spec_mode;
            rs->addr = 0;
            /* rs->tag is already set */
            rs->seq = ++inst_seq;
            rs->queued = rs->issued = rs->completed = FALSE;
            rs->ptrace_seq = pseq;

            /* split ld/st's into two operations: eff addr comp + mem
access */
            if (SS_OP_FLAGS(op) & F_MEM)
              {
                /* convert RUU operation from ld/st to an add (eff addr
comp) */
                rs->op = ADD;
                rs->ea_comp = TRUE;

                /* fill in LSQ reservation station */
                lsq = &LSQ[LSQ_tail];

                lsq->IR = inst;
                lsq->op = op;
                lsq->PC = regs_PC;
                lsq->next_PC = next_PC; lsq->pred_PC = pred_PC;
                lsq->in_LSQ = TRUE;
                lsq->ea_comp = FALSE;
                lsq->recover_inst = FALSE;
```

```
		lsq->dir_update.pdir1 = lsq->dir_update.pdir2 = NULL;
		lsq->dir_update.pmeta = NULL;
		lsq->stack_recover_idx = 0;
		lsq->spec_mode = spec_mode;
		lsq->addr = addr;
		/* lsq->tag is already set */
		lsq->seq = ++inst_seq;
		lsq->queued = lsq->issued = lsq->completed = FALSE;
		lsq->ptrace_seq = ptrace_seq++;

		/* pipetrace this uop */
		ptrace_newuop(lsq->ptrace_seq, "internal ld/st", lsq->PC, 0);
		ptrace_newstage(lsq->ptrace_seq, PST_DISPATCH, 0);

		/* link eff addr computation onto operand's output chains */
		ruu_link_idep(rs, /* idep_ready[] index */0, NA);
		ruu_link_idep(rs, /* idep_ready[] index */1, in2);
		ruu_link_idep(rs, /* idep_ready[] index */2, in3);

		/* install output after inputs to prevent self reference */
		ruu_install_odep(rs, /* odep_list[] index */0, DTMP);
		ruu_install_odep(rs, /* odep_list[] index */1, NA);

		/* link memory access onto output chain of eff addr operation */
		ruu_link_idep(lsq,
			      /* idep_ready[] index */STORE_OP_INDEX/* 0 */,
			      in1);
		ruu_link_idep(lsq,
			      /* idep_ready[] index */STORE_ADDR_INDEX/* 1 */,
			      DTMP);
		ruu_link_idep(lsq, /* idep_ready[] index */2, NA);

		/* install output after inputs to prevent self reference */
		ruu_install_odep(lsq, /* odep_list[] index */0, out1);
		ruu_install_odep(lsq, /* odep_list[] index */1, out2);

		/* install operation in the RUU and LSQ */
		n_dispatched++;
		RUU_tail = (RUU_tail + 1) % RUU_size;
		RUU_num++;
```

```
            LSQ_tail = (LSQ_tail + 1) % LSQ_size;
            LSQ_num++;

            if (OPERANDS_READY(rs))
                {
                  /* eff addr computation ready, queue it on ready
list */
                  readyq_enqueue(rs);
                }
            /* issue may continue when the load/store is issued */
            RSLINK_INIT(last_op, lsq);

            /* issue stores only, loads are issued by lsq_refresh() */
            if (((SS_OP_FLAGS(op) & (F_MEM|F_STORE)) ==
(F_MEM|F_STORE))
                  && OPERANDS_READY(lsq))
                {
                  /* panic("store immediately ready"); */
                  /* put operation on ready list, ruu_issue() issue it
later */
                  readyq_enqueue(lsq);
                }
          }
        else /* !(SS_OP_FLAGS(op) & F_MEM) */
          {
            /* link onto producing operation */
            ruu_link_idep(rs, /* idep_ready[] index */0, in1);
            ruu_link_idep(rs, /* idep_ready[] index */1, in2);
            ruu_link_idep(rs, /* idep_ready[] index */2, in3);

            /* install output after inputs to prevent self reference */
            ruu_install_odep(rs, /* odep_list[] index */0, out1);
            ruu_install_odep(rs, /* odep_list[] index */1, out2);

            /* install operation in the RUU */
            n_dispatched++;
            RUU_tail = (RUU_tail + 1) % RUU_size;
            RUU_num++;

            /* issue op if all its reg operands are ready (no mem
input) */
            if (OPERANDS_READY(rs))
                {
                  /* put operation on ready list, ruu_issue() issue it
later */
                  readyq_enqueue(rs);
```

```
                    /* issue may continue */
                    last_op = RSLINK_NULL;
                  }
                else
                  {
                    /* could not issue this inst, stall issue until we can
*/
                    RSLINK_INIT(last_op, rs);
                  }
              }
          }
        else
          {
            /* this is a NOP, no need to update RUU/LSQ state */
            rs = NULL;
          }

        /* one more instruction executed, speculative or otherwise */
        sim_total_insn++;
        if (SS_OP_FLAGS(op) & F_CTRL)
          sim_total_branches++;

        if (!spec_mode)
          {
            /* one more non-speculative instruction executed */
            sim_num_insn++;

            /* if this is a branching instruction update BTB, i.e., only
               non-speculative state is committed into the BTB */
            if (SS_OP_FLAGS(op) & F_CTRL)
              {
                sim_num_branches++;
                if (pred && bpred_spec_update == spec_ID)
                  {
                    bpred_update(pred,
                                 /* branch address */regs_PC,
                                 /* actual target address */next_PC,
                                 /* taken? */next_PC != (regs_PC +

sizeof(SS_INST_TYPE)),
                                 /* pred taken? */pred_PC != (regs_PC +

            sizeof(SS_INST_TYPE)),
                                 /* correct pred? */pred_PC == next_PC,
                                 /* opcode */op,
                                 /* jump through R31? */(RS) == 31,
```

```
                                /* predictor update ptr */&rs-
>dir_update,
                        /* instruction for FFBPANN */ inst);
                    }
                }

              /* is the trace generator trasitioning into mis-speculation
mode? */
              if (pred_PC != next_PC && !fetch_redirected)
                {
                  /* entering mis-speculation mode, indicate this and save
PC */
                  spec_mode = TRUE;
                  rs->recover_inst = TRUE;
                  recover_PC = next_PC;
                }
            }

          /* entered decode/allocate stage, indicate in pipe trace */
          ptrace_newstage(pseq, PST_DISPATCH,
                          (pred_PC != next_PC) ? PEV_MPOCCURED :
0);
          if (op == NOP)
            {
              /* end of the line */
              ptrace_endinst(pseq);
            }

          /* update any stats tracked by PC */
          for (i=0; i<pcstat_nelt; i++)
            {
              SS_COUNTER_TYPE newval;
              int delta;

              /* check if any tracked stats changed */
              newval = STATVAL(pcstat_stats[i]);
              delta = newval - pcstat_lastvals[i];
              if (delta != 0)
                {
                  stat_add_samples(pcstat_sdists[i], regs_PC, delta);
                  pcstat_lastvals[i] = newval;
                }
            }

          /* consume instruction from IFETCH -> DISPATCH queue */
          fetch_head = (fetch_head+1) & (ruu_ifq_size - 1);
```

```
      fetch_num--;

      /* check for DLite debugger entry condition */
      made_check = TRUE;
      if (dlite_check_break(pred_PC,
                            is_write ? ACCESS_WRITE :
ACCESS_READ,
                            addr, sim_num_insn, sim_cycle))
        dlite_main(regs_PC, pred_PC, sim_cycle);
    }

  /* need to enter DLite at least once per cycle */
  if (!made_check)
    {
      if (dlite_check_break(/* no next PC */0,
                            is_write ? ACCESS_WRITE :
ACCESS_READ,
                            addr, sim_num_insn, sim_cycle))
        dlite_main(regs_PC, /* no next PC */0, sim_cycle);
    }
}


/*
 *  RUU_FETCH() - instruction fetch pipeline stage(s)
 */

/* initialize the instruction fetch pipeline stage */
static void
fetch_init(void)
{
  /* allocate the IFETCH -> DISPATCH instruction queue */
  fetch_data =
    (struct fetch_rec *)calloc(ruu_ifq_size, sizeof(struct fetch_rec));
  if (!fetch_data)
    fatal("out of virtual memory");

  fetch_num = 0;
  fetch_tail = fetch_head = 0;
}

/* dump contents of fetch stage registers and fetch queue */
void
fetch_dump(FILE *stream)                        /* output stream */
{
  int num, head;
```

```
  fprintf(stream, "** fetch stage state **\n");

  fprintf(stream, "spec_mode: %s\n", spec_mode ? "t" : "f");
  fprintf(stream, "pred_PC: 0x%08x, recover_PC: 0x%08x\n",
          pred_PC, recover_PC);
  fprintf(stream, "fetch_regs_PC: 0x%08x, fetch_pred_PC:
0x%08x\n",
          fetch_regs_PC, fetch_pred_PC);
  fprintf(stream, "\n");

  fprintf(stream, "** fetch queue contents **\n");
  fprintf(stream, "fetch_num: %d\n", fetch_num);
  fprintf(stream, "fetch_head: %d, fetch_tail: %d\n",
          fetch_head, fetch_tail);

  num = fetch_num;
  head = fetch_head;
  while (num)
    {
      fprintf(stream, "idx: %2d: inst: `", head);
      ss_print_insn(fetch_data[head].IR, fetch_data[head].regs_PC,
stream);
      fprintf(stream, "'\n");
      fprintf(stream, "        regs_PC: 0x%08x, pred_PC: 0x%08x\n",
              fetch_data[head].regs_PC, fetch_data[head].pred_PC);
      head = (head + 1) & (ruu_ifq_size - 1);
      num--;
    }
}

static int last_inst_missed = FALSE;
static int last_inst_tmissed = FALSE;

/* fetch up as many instruction as one branch prediction and one
cache line
   acess will support without overflowing the IFETCH ->
DISPATCH QUEUE */
static void
ruu_fetch(void)
{
  int i, lat, tlb_lat, done = FALSE;
  SS_INST_TYPE inst;
  int stack_recover_idx;
  int branch_cnt;
```

```
  for (i=0, branch_cnt=0;
       /* fetch up to as many instruction as the DISPATCH stage can
decode */
       i < (ruu_decode_width * fetch_speed)
       /* fetch until IFETCH -> DISPATCH queue fills */
       && fetch_num < ruu_ifq_size
       /* and no IFETCH blocking condition encountered */
       && !done;
       i++)
     {
       /* fetch an instruction at the next predicted fetch address */
       fetch_regs_PC = fetch_pred_PC;

       /* is this a bogus text address? (can happen on mis-spec path)
*/
       if (ld_text_base <= fetch_regs_PC
           && fetch_regs_PC < (ld_text_base+ld_text_size)
           && !(fetch_regs_PC & (sizeof(SS_INST_TYPE)-1)))
         {
           /* read instruction from memory */
           mem_access(Read, fetch_regs_PC, &inst,
sizeof(SS_INST_TYPE));

           /* address is within program text, read instruction from
memory */
           lat = cache_il1_lat;
           if (cache_il1)
             {
               /* access the I-cache */
               lat =
                   cache_access(cache_il1, Read, fetch_regs_PC,
                                NULL, sizeof(SS_INST_TYPE),
sim_cycle,
                                NULL, NULL);
               if (lat > cache_il1_lat)
                   last_inst_missed = TRUE;
             }

           if (itlb)
             {
               /* access the I-TLB, NOTE: this code will initiate
                    speculative TLB misses */
               tlb_lat =
                   cache_access(itlb, Read, fetch_regs_PC,
                                NULL, sizeof(SS_INST_TYPE),
sim_cycle,
```

```
                                NULL, NULL);
              if (tlb_lat > 1)
                  last_inst_tmissed = TRUE;

              /* I-cache/I-TLB accesses occur in parallel */
              lat = MAX(tlb_lat, lat);
            }

          /* I-cache/I-TLB miss? assumes I-cache hit >= I-TLB hit
*/
          if (lat != cache_il1_lat)
            {
              /* I-cache miss, block fetch until it is resolved */
              ruu_fetch_issue_delay += lat - 1;
              break;
            }
          /* else, I-cache/I-TLB hit */
        }
      else
        {
          /* fetch PC is bogus, send a NOP down the pipeline */
          inst = SS_NOP_INST;
        }

      /* have a valid inst, here */

      /* possibly use the BTB target */
      if (pred)
        {
          /* get the next predicted fetch address; only use branch
predictor
             result for branches (assumes pre-decode bits); NOTE:
returned
             value may be 1 if bpred can only predict a direction */
          if (SS_OP_FLAGS(SS_OPCODE(inst)) & F_CTRL)
            fetch_pred_PC =
              bpred_lookup(pred,
                           /* branch address */fetch_regs_PC,
                           /* target address *//* FIXME: not
computed */0,
                           /* opcode */SS_OPCODE(inst),
                           /* jump through r31? */(RS) == 31,
                           /* updt
*/&(fetch_data[fetch_tail].dir_update),
                           /* RSB index */&stack_recover_idx,
                   /* instruction for FFBPANN */ inst);
```

```
            else
              fetch_pred_PC = 0;

            /* valid address returned from branch predictor? */
            if (!fetch_pred_PC)
              {
                /* no predicted taken target, attempt not taken target */
                fetch_pred_PC = fetch_regs_PC +
sizeof(SS_INST_TYPE);
              }
            else
              {
                /* go with target, NOTE: discontinuous fetch, so
terminate */
                branch_cnt++;
                if (branch_cnt >= fetch_speed)
                     done = TRUE;
              }
          }
        else
          {
            /* no predictor, just default to predict not taken, and
               continue fetching instructions linearly */
            fetch_pred_PC = fetch_regs_PC +
sizeof(SS_INST_TYPE);
          }

        /* commit this instruction to the IFETCH -> DISPATCH queue
*/
        fetch_data[fetch_tail].IR = inst;
        fetch_data[fetch_tail].regs_PC = fetch_regs_PC;
        fetch_data[fetch_tail].pred_PC = fetch_pred_PC;
        fetch_data[fetch_tail].stack_recover_idx = stack_recover_idx;
        fetch_data[fetch_tail].ptrace_seq = ptrace_seq++;

        /* for pipe trace */
        ptrace_newinst(fetch_data[fetch_tail].ptrace_seq,
                       inst, fetch_data[fetch_tail].regs_PC,
                       0);
        ptrace_newstage(fetch_data[fetch_tail].ptrace_seq,
                        PST_IFETCH,
                        ((last_inst_missed ? PEV_CACHEMISS : 0)
                         | (last_inst_tmissed ? PEV_TLBMISS : 0)));
        last_inst_missed = FALSE;
        last_inst_tmissed = FALSE;
```

```
      /* adjust instruction fetch queue */
      fetch_tail = (fetch_tail + 1) & (ruu_ifq_size - 1);
      fetch_num++;
    }
}

/* default machine state accessor, used by DLite */
static char *                                   /* err str, NULL for
no err */
simoo_mstate_obj(FILE *stream,                          /* output
stream */
                 char *cmd)                     /* optional command
string */
{
  if (!cmd || !strcmp(cmd, "help"))
    fprintf(stream,
"mstate commands:\n"
"\n"
"    mstate help   - show all machine-specific commands (this
list)\n"
"    mstate stats  - dump all statistical variables\n"
"    mstate res    - dump current functional unit resource states\n"
"    mstate ruu    - dump contents of the register update unit\n"
"    mstate lsq    - dump contents of the load/store queue\n"
"    mstate eventq - dump contents of event queue\n"
"    mstate readyq - dump contents of ready instruction queue\n"
"    mstate cv     - dump contents of the register create vector\n"
"    mstate rspec  - dump contents of speculative regs\n"
"    mstate mspec  - dump contents of speculative memory\n"
"    mstate fetch  - dump contents of fetch stage registers and fetch
queue\n"
"\n"
            );
  else if (!strcmp(cmd, "stats"))
    {
      /* just dump intermediate stats */
      sim_print_stats(stream);
    }
  else if (!strcmp(cmd, "res"))
    {
      /* dump resource state */
      res_dump(fu_pool, stream);
    }
  else if (!strcmp(cmd, "ruu"))
    {
      /* dump RUU contents */
```

```
      ruu_dump(stream);
    }
  else if (!strcmp(cmd, "lsq"))
    {
      /* dump LSQ contents */
      lsq_dump(stream);
    }
  else if (!strcmp(cmd, "eventq"))
    {
      /* dump event queue contents */
      eventq_dump(stream);
    }
  else if (!strcmp(cmd, "readyq"))
    {
      /* dump event queue contents */
      readyq_dump(stream);
    }
  else if (!strcmp(cmd, "cv"))
    {
      /* dump event queue contents */
      cv_dump(stream);
    }
  else if (!strcmp(cmd, "rspec"))
    {
      /* dump event queue contents */
      rspec_dump(stream);
    }
  else if (!strcmp(cmd, "mspec"))
    {
      /* dump event queue contents */
      mspec_dump(stream);
    }
  else if (!strcmp(cmd, "fetch"))
    {
      /* dump event queue contents */
      fetch_dump(stream);
    }
  else
    return "unknown mstate command";

  /* no error */
  return NULL;
}


/* start simulation, program loaded, processor precise state
```

```
initialized */
void
sim_main(void)
{
  fprintf(stderr, "sim: ** starting performance simulation **\n");

  /* ignore any floating point exceptions, they may occur on mis-
speculated
     execution paths */
  signal(SIGFPE, SIG_IGN);

  /* set up program entry state */
  SET_CPC(ld_prog_entry-sizeof(SS_INST_TYPE));
  fetch_regs_PC = regs_PC;
  fetch_pred_PC = ld_prog_entry;

  /* check for DLite debugger entry condition */
  if (dlite_check_break(regs_PC, /* no access */0, /* addr */0, 0,
0))
    dlite_main(regs_PC, regs_PC + SS_INST_SIZE, sim_cycle);

  /* main simulator loop, NOTE: the pipe stages are traverse in
reverse order
     to eliminate this/next state synchronization and relaxation
problems */
  for (;;)
    {
      /* RUU/LSQ sanity checks */
      if (RUU_num < LSQ_num)
          panic("RUU_num < LSQ_num");
      if (((RUU_head + RUU_num) % RUU_size) != RUU_tail)
          panic("RUU_head/RUU_tail wedged");
      if (((LSQ_head + LSQ_num) % LSQ_size) != LSQ_tail)
          panic("LSQ_head/LSQ_tail wedged");

      /* check if pipetracing is still active */
      ptrace_check_active(regs_PC, sim_num_insn, sim_cycle);

      /* indicate new cycle in pipetrace */
      ptrace_newcycle(sim_cycle);

      /* commit entries from RUU/LSQ to architected register file */
      ruu_commit();

      /* service function unit release events */
      ruu_release_fu();
```

```
    /* ==> may have ready queue entries carried over from
previous cycles */

    /* service result completions, also readies dependent operations
*/
    /* ==> inserts operations into ready queue --> register deps
resolved */
    ruu_writeback();

    if (!bugcompat_mode)
        {
          /* try to locate memory operations that are ready to
execute */
          /* ==> inserts operations into ready queue --> mem deps
resolved */
          lsq_refresh();

          /* issue operations ready to execute from a previous cycle
*/
          /* <== drains ready queue <-- ready operations commence
execution */
          ruu_issue();
        }

    /* decode and dispatch new operations */
    /* ==> insert ops w/ no deps or all regs ready --> reg deps
resolved */
    ruu_dispatch();

    if (bugcompat_mode)
        {
          /* try to locate memory operations that are ready to
execute */
          /* ==> inserts operations into ready queue --> mem deps
resolved */
          lsq_refresh();

          /* issue operations ready to execute from a previous cycle
*/
          /* <== drains ready queue <-- ready operations commence
execution */
          ruu_issue();
        }

    /* call instruction fetch unit if it is not blocked */
```

```
      if (!ruu_fetch_issue_delay)
          ruu_fetch();
      else
          ruu_fetch_issue_delay--;

      /* go to next cycle */
      sim_cycle++;
    }
  }
```

sim-bpred.c

```
  /*
   * sim-bpred.c - sample branch predictor simulator implementation
   *
   * This file is a part of the SimpleScalar tool suite written by
   * Todd M. Austin as a part of the Multiscalar Research Project.
   *
   * The tool suite is currently maintained by Doug Burger and Todd
  M. Austin.
   *
   * Copyright (C) 1994, 1995, 1996, 1997 by Todd M. Austin
   *
   * This source file is distributed "as is" in the hope that it will be
   * useful.  The tool set comes with no warranty, and no author or
   * distributor accepts any responsibility for the consequences of its
   * use.
   *
   * Everyone is granted permission to copy, modify and redistribute
   * this tool set under the following conditions:
   *
   *   This source code is distributed for non-commercial use only.
   *   Please contact the maintainer for restrictions applying to
   *   commercial use.
   *
   *   Permission is granted to anyone to make or distribute copies
   *   of this source code, either as received or modified, in any
   *   medium, provided that all copyright notices, permission and
   *   nonwarranty notices are preserved, and that the distributor
   *   grants the recipient permission for further redistribution as
   *   permitted by this document.
   *
   *   Permission is granted to distribute this file in compiled
   *   or executable form under the same conditions that apply for
   *   source code, provided that either:
   *
```

```
 *    A. it is accompanied by the corresponding machine-readable
 *       source code,
 *    B. it is accompanied by a written offer, with no time limit,
 *       to give anyone a machine-readable copy of the
corresponding
 *       source code in return for reimbursement of the cost of
 *       distribution.  This written offer must permit verbatim
 *       duplication by anyone, or
 *    C. it is distributed by someone who received only the
 *       executable form, and is accompanied by a copy of the
 *       written offer of source code that they received concurrently.
 *
 * In other words, you are welcome to use, share and improve this
 * source file.  You are forbidden to forbid anyone else to use,
share
 * and improve what you give them.
 *
 * INTERNET: dburger@cs.wisc.edu
 * US Mail:  1210 W. Dayton Street, Madison, WI 53706
 *
 * $Id$
 *
 * $Log$
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "misc.h"
#include "ss.h"
#include "regs.h"
#include "memory.h"
#include "loader.h"
#include "syscall.h"
#include "dlite.h"
#include "options.h"
#include "stats.h"
#include "bpred.h"
#include "sim.h"

/*
 * This file implements a branch predictor analyzer.
 */
```

```
/* branch predictor type
{nottaken|taken|perfect|bimod|2lev|ffbpann} */
static char *pred_type;

/* bimodal predictor config (<table_size>) */
static int bimod_nelt = 1;
static int bimod_config[1] =
  { /* bimod tbl size */2048 };

/* ffbpann predictor config (<meta_table_size> <hl1nodes>
<hl2nodes> <hl3nodes>
<hl4nodes>) */
static int ffbpann_nelt = 5;
static int ffbpann_config[5] =
  { /* meta_table_size */1024, /* hidden layer nodes*/4, 3, 2, 1 };

/* 2-level predictor config (<l1size> <l2size> <hist_size> <xor>)
*/
static int twolev_nelt = 4;
static int twolev_config[4] =
  { /* l1size */1, /* l2size */1024, /* hist */8, /* xor */FALSE};

/* combining predictor config (<meta_table_size> */
static int comb_nelt = 1;
static int comb_config[1] =
  { /* meta_table_size */1024 };

/* return address stack (RAS) size */
static int ras_size = 8;

/* BTB predictor config (<num_sets> <associativity>) */
static int btb_nelt = 2;
static int btb_config[2] =
  { /* nsets */512, /* assoc */4 };

/* branch predictor */
static struct bpred *pred;

/* track number of insn and refs */
static SS_COUNTER_TYPE sim_num_insn = 0;
static SS_COUNTER_TYPE sim_num_refs = 0;

/* total number of branches executed */
static SS_COUNTER_TYPE sim_num_branches = 0;
```

```
/* register simulator-specific options */
void
sim_reg_options(struct opt_odb_t *odb)
{
  opt_reg_header(odb,
"sim-bpred: This simulator implements a branch predictor
analyzer.\n"
                 );

  /* branch predictor options */
  opt_reg_note(odb,
"  Branch predictor configuration examples for 2-level
predictor:\n"
"    Configurations:   N, M, W, X\n"
"      N   # entries in first level (# of shift register(s))\n"
"      W   width of shift register(s)\n"
"      M   # entries in 2nd level (# of counters, or other FSM)\n"
"      X   (yes-1/no-0) xor history and address for 2nd level index\n"
"    Sample predictors:\n"
"      GAg     : 1, W, 2^W, 0\n"
"      GAp     : 1, W, M (M > 2^W), 0\n"
"      PAg     : N, W, 2^W, 0\n"
"      PAp     : N, W, M (M == 2^(N+W)), 0\n"
"      gshare  : 1, W, 2^W, 1\n"
"  Predictor `comb' combines a bimodal and a 2-level predictor.\n"
"\n"
"  Predictor `ffbpann' combines a bimodal and a feed-forward
artificial\n"
"  neural network predictor.  The configuration for the FFBPANN
is:\n"
"   Meta, HL1Nodes, HL2Nodes, NL3Nodes, HL4Nodes\n"
"     Meta       size of meta table\n"
"     HL1Nodes   number of nodes in Hidden Layer 1\n"
"     HL2Nodes   number of nodes in Hidden Layer 2\n"
"     HL3Nodes   number of nodes in Hidden Layer 3\n"
"     HL4Nodes   number of nodes in Hidden Layer 4\n"
"   There are four inputs to the neural network (the input layer):\n"
"    The branch address, the op code, the rs value and the rt
value.\n"
"    There is one output (the output layer).\n"
"   Therefore the neural network can have from 2 (input and
output) to\n"
"   6 (up to 4 hidden) layers, of user specified sizes.\n"
             );

  opt_reg_string(odb, "-bpred",
```

```
                "branch predictor type
{nottaken|taken|bimod|2lev|comb|ffbpann}",
            &pred_type, /* default */"bimod",
            /* print */TRUE, /* format */NULL);

  opt_reg_int_list(odb, "-bpred:bimod",
                   "bimodal predictor config (<table size>)",
                   bimod_config, bimod_nelt, &bimod_nelt,
                   /* default */bimod_config,
                   /* print */TRUE, /* format */NULL, /* !accrue
*/FALSE);

  opt_reg_int_list(odb, "-bpred:ffbpann",
                "ffbpann config (<meta_table_size> <hl1nodes>
<hl2nodes> <hl3nodes> <hl4nodes>)",
                ffbpann_config, ffbpann_nelt, &ffbpann_nelt,
                /* default */ffbpann_config,
                /* print */TRUE, /* format */NULL, /* !accrue
*/FALSE);

  opt_reg_int_list(odb, "-bpred:2lev",
                "2-level predictor config "
                     "(<l1size> <l2size> <hist_size> <xor>)",
                twolev_config, twolev_nelt, &twolev_nelt,
                     /* default */twolev_config,
                /* print */TRUE, /* format */NULL, /* !accrue
*/FALSE);

  opt_reg_int_list(odb, "-bpred:comb",
                   "combining predictor config
(<meta_table_size>)",
                   comb_config, comb_nelt, &comb_nelt,
                   /* default */comb_config,
                   /* print */TRUE, /* format */NULL, /* !accrue
*/FALSE);

  opt_reg_int(odb, "-bpred:ras",
              "return address stack size (0 for no return stack)",
              &ras_size, /* default */ras_size,
              /* print */TRUE, /* format */NULL);

  opt_reg_int_list(odb, "-bpred:btb",
                   "BTB config (<num_sets> <associativity>)",
                   btb_config, btb_nelt, &btb_nelt,
                   /* default */btb_config,
                   /* print */TRUE, /* format */NULL, /* !accrue
```

```
*/FALSE);
}

/* check simulator-specific option values */
void
sim_check_options(struct opt_odb_t *odb, int argc, char **argv)
{
  if (!mystricmp(pred_type, "taken"))
    {
      /* static predictor, not taken */
      pred = bpred_create(BPredTaken, 0, 0, 0, 0, 0, 0, 0, 0, 0);
    }
  else if (!mystricmp(pred_type, "nottaken"))
    {
      /* static predictor, taken */
      pred = bpred_create(BPredNotTaken, 0, 0, 0, 0, 0, 0, 0, 0, 0);
    }
  else if (!mystricmp(pred_type, "bimod"))
    {
      if (bimod_nelt != 1)
          fatal("bad bimod predictor config (<table_size>)");
      if (btb_nelt != 2)
          fatal("bad btb config (<num_sets> <associativity>)");

      /* bimodal predictor, bpred_create() checks BTB_SIZE */
      pred = bpred_create(BPred2bit,
                          /* bimod table size */bimod_config[0],
                          /* 2lev l1 size */0,
                          /* 2lev l2 size */0,
                          /* meta table size */0,
                          /* history reg size */0,
                          /* history xor address */0,
                          /* btb sets */btb_config[0],
                          /* btb assoc */btb_config[1],
                          /* ret-addr stack size */ras_size);
    }
  else if (!mystricmp(pred_type, "ffbpann"))
    {
      /* ffbpann predictor - combination bimodal and ffbpann */
      if (bimod_nelt != 1)
        fatal("bad bimod predictor config (<table_size>)");
      if (btb_nelt != 2)
        fatal("bad btb config (<num_sets> <associativity>)");
      if (ffbpann_nelt != 5)
        fatal("bad ffbpann predictor config (<meta_table_size>
<hl1nodes> <hl2nodes> <hl3nodes> <hl4nodes>)");
```

```
    /* bimodal predictor, bpred_create() checks BTB_SIZE */
    pred = bpred_create(BPredFFBPANN,
                        /* bimod table size */bimod_config[0],
                        /* # of nodes in hidden layer1
*/ffbpann_config[1],
                        /* # of nodes in hidden layer2
*/ffbpann_config[2],
                        /* meta table size */ffbpann_config[0],
                        /* # of nodes in hidden layer3
*/ffbpann_config[3],
                        /* # of nodes in hidden layer4
*/ffbpann_config[4],
                        /* btb sets */btb_config[0],
                        /* btb assoc */btb_config[1],
                        /* ret-addr stack size */ras_size);
  }
 else if (!mystricmp(pred_type, "2lev"))
  {
    /* 2-level adaptive predictor, bpred_create() checks args */
    if (twolev_nelt != 4)
        fatal("bad 2-level pred config (<l1size> <l2size>
<hist_size> <xor>)");
    if (btb_nelt != 2)
        fatal("bad btb config (<num_sets> <associativity>)");

    pred = bpred_create(BPred2Level,
                        /* bimod table size */0,
                        /* 2lev l1 size */twolev_config[0],
                        /* 2lev l2 size */twolev_config[1],
                        /* meta table size */0,
                        /* history reg size */twolev_config[2],
                        /* history xor address */twolev_config[3],
                        /* btb sets */btb_config[0],
                        /* btb assoc */btb_config[1],
                        /* ret-addr stack size */ras_size);
  }
 else if (!mystricmp(pred_type, "comb"))
  {
    /* combining predictor, bpred_create() checks args */
    if (twolev_nelt != 4)
        fatal("bad 2-level pred config (<l1size> <l2size>
<hist_size> <xor>)");
    if (bimod_nelt != 1)
        fatal("bad bimod predictor config (<table_size>)");
    if (comb_nelt != 1)
```

```
        fatal("bad combining predictor config
(<meta_table_size>)");
    if (btb_nelt != 2)
        fatal("bad btb config (<num_sets> <associativity>)");

    pred = bpred_create(BPredComb,
                        /* bimod table size */bimod_config[0],
                        /* l1 size */twolev_config[0],
                        /* l2 size */twolev_config[1],
                        /* meta table size */comb_config[0],
                        /* history reg size */twolev_config[2],
                        /* history xor address */twolev_config[3],
                        /* btb sets */btb_config[0],
                        /* btb assoc */btb_config[1],
                        /* ret-addr stack size */ras_size);
  }
 else
  fatal("cannot parse predictor type `%s'", pred_type);
}

/* register simulator-specific statistics */
void
sim_reg_stats(struct stat_sdb_t *sdb)
{
 stat_reg_counter(sdb, "sim_num_insn",
                  "total number of instructions executed",
                  &sim_num_insn, 0, NULL);
 stat_reg_counter(sdb, "sim_num_refs",
                  "total number of loads and stores executed",
                  &sim_num_refs, 0, NULL);
 stat_reg_int(sdb, "sim_elapsed_time",
              "total simulation time in seconds",
              (int *)&sim_elapsed_time, 0, NULL);
 stat_reg_formula(sdb, "sim_inst_rate",
                  "simulation speed (in insts/sec)",
                  "sim_num_insn / sim_elapsed_time", NULL);

 stat_reg_counter(sdb, "sim_num_branches",
              "total number of branches executed",
              &sim_num_branches, /* initial value */0, /* format
*/NULL);
 stat_reg_formula(sdb, "sim_IPB",
              "instruction per branch",
              "sim_num_insn / sim_num_branches", /* format
*/NULL);
```

```
  /* register predictor stats */
  if (pred)
    bpred_reg_stats(pred, sdb);
}

/* initialize the simulator */
void
sim_init(void)
{
  SS_INST_TYPE inst;

  sim_num_insn = 0;
  sim_num_refs = 0;

  regs_PC = ld_prog_entry;

  /* decode all instructions */
  {
    SS_ADDR_TYPE addr;

    if (OP_MAX > 255)
      fatal("cannot perform fast decoding, too many opcodes");

    debug("sim: decoding text segment...");
    for (addr=ld_text_base;
         addr < (ld_text_base+ld_text_size);
         addr += SS_INST_SIZE)
      {
        inst = __UNCHK_MEM_ACCESS(SS_INST_TYPE,
addr);
        inst.a = SWAP_WORD(inst.a);
        inst.b = SWAP_WORD(inst.b);
        inst.a = (inst.a & ~0xff) | (unsigned
int)SS_OP_ENUM(SS_OPCODE(inst));
        __UNCHK_MEM_ACCESS(SS_INST_TYPE, addr) =
inst;
      }
  }

  /* initialize the DLite debugger */
  dlite_init(dlite_reg_obj, dlite_mem_obj, dlite_mstate_obj);
}


/* print simulator-specific configuration information */
void
```

```
sim_aux_config(FILE *stream)                    /* output stream */
{
  /* nothing currently */
}

/* dump simulator-specific auxiliary simulator statistics */
void
sim_aux_stats(FILE *stream)           /* output stream */
{
  /* nada */
}

/* un-initialize simulator-specific state */
void
sim_uninit(void)
{
  /* nada */
}


/*
 * configure the execution engine
 */

/*
 * precise architected register accessors
 */

/* next program counter */
#define SET_NPC(EXPR)               (next_PC = (EXPR))

/* target program counter */
#undef  SET_TPC
#define SET_TPC(EXPR)               (target_PC = (EXPR))

/* current program counter */
#define CPC                         (regs_PC)

/* general purpose registers */
#define GPR(N)                      (regs_R[N])
#define SET_GPR(N,EXPR)             (regs_R[N] = (EXPR))

/* floating point registers, L->word, F->single-prec, D->double-
prec */
#define FPR_L(N)                (regs_F.l[(N)])
#define SET_FPR_L(N,EXPR)       (regs_F.l[(N)] = (EXPR))
```

```
#define FPR_F(N)		(regs_F.f[(N)])
#define SET_FPR_F(N,EXPR)	(regs_F.f[(N)] = (EXPR))
#define FPR_D(N)		(regs_F.d[(N) >> 1])
#define SET_FPR_D(N,EXPR)	(regs_F.d[(N) >> 1] =
(EXPR))

/* miscellaneous register accessors */
#define SET_HI(EXPR)		(regs_HI = (EXPR))
#define HI			(regs_HI)
#define SET_LO(EXPR)		(regs_LO = (EXPR))
#define LO			(regs_LO)
#define FCC			(regs_FCC)
#define SET_FCC(EXPR)		(regs_FCC = (EXPR))

/* precise architected memory state help functions */
#define __READ_WORD(DST_T, SRC_T, SRC)
		\
  ((unsigned int)((DST_T)(SRC_T)MEM_READ_WORD(addr =
(SRC))))

#define __READ_HALF(DST_T, SRC_T, SRC)
		\
  ((unsigned int)((DST_T)(SRC_T)MEM_READ_HALF(addr =
(SRC))))

#define __READ_BYTE(DST_T, SRC_T, SRC)
		\
  ((unsigned int)((DST_T)(SRC_T)MEM_READ_BYTE(addr =
(SRC))))

/* precise architected memory state accessor macros */
#define READ_WORD(SRC)
		\
  __READ_WORD(unsigned int, unsigned int, (SRC))

#define READ_UNSIGNED_HALF(SRC)
		\
  __READ_HALF(unsigned int, unsigned short, (SRC))

#define READ_SIGNED_HALF(SRC)
		\
  __READ_HALF(signed int, signed short, (SRC))

#define READ_UNSIGNED_BYTE(SRC)
		\
  __READ_BYTE(unsigned int, unsigned char, (SRC))
```

```
#define READ_SIGNED_BYTE(SRC)
                \
  __READ_BYTE(signed int, signed char, (SRC))

#define WRITE_WORD(SRC, DST)
          \
  (MEM_WRITE_WORD(addr = (DST), (unsigned int)(SRC)))

#define WRITE_HALF(SRC, DST)
          \
  (MEM_WRITE_HALF(addr = (DST), (unsigned short)(unsigned
int)(SRC)))

#define WRITE_BYTE(SRC, DST)
          \
  (MEM_WRITE_BYTE(addr = (DST), (unsigned char)(unsigned
int)(SRC)))

/* system call handler macro */
#define SYSCALL(INST)                 (ss_syscall(mem_access,
INST))

/* instantiate the helper functions in the '.def' file */
#define
DEFINST(OP,MSK,NAME,OPFORM,RES,CLASS,O1,O2,I1,I2,I
3,EXPR)
#define DEFLINK(OP,MSK,NAME,MASK,SHIFT)
#define CONNECT(OP)
#define IMPL
#include "ss.def"
#undef DEFINST
#undef DEFLINK
#undef CONNECT
#undef IMPL

/* start simulation, program loaded, processor precise state
initialized */
void
sim_main(void)
{
  SS_INST_TYPE inst;
  register SS_ADDR_TYPE next_PC, target_PC;
  register SS_ADDR_TYPE addr;
  enum ss_opcode op;
  register int is_write;
```

```
  int stack_idx;

  fprintf(stderr, "sim: ** starting functional simulation **\n");

  /* set up initial default next PC */
  next_PC = regs_PC + SS_INST_SIZE;

  /* check for DLite debugger entry condition */
  if (dlite_check_break(regs_PC, /* no access */0, /* addr */0, 0,
0))
    dlite_main(regs_PC - SS_INST_SIZE, regs_PC,
sim_num_insn);

  while (TRUE)
    {
      /* maintain $r0 semantics */
      regs_R[0] = 0;

      /* keep an instruction count */
      sim_num_insn++;

      /* get the next instruction to execute */
      inst = __UNCHK_MEM_ACCESS(SS_INST_TYPE,
regs_PC);

      /* set default reference address and access mode */
      addr = 0; is_write = FALSE;

      /* decode the instruction */
      op = SS_OPCODE(inst);
      switch (op)
          {
#define
DEFINST(OP,MSK,NAME,OPFORM,RES,FLAGS,O1,O2,I1,I2,I
3,EXPR)        \
          case OP:                                          \
          EXPR;                                             \
          break;
#define DEFLINK(OP,MSK,NAME,MASK,SHIFT)
\
        case OP:                                            \
          panic("attempted to execute a linking opcode");
#define CONNECT(OP)
#include "ss.def"
#undef DEFINST
#undef DEFLINK
```

```
#undef CONNECT
        default:
          panic("bogus opcode");
    }

    if (SS_OP_FLAGS(op) & F_MEM)
        {
          sim_num_refs++;
          if (SS_OP_FLAGS(op) & F_STORE)
            is_write = TRUE;
        }

    if (SS_OP_FLAGS(op) & F_CTRL)
        {
          SS_ADDR_TYPE pred_PC;
          struct bpred_update update_rec;

          sim_num_branches++;

          if (pred)
            {
              /* get the next predicted fetch address */
              pred_PC = bpred_lookup(pred,
                                     /* branch addr */regs_PC,
                                     /* target */target_PC,
                                     /* opcode */op,
                                     /* jump through R31? */(RS) ==
31,
                                     /* stash an update ptr
*/&update_rec,
                                     /* stash return stack ptr
*/&stack_idx,
                                     /* instruction for FFBPANN */
inst);

              /* valid address returned from branch predictor? */
              if (!pred_PC)
                  {
                    /* no predicted taken target, attempt not taken
target */
                    pred_PC = regs_PC + sizeof(SS_INST_TYPE);
                  }

              bpred_update(pred,
                           /* branch addr */regs_PC,
                           /* resolved branch target */next_PC,
```

```
                                /* taken? */next_PC != (regs_PC +
    sizeof(SS_INST_TYPE)),
                                /* pred taken? */pred_PC != (regs_PC +
            sizeof(SS_INST_TYPE)),
                                /* correct pred? */pred_PC == next_PC,
                                /* opcode */op,
                                /* jump through R31? */(RS) == 31,
                                /* predictor update pointer */&update_rec,
                        /* instruction for FFBPANN */ inst);

              }
            }

          /* check for DLite debugger entry condition */
          if (dlite_check_break(next_PC,
                                is_write ? ACCESS_WRITE :
    ACCESS_READ,
                                addr, sim_num_insn, sim_num_insn))
            dlite_main(regs_PC, next_PC, sim_num_insn);

          /* go to the next instruction */
          regs_PC = next_PC;
          next_PC += SS_INST_SIZE;
        }
    }
```

Makefile

```
    #
    # Makefile - simulator suite make file
    #
    # This file is a part of the SimpleScalar tool suite written by
    # Todd M. Austin as a part of the Multiscalar Research Project.
    #
    # The tool suite is currently maintained by Doug Burger and Todd
    M. Austin.
    #
    # Copyright (C) 1994, 1995, 1996, 1997 by Todd M. Austin
    #
    # This source file is distributed "as is" in the hope that it will be
    # useful.  It is distributed with no warranty, and no author or
    # distributor accepts any responsibility for the consequences of its
    # use.
    #
```

```
# Everyone is granted permission to copy, modify and redistribute
# this source file under the following conditions:
#
#   This tool set is distributed for non-commercial use only.
#   Please contact the maintainer for restrictions applying to
#   commercial use of these tools.
#
#   Permission is granted to anyone to make or distribute copies
#   of this source code, either as received or modified, in any
#   medium, provided that all copyright notices, permission and
#   nonwarranty notices are preserved, and that the distributor
#   grants the recipient permission for further redistribution as
#   permitted by this document.
#
#   Permission is granted to distribute this file in compiled
#   or executable form under the same conditions that apply for
#   source code, provided that either:
#
#   A. it is accompanied by the corresponding machine-readable
#      source code,
#   B. it is accompanied by a written offer, with no time limit,
#      to give anyone a machine-readable copy of the corresponding
#      source code in return for reimbursement of the cost of
#      distribution.  This written offer must permit verbatim
#      duplication by anyone, or
#   C. it is distributed by someone who received only the
#      executable form, and is accompanied by a copy of the
#      written offer of source code that they received concurrently.
#
# In other words, you are welcome to use, share and improve this
# source file.  You are forbidden to forbid anyone else to use, share
# and improve what you give them.
#
# INTERNET: dburger@cs.wisc.edu
# US Mail:  1210 W. Dayton Street, Madison, WI 53706
#
# $Id: Makefile,v 1.6 1997/04/16 22:08:40 taustin Exp taustin $
#
# $Log: Makefile,v $
# Revision 1.6  1997/04/16  22:08:40  taustin
# added standalone loader support
#
# Revision 1.5  1997/03/11  01:04:13  taustin
# updated copyright
# CC target now supported
# RANLIB target now supported
```

```
# MAKE target now supported
# CFLAGS reorganized
# MFLAGS and MLIBS to improve portability
#
# Revision 1.1  1996/12/05  18:56:09  taustin
# Initial revision
#
#

##########################################################
############
#
# Modify the following definitions to suit your build environment,
# NOTE: most platforms should not require any changes
#
##########################################################
############

#
# Insert your favorite C compiler, here.  NOTE: the SimpleScalar
simulators
# must be compiled with an ANSI C compatible compiler.
#
CC = gcc

#
# Insert the name of RANLIB tool here, most machines can simply
use "ranlib"
# machines that do not use RANLIB, should put "true" here
#
RANLIB = ranlib

#
# Insert your favorite make tool, here.  Most anything should work.
#
MAKE = make

#
# Compilation-specific feature flags
#
# -DDEBUG   - turns on debugging features
# -DBFD_LOADER   - use libbfd.a to load programs (also
required BINUTILS_INC
#                  and BINUTILS_LIB to be defined, see below)
#
FFLAGS = -DDEBUG
```

```
#
# Choose your optimization level here
#
# for optimization:      OFLAGS = -O2 -g -finline-functions -
funroll-loops
# for debug:             OFLAGS = -g -Wall
#
#OFLAGS = -O
OFLAGS = -g

#
# Point the Makefile to your Simplescalar-based bunutils, these
definitions
# should indicate where the include and library directories reside.
# NOTE: these definitions are only required if BFD_LOADER is
defined.
#
#BINUTILS_INC = -I../include
#BINUTILS_LIB = -L../lib


##########################################################
############
#
# YOU SHOULD NOT NEED TO MODIFY ANYTHING
BELOW THIS COMMENT
#
##########################################################
############

#
# Machine-specific flags and libraries, generated by sysprobe
#
MFLAGS = `./sysprobe -flags`
MLIBS  = `./sysprobe -libs`

#
# complete flags
#
CFLAGS = $(MFLAGS) $(FFLAGS) $(OFLAGS)
$(BINUTILS_INC) $(BINUTILS_LIB)

#
# all the sources
#
```

```
SIM_SRC = main.c sim-fast.c sim-safe.c sim-cache.c sim-profile.c
\
        sim-bpred.c sim-cheetah.c sim-outorder.c syscall.c
memory.c \
        regs.c loader.c cache.c bpred.c ptrace.c eventq.c
resource.c \
        endian.c dlite.c symbol.c eval.c options.c range.c stats.c \
        ss.c endian.c misc.c ffbpann.c
SIM_HDR = syscall.h memory.h regs.h sim.h loader.h cache.h
bpred.h ptrace.h \
        eventq.h resource.h endian.h dlite.h symbol.h eval.h
bitmap.h \
        range.h version.h ss.h ss.def endian.h ecoff.h misc.h
ffbpann.h

#
# common objects
#
SIM_OBJ = main.o syscall.o memory.o regs.o loader.o ss.o
endian.o dlite.o \
        symbol.o eval.o options.o stats.o range.o misc.o

#
# external libraries required for build
#
SIM_LIB = -lm

#
# all targets
#
all: sim-fast sim-safe sim-bpred sim-profile sim-cheetah sim-cache
sim-outorder
        @echo "my work is done here..."

sysprobe:       sysprobe.c
        $(CC) $(FFLAGS) -o sysprobe sysprobe.c
        @echo endian probe results: `./sysprobe -s`
        @echo probe flags: `./sysprobe -flags`
        @echo probe libs: `./sysprobe -libs`

sim-fast:       sysprobe sim-fast.o $(SIM_OBJ)
        $(CC) -o sim-fast $(CFLAGS) sim-fast.o $(SIM_OBJ)
$(SIM_LIB) $(MLIBS)

sim-safe:       sysprobe sim-safe.o $(SIM_OBJ)
        $(CC) -o sim-safe $(CFLAGS) sim-safe.o $(SIM_OBJ)
```

```
$(SIM_LIB) $(MLIBS)

sim-profile:	sysprobe sim-profile.o $(SIM_OBJ)
	$(CC) -o sim-profile $(CFLAGS) sim-profile.o
$(SIM_OBJ) $(SIM_LIB) $(MLIBS)

sim-bpred:	sysprobe sim-bpred.o bpred.o ffbpann.o
$(SIM_OBJ)
	$(CC) -o sim-bpred $(CFLAGS) sim-bpred.o bpred.o
ffbpann.o $(SIM_OBJ) $(SIM_LIB) $(MLIBS)

sim-cheetah:	sysprobe sim-cheetah.o libcheetah/libcheetah.a
$(SIM_OBJ)
	$(CC) -o sim-cheetah $(CFLAGS) sim-cheetah.o
$(SIM_OBJ) libcheetah/libcheetah.a $(SIM_LIB) $(MLIBS)

sim-cache:	sysprobe sim-cache.o cache.o $(SIM_OBJ)
	$(CC) -o sim-cache $(CFLAGS) sim-cache.o cache.o
$(SIM_OBJ) $(SIM_LIB) $(MLIBS)

sim-outorder:	sysprobe sim-outorder.o cache.o bpred.o ffbpann.o
resource.o ptrace.o $(SIM_OBJ)
	$(CC) -o sim-outorder $(CFLAGS) sim-outorder.o cache.o
bpred.o ffbpann.o resource.o ptrace.o $(SIM_OBJ) $(SIM_LIB)
$(MLIBS)

libcheetah/libcheetah.a: libcheetah/ascbin.c libcheetah/din.c
libcheetah/dmvl.c libcheetah/faclru.c libcheetah/facopt.c
libcheetah/libcheetah.c libcheetah/pixie.c libcheetah/ppopt.c
libcheetah/saclru.c libcheetah/sacopt.c libcheetah/util.c
	cd libcheetah; $(MAKE) "MAKE=$(MAKE)"
"CC=$(CC)" "RANLIB=$(RANLIB)" "CFLAGS=$(FFLAGS)
$(OFLAGS)" libcheetah.a

.c.o:
	$(CC) $(CFLAGS) -c $*.c

filelist:
	@echo $(SIM_SRC) $(SIM_HDR) Makefile

diffs:
	-rcsdiff RCS/*

sim-tests: sysprobe sim-fast sim-safe sim-cache sim-cheetah sim-
bpred sim-profile sim-outorder
	cd tests; $(MAKE) "MAKE=$(MAKE)" tests
```

```
"SIM_DIR=.." "SIM_BIN=sim-fast"
	cd tests; $(MAKE) "MAKE=$(MAKE)" tests
"SIM_DIR=.." "SIM_BIN=sim-safe"
	cd tests; $(MAKE) "MAKE=$(MAKE)" tests
"SIM_DIR=.." "SIM_BIN=sim-cache"
	cd tests; $(MAKE) "MAKE=$(MAKE)" tests
"SIM_DIR=.." "SIM_BIN=sim-cheetah"
	cd tests; $(MAKE) "MAKE=$(MAKE)" tests
"SIM_DIR=.." "SIM_BIN=sim-bpred"
	cd tests; $(MAKE) "MAKE=$(MAKE)" tests
"SIM_DIR=.." "SIM_BIN=sim-profile" "SIM_OPTS=-all"
	cd tests; $(MAKE) "MAKE=$(MAKE)" tests
"SIM_DIR=.." "SIM_BIN=sim-outorder"

sim-tests-nt: sysprobe sim-fast sim-safe sim-cache sim-cheetah
sim-bpred sim-profile sim-outorder
	cd tests; $(MAKE) "MAKE=$(MAKE)" tests
"SIM_DIR=.." "REDIR=redir.bash" "SIM_BIN=sim-fast"
	cd tests; $(MAKE) "MAKE=$(MAKE)" tests
"SIM_DIR=.." "REDIR=redir.bash" "SIM_BIN=sim-safe"
	cd tests; $(MAKE) "MAKE=$(MAKE)" tests
"SIM_DIR=.." "REDIR=redir.bash" "SIM_BIN=sim-cache"
	cd tests; $(MAKE) "MAKE=$(MAKE)" tests
"SIM_DIR=.." "REDIR=redir.bash" "SIM_BIN=sim-cheetah"
	cd tests; $(MAKE) "MAKE=$(MAKE)" tests
"SIM_DIR=.." "REDIR=redir.bash" "SIM_BIN=sim-bpred"
	cd tests; $(MAKE) "MAKE=$(MAKE)" tests
"SIM_DIR=.." "REDIR=redir.bash" "SIM_BIN=sim-profile"
"SIM_OPTS=-all"
	cd tests; $(MAKE) "MAKE=$(MAKE)" tests
"SIM_DIR=.." "REDIR=redir.bash" "SIM_BIN=sim-outorder"

clean:
	rm -f *.o *.exe core *~ Makefile.bak sim-fast sim-safe sim-
profile \
		sim-bpred sim-cheetah sim-cache sim-outorder
sysprobe
	cd libcheetah; $(MAKE) clean
	cd tests; $(MAKE) clean

unpure:
	rm -f sim.pure *pure*.o sim.pure.pure_hardlink
sim.pure.pure_linkinfo

depend:
	makedepend -n $(BINUTILS_INC) $(SIM_SRC)
```

```
# DO NOT DELETE THIS LINE -- make depend depends on it.

main.o: misc.h regs.h ss.h ss.def memory.h endian.h options.h
stats.h eval.h
main.o: loader.h version.h dlite.h sim.h
sim-fast.o: misc.h ss.h ss.def regs.h memory.h endian.h options.h
stats.h
sim-fast.o: eval.h loader.h syscall.h dlite.h sim.h
sim-safe.o: misc.h ss.h ss.def regs.h memory.h endian.h options.h
stats.h
sim-safe.o: eval.h loader.h syscall.h dlite.h sim.h
sim-cache.o: misc.h ss.h ss.def regs.h memory.h endian.h options.h
stats.h
sim-cache.o: eval.h cache.h loader.h syscall.h dlite.h sim.h
sim-profile.o: misc.h ss.h ss.def regs.h memory.h endian.h
options.h stats.h
sim-profile.o: eval.h loader.h syscall.h dlite.h symbol.h sim.h
sim-bpred.o: misc.h ss.h ss.def regs.h memory.h endian.h options.h
stats.h
sim-bpred.o: eval.h loader.h syscall.h dlite.h bpred.h sim.h
ffbpann.h
sim-cheetah.o: misc.h ss.h ss.def regs.h memory.h endian.h
options.h stats.h
sim-cheetah.o: eval.h loader.h syscall.h dlite.h
libcheetah/libcheetah.h
sim-cheetah.o: sim.h
sim-outorder.o: misc.h ss.h ss.def regs.h memory.h endian.h
options.h stats.h
sim-outorder.o: eval.h cache.h loader.h syscall.h bpred.h
resource.h bitmap.h
sim-outorder.o: ptrace.h range.h dlite.h sim.h ffbpann.h
syscall.o: misc.h ss.h ss.def regs.h memory.h endian.h options.h
stats.h
syscall.o: eval.h loader.h sim.h syscall.h
memory.o: misc.h ss.h ss.def loader.h memory.h endian.h
options.h stats.h
memory.o: eval.h regs.h
regs.o: misc.h ss.h ss.def loader.h memory.h endian.h options.h
stats.h
regs.o: eval.h regs.h
loader.o: ecoff.h misc.h ss.h ss.def regs.h memory.h endian.h
options.h
loader.o: stats.h eval.h sim.h loader.h
cache.o: misc.h ss.h ss.def cache.h memory.h endian.h options.h
```

stats.h
cache.o: eval.h
bpred.o: misc.h ss.h ss.def bpred.h stats.h eval.h
ptrace.o: misc.h ss.h ss.def range.h ptrace.h
eventq.o: misc.h ss.h ss.def eventq.h bitmap.h
resource.o: misc.h resource.h
endian.o: loader.h ss.h ss.def memory.h endian.h options.h stats.h
eval.h
dlite.o: misc.h version.h eval.h regs.h ss.h ss.def memory.h
endian.h
dlite.o: options.h stats.h sim.h symbol.h loader.h range.h dlite.h
symbol.o: ecoff.h misc.h loader.h ss.h ss.def memory.h endian.h
options.h
symbol.o: stats.h eval.h symbol.h
eval.o: misc.h eval.h
options.o: misc.h options.h
range.o: misc.h ss.h ss.def symbol.h loader.h memory.h endian.h
options.h
range.o: stats.h eval.h range.h
stats.o: misc.h eval.h stats.h
ss.o: misc.h ss.h ss.def
endian.o: loader.h ss.h ss.def memory.h endian.h options.h stats.h
eval.h
misc.o: misc.h
ffbpann.o: ffbpann.h misc.h

BIBLIOGRAPHY

BIBLIOGRAPHY

[Anderson] Anderson, Paul and Gail Anderson. Advanced C Tips and Techniques. Indianapolis: Hayden Books, 1988.

[Austin] Austin, Todd and Doug Burger. SimpleScalar Tutorial. [Online] Available http://www.cs.wisc.edu/~mscalar/ss/tutorial.html, January 1998.

[Austin97] Austin, Todd. A User's and Hacker's Guide to the SimpleScalar Architectural Research Toolset. January, 1997.

[Brehob] Brehob, Mark, Travis Doom, Richard Enbody, William H. Moore, Sherry Q. Moore,Ron Sass, and Charles Severance. Beyond RISC - The Post-RISC Architecture. [Online] Available http://www.egr.msu.edu/~crs/papers/postrisc2/, August 2, 1999.

[Demuth] Demuth, Howard and MArk Beale. Neural Network Toolbox User's Guide. Natick, Massachusetts: The MathWorks, Inc., 1994.

[Driesen] Driesen, Karel and Urs Holzle. Accurate Indirect Branch Prediction. The 25th International Symposium on Computer Architecture. IEEE, Inc., 1998.

[Emer] Emer, Joel and Nikolas Gloy. A Language for Describing Predictors and its Application to Automatic Synthesis. The 24th Annual International Symposium on Computer Architecture. Association for Computing Machinery Press, 1997.

[Haykin] Haykin, Simon. Neural Networks: A Comprehensive Foundation. New Jersey: Prentice-Hall, Inc., 1994.

[Hsieh]Hsieh, Paul. Sixth Generation CPU Comparisons. [Online] Available http://www.azillionmonkeys.com/qed/cpuwar.html, August 2, 1999.

[Intel] Intel Corporation. Micoprocessor Quick Reference Guide. [Online] Available http://www.intel.com/pressroom/kits/quickrefyr.htm, August 8, 2001.

[Kandel] Kandel, Eric; Schwartz, James; and Thomas, Jessell. Essentials of Neural Science and Behavior. Appleton & Lange. Norwalk, Connecticut, 1995.

[Kutza] Kutza, Karsten. Neural Networks at Your Fingertips. [Online] Available http://www.geocities.com/CapeCanaveral/1624/, July 2,1999.

[Patterson] Patterson, David A. and John L. Hennessy. Computer Architecture A Quantitative Approach. San Francisco: Morgan Kaufmann Publishers, Inc., 1996.

[SimpleScalar]The SimpleScalar Architectural Research Tool Set, Version 2.0 [Software] http://www.cs.wisc.edu/~mscalar/simplescalar.html

[Wang] Wang, Yiwen. CMOS VLSI Implementations of a New Feedback Neural Network Architecture. diss. Michigan State University, 1991.

[West] West, Patricia M., Patrick L. Brockett, Linda L. Golden. “A Comparative Analysis of Neural Networks and Statistical Methods for Predicting Consumer Choice.” Marketing Science, vol.16, no. 4, 1997, pp.370-391.

[Yeh93] Yeh, Tse-Yu and Yale Patt. A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History. The 20th Annual International Symposium on Computer Architecture. Los Alimitos, CA: IEEE Computer Society Press, 1993.

[Yeh92] Yeh, Tsu-Yu and Yale N. Patt. Alternative Implementations of 2-Level Adaptive Branch Prediction. The 19th Annual International Symposium on Computer Architecture. Association for Computing Machinery, 1992.