A MIPS R10000-LIKE OUT-OF-ORDER MICROPROCESSOR IMPLEMENTATION IN VERILOG HDL

A Design Project Report

Presented to the Engineering Division of the Graduate School

Of Cornell University

In Partial Fulfillment of the Requirements for the Degree of

Master of Engineering (Electrical)

By

Scott Thomas Bingham

Project Advisor: Dr. Bruce R Land

Degree Date: May, 2007

Abstract

Master of Electrical Engineering Program
Cornell University
Design Project Report

Project Title: A MIPS R10000-Like Out-Of-Order Microprocessor Implementation in Verilog HDL

Author: Scott Thomas Bingham

Abstract:

Microprocessors have evolved greatly over the past few decades from single cycle state machines, to pipelined architectures, to wide issue superscalar processors to out of order execution engines. This project implements one such out-of-order processor using the MIPS instruction set architecture taught in ECE314 and ECE475. Because each of those classes culminates in a microprocessor design in Verilog HDL, those implementations provide a good performance baseline for comparison of this design. This microprocessor was designed to exploit instruction level parallelism as much as possible while still maintaining reasonable amount of logic per clock cycle. Ultimately the design implemented is capable of fetching, decoding, renaming, issuing, executing, and retiring up to four instructions per clock cycle with speculative execution; this results in a performance improvement of almost 3x over the two-way superscalar MIPS processor implemented in ECE475. Upon successful implementation, the processor was variably configured to try and gauge the effects of each component on performance.

Report Approved by
Project Advisor: ____Date:______________

Executive Summary

A major breakthrough in microprocessor architecture was the realization that independent instructions can be executed in parallel and that there exists a significant amount of parallelism in most code. The first step to exploiting this is to pipeline a datapath so that multiple instructions can execute at a time. However, there is further parallelism that can be exploited by replicating the datapath and executing independent instructions in parallel. Even this does not take advantage of all the parallelism, however. Taking this a step further, we can fetch a large number of instructions and analyze them at once and execute those for which we already have the operands, regardless of their actual ordering in the program.

However, this executing based purely on real data flow introduces several design issues. First, we must be able to supply the execution engine with a sufficiently large number of instructions so that parallelism can be extracted. Once these instructions have been fetched, we next need to decode them to see what operations they actually perform and where data flow hazards exist. Next, these instructions must be renamed to eliminate hazards where the register names are reused but no data flows between the instructions. These instructions can then be placed in a queue that issues instructions for execution as their operands become available. To maximize throughput, dependant instructions need to be executed immediately subsequent to their operands becoming available. Even when this is achieved, there exist further problems. Almost one in five instructions is a control flow instruction that can change the execution path of the program. Without knowing which way to go, we would have to stall until the data is available. This goes contrary to the desire to achieve high throughput and so we make an educated guess based on the past behavior of the program as to which way to actually go. However, our guesses may not be good enough all the time and so mechanisms must be provided to check the guesses and recover before we irreversibly alter the execution of the program.

This project implements a processor in Verilog HDL that does the aforementioned steps and attempts to improve performance as much as possible under the constraints of the instruction set architecture and program semantics. In ideal conditions, the microprocessor implemented is capable of sustaining the execution of four instructions per cycle. While these conditions are rare in all but trivial programs meant to exploit this scheduling, this implementation still yields a nearly 3x improvement over a two-way superscalar processor previously implemented in ECE475. The major components implemented in this design are:

- instruction and data cache interfaces
- an instruction decoder which removes the now unnecessary branch delay slot imposed by the ISA for backwards compatibility
- register allocation tables and a register renamer
- a 64 entry physical register file, a free register list, and ready operand tracking
- four execution units (2 general purpose ALU’s, one dedicated branch/jump resolver and one address calculation unit) and issue queues for these execution units
- a load store queue used to track memory dependencies and forward data when possible to avoid a cache access
- forwarding units to maintain back to back execution of dependant instructions
- a reorder buffer to maintain the sequential consistency of a program

The resulting design is described by over six thousand lines of Verilog code and was verified using the `ncverilog` simulator and compiled C micro-benchmarks.

Table of Contents

I. Design Problem Overview

The first microprocessors evaluated one instruction at a time, where each instruction could take one or many cycles for the state machine to complete. The next step in CPU design came when researchers realized that different pieces of the state machine were needed at different points of the instructions execution and that it was possible to pipeline these components such that many instructions could be executed at a time in different parts of the state machine provided that they were independent of each other. While the latency of each individual instruction increases due to pipelining for various reasons, such as the overhead associated with registers separating each stage and the clock speed being determined by the slowest stage, this realization of instruction level parallelism (ILP) has driven massive performance improvements over the past few decades.

Exploiting ILP has some serious limitations. Very rarely is code written where each instruction is independent of those around it. It is almost always the case that one calculation feeds another which feeds another creating a string of data dependencies. Coupled with a limited number of registers in an instruction set architecture (ISA), four types of data hazards arrive: read-after-write (RAW), write-after-read (WAR), write-after-write (WAW), and read-after-read (RAR). The RAW dependency is the only real dependency in that data flows. The middle two after merely dependencies in name only. With a sufficiently large number of registers, they can be resolved. These WAR and WAW dependencies are not of concern to a pipelined processor that performs each instruction in order because any reader will have read the value before the writer overwrites it and subsequent writes will happen in order. The final dependency represents no dependency at all because reading a register does not modify its contents; however it is included for completeness.

Other hazards arise from pipelining the CPU. Structural hazards arise from the fact that multiple instructions in the pipeline may need to perform an addition, for example. This hazard is typically resolved through duplication of hardware or through scheduling of resource usage. The final hazard that arises from pipelining is control flow hazards. Approximately 20% of a program (1 in 5 instructions) is an instruction that alters the control flow of the program execution. These conditional branches and jumps often need the results of previously executed instructions to determine which path to take. The resulting problem is quite serious: you must wait until the condition has been resolved or guess. Early pipelined microprocessors either waited or made a static guess, typically that the branch was not taken because if so, you already know where to fetch instructions from next. This lead to the creation of the branch delay slot, where the instruction following the branch is always executed regardless of the path the branch takes. Essentially, you can execute this instruction "for free" because you would have been waiting otherwise. This delay slot becomes a serious issue in future hardware improvements where it is no longer needed but must be supported for backwards compatibility with legacy code.

If we consider pipelining the CPU to be parallelism in time, the next exploitation of ILP may be considered parallelism in space. To further exploit ILP, the next generation of CPU's executed several instructions in parallel in each stage of the execution. These superscalar CPU's could execute 2, 4, or more instructions in parallel as well as pipelining the execution. Once again, however, there is only so much parallelism that can be realized through this method because the instructions executed simultaneously must be independent of each other and only neighboring instructions are considered for execution. To avoid structural hazards, the execution engine (the back-end of the CPU; fetching and decoding are considered the front-end), is replicated, though sometimes not completely, to provide parallelism in instructions. Somewhat sophisticated scheduling is needed to selectively issue instructions to execute or to wait until the instructions they depend on have produced results.

The current state of the microprocessor is what this project addresses: out of order execution. To further exploit ILP, a large number of instructions are fetched and considered for execution. Those that have their operands ready are issued for execution even if they were fetched after the other instructions. This allows execution of instructions based purely on data dependencies and not on adjacency, although limited by the size of the window of instructions considered for execution. This project implements such an out of order processor in Verilog HDL using a subset of the MIPS ISA, particularly the majority of the integer instructions. These are sufficient to execute most non-floating point programs, compiled from C source code in my case. Execution correctness and performance was evaluated in the `ncverilog` Cadence Simulator on the Linuxpool/AMDPool. ECE314 implements the canonical five stage pipeline for the same MIPS ISA subset. ECE475 implements a two-way superscalar processor, again using MIPS. However, no subsequent computer architecture class at Cornell University implements an out of order processor. This "gap" in the implementation knowledge is the motivation for this project which implements an R10000-like MIPS CPU. The subsequent sections give a brief overview of the goals of the project and how each of the previously implemented processor works or differs from the others. The implementations from previous ECE classes provide a baseline benchmark for the out of order processor.

I.A. Project Goals

Initially, the goal of the project was to implement an out-of-order CPU on the DE2 FPGA board used in ECE576. However, due to the immense complexity of the design (over 6,000 lines of Verilog code, not including header files), the limited debugging capabilities of the FPGA, and the limited size of the FPGA, it was determined that the functionality of the design would be better verified using the `ncverilog` simulator rather than scaling down the design to fit on an FPGA. This allows for a wider range of benchmarks to be tested and functionality to be verified for real world applications. Ultimately, the goal of this project is to implement an R10000-like out of order MIPS CPU in Verilog HDL with the following capabilities:

- Execute same subset of MIPS ISA implemented by ECE314 and ECE475 CPU's
- Achieve significantly higher IPC throughput than ECE314 and ECE475 CPU's
- Execute instructions out-of-order but commit in program order to exploit ILP but maintain sequential consistency
- Execute instructions as fast as structural, data, and control hazards permit
- Schedule instruction execution for at least 2 execution units
- Predict control flow direction and speculatively execute until control flow resolved and recover if prediction incorrect
- Fetch, decode, rename, execute, and retire up to 4 instructions per cycle

The processor implementation should minimally have the following structures:

- Separate Instruction and Data Memory
- Instruction Decoder

- Branch Predictor / Return Address Stack
- Register Renaming / Register Allocation Table
- Issue Queues
- Reorder Buffer
- Physical Register File with Size Greater than Logical Register File
- 2 Arithmetic Logic Units with Data Forwarding

Upon successful implementation of the above, additional structures were added to improve performance, such as a load-store queue. Performance tweaks were made to various structures to squeeze as much performance as possible out of the CPU while maintaining realistic workloads per cycle. Several branch predictors are also implemented to measure their accuracy and its effect on performance. Structure sizes and throughputs are varied as well to measure their effect on performance. Finally, micro-benchmarks were written in C to measure performance to run along with benchmarks from other ECE computer architecture courses to measure correctness and performance.

I.B. Canonical Five Stage Pipelined Processor

The standard MIPS pipeline is divided into five stages: instruction fetch (IF), instruction decode (ID), execution (EX), memory (MEM), and write back (WB). The functionality of each stage and its components will be described briefly so that a comparison can be made to the out of order implementation. The interface to memory is uniform to all three processors, although heavily modified to suit the needs of each processor. This yields a more accurate comparison as to the performance of each core by keeping the memory system consistent.

Instruction Fetch

The instruction fetch stage is responsible for retrieving instructions from memory or an instruction cache. Where to fetch is determined by the program counter (PC) which maintains the memory address of the current instruction. For straight-line code execution, the PC is incremented by four (4 byte instruction words, 32 bit PC) every cycle. However, when a branch is determined to be taken or a jump is taken, the target of these control flow instructions is loaded into the PC instead. Because the control flow instructions are not evaluated until the first half of the EX stage, the processor increments the PC after a control flow instruction to fetch the delay slot. The PC is negative edge triggered so that the target can be calculated in the first half of the cycle resulting in a single delay slot. The instruction memory has to fetch the target instruction and pass it to the next stage where it is decoded before the next falling edge of the clock. To stall this processor, all that must be done is to freeze the PC and pass a NOP to the ID stage. If a physically addressed or tagged instruction cache is used, this stage will also contain a translation look-aside buffer (TLB). However, in my simulation, there is no operating system (O/S); hence, there is only one process running on the processor at a time. For this reason, virtually addressed and tagged caches are used, and there is no need for a TLB. An instruction cache miss need not stall the processor. Since it will have no useful work to do other than finish the current instructions while it waits, it is simpler to stall until the miss is resolved.

Instruction Decode

The instruction decode stage contains a control unit that decodes instructions depending on their type to determine what operations the rest of the processor should perform as the instruction enters each stage. The source and destination registers are determined, and any immediate instructions have their immediate values extracted from the instruction and sign extended if needed. On the first half of the cycle, the register file that resides in this stage is updated with previously calculated values. In the second half of the cycle, the instruction reads its operands from the register file. This ordering eliminates any RAW hazards that may have been present between the current instruction and the one three previous. The fetch and decode stages are collectively the front end of the processor. When a system call enters this stage, the front end stalls so that the back end of the processor (EX, MEM, and WB) can write all values back to the register file before the system call is allowed to proceed so that the register file is fully updated before jumping to the O/S to handle the system call. Because our simulator does not run an O/S, system calls are emulated with function calls to C libraries. This emulation requires the register file be completely updated with the values from all instructions ahead of the system call. The system call is then performed when the system call enters the EX stage, and the return values are immediately written into the register file.

Execution

The execution stage contains the ALU which performs arithmetic and logical operations and calculates effective addresses for loads and stores. Also in the EX stage is a branch comparator that evaluates branch conditions to determine the direction that the processor should take. The target of PC relative jumps and branches is calculated at the same time. Depending on the direction of the branch (always taken for jumps), the PC is then updated to fetch from the branch target or the fall through path next. In the MIPS ISA, multiplies and divides do not write to the regular register file, rather to two dedicated registers HI and LO which can only be accessed with special instructions MFHI and MFLO, move from high and move from low, respectively. Because multiplies and divides can take multiple cycles to complete and their results can only be retrieved with special instructions, the processor can continue to execute while the multiply/divide happens off the path and only needs to stall if MFHI or MFLO are seen before the calculation is completed. The infrequency of multiplies and divides and the fact that they write to the same registers means that they share the same block and only one of either can occur at a time. However, because they always write to HI and LO registers, when a new multiply or divide comes upon the execution unit while it is busy, it simply starts the next multiply/divide because there is guaranteed to be no consumers of the value being currently calculated. To resolve RAW hazards, data forwarding or bypassing is used. If the operands to the current instruction were calculated in either of the two previous instructions, they were not in the register file when the current instruction read it. Therefore, these values are passed back to the execution stage so that stalling is not needed. The newest value (MEM before WB, WB before register file) is used if there is a dependency. Because loads do not evaluate until the next stage, their value can not be passed back in time for the execution stage. Therefore, the MIPS ISA introduces a load delay slot similar to the branch delay slot in which the instruction following a load cannot consume its result.

Memory

The memory stage contains a data cache (and another TLB if physical tags or indexes are used). All three implementations discussed here use the same basic memory interface. Separate L1 caches are present for instructions and data but share a main memory (there is no L2). Accesses and tag checks to the L1 caches are emulated in a C library. The interface to the cache is responsible for checking the tag and valid bit of accesses. If there is a miss, a memory state machine stalls the processor, retrieves the data values from main memory, updates the cache, and retries the access before proceeding. Because stores can happen on the word, half word, and byte levels, input store values must be properly masked to only modify the desired addresses. Similarly, loads can happen on the word, half word, and bytes levels, and can be either signed or unsigned. All loads are therefore treated as word accesses to the cache and the result is masked, shifted, and sign extended as required by the instruction. The load value or the ALU result if the instruction was not a load, are then passed to the WB stage. Because of the assembly line nature of this processor, a data cache miss stops the entire pipeline until it is resolved. If both the data cache and the instruction cache misses occur "at the same time," the instruction cache fill is performed first because it happens just after the negative edge and will be triggered before data access which comes shortly after the positive edge. If a data cache miss was to occur first, there could be no instruction cache miss because the PC would have been stalled before it could update to a new address.

Write-back

In the final stage, there is very little to do other than forward data values back to the register file and execution stage. The resulting value can either be from the ALU, memory, or a link address for branches and jumps that link. Linking saves the point in the program which should be saved to the return address register. This is mostly used for function calls so the program knows where to return upon exiting the function. In the MIPS ISA, the link address is the instruction after the delay slot or PC+8 from the control flow instruction.

I.C. Two-way Superscalar Processor

Because the superscalar processor is similar in most respects to the simple five-stage pipelined CPU, only the differences involved with duplicating pipeline will be discussed. There are two execution pipelines (A and B), both of which can execute ALU instructions. However, the A pipeline is responsible for processing control flow instructions and system calls. The B pipe handles all data memory accesses. This division of labor is instituted because only one control flow and only one data memory access can happen at a time.

Instruction Fetch

The instruction memory now must fetch two instructions per cycle, the current PC and the subsequent instruction. The simplest implementation requires that accesses be double word aligned; this limitation can lead to wasted fetch slots when a branch or jump targets the second word of a double word block or if only one instruction of the pair could be issued at a time. Therefore, logic is added to the memory interface to access both PC and PC+4 across two cache lines if needed. This could require handling two cache misses before being able to proceed as a side effect. The next PC to fetch is still determined by control flow instructions (in the EX stage of pipe A) but can also be determined by the superscalar issue logic in the ID stage.

Instruction Decode

The majority of the changes to make the processor superscalar are instituted in this stage. Superscalar issue logic determines which instruction to issue down which pipe. The restrictions of control flow down pipe A and data accesses down pipe B are handled here. Data hazards such as RAW and WAW now must be scheduled to be avoided. Because there is no active forwarding between EX stages, a dependent instruction cannot be issued with its producer. Similarly, if both instructions write to the same register, they cannot issue at the same time. This implementation holds the B instruction for the next cycle rather than trying to resolve whether or not it was necessary to issue the A instruction in the first place if they both write to the same register. System calls now require both pipes to be drained before issuing by themselves to the EX stage. The superscalar logic is also responsible for ensuring that branch delay slots get fetched in the next cycle if not fetched with the branch. Any subsequent instructions need to be squashed in the event of a jump or branch being taken. The simplest implementation would issue the second instruction alone if it could not be issued with the first instruction. This implementation, however, saves the instruction that was not issued and takes a new one from the next fetch block. The first instruction need not go down pipe A if swapping the two instructions allows the both to issue together. The load delay slot must be still respected in the scheduling of consumers even though they may be fetched on the next cycle. The occurrence of a MFHI or MFLO instruction while the multiply/divide block is busy causes a stall. Finally, the number of read and write ports on the register file are doubled to allow for two instructions reading and writing simultaneously.

Execution / Memory / Write-back
The changes in these stages are fairly simple as the scheduling is handled in the ID stage. The restrictions of what instructions can enter which pipeline have already been discussed. There is still only one multiply/divide unit for the reason that any subsequent multiply/divide would overwrite the first one or have to wait while the MFHI / MFLO instructions stall waiting for their results. For greater flexibility in scheduling, multiplies / divides can be issued to either pipe. While it is possible to continue executing around a data cache miss now, this was not implemented due to the complexity of scheduling around dependencies. The final point about the superscalar processor is that the number of forwarding paths has doubled, from MEM-A, WB-A, MEM-B, or WB-B.

I.D. MIPS Instruction Set Architecture

MIPS is a reduced instruction set computer (RISC) meaning that it has fairly simple instructions based on the assumption that it is faster to perform simple things this way and complex operations can be constructed from the smallest logical parts. Code density, and therefore cache hit rates, are worse with RISC computers due to more instructions being needed to describe an operation. And typically instructions are fixed length (4 bytes) whereas CISC machines such as x86 have variable length instructions. However, RISC machines tend to be easier to implement and can be made faster. In fact, modern CISC processors often convert CISC instructions into micro-ops to be executed by a RISC-like execution core. As with most RISC machines, there are many registers, 32 integer registers, available to the programmer. One key feature is that there is a hardcoded zero register which is useful in many operations such as logical comparisons or clearing a value. MIPS uses big endian memory addressing, meaning that the most significant byte (MSB) is stored at the memory address. The prescribed function of each register is briefly described in **Table 1**, followed by a description of the subset of the ISA implemented by my out of order processor (as well as the 5-stage and superscalar CPU’s) in **Table 2**, **Table 3**, and **Table 4**.

Table 1 MIPS Register Usage

Name	Number	Register Usage
`$zero`	`$0`	Hardcoded Zero
`$at`	`$1`	Assembler Temporary
`$v0-$v1`	`$2-$3`	Return Values
`$a0-$a3`	`$4-$7`	Argument Registers (More Arguments Passed on Stack)
`$t0-$t7`	`$8-$15`	Temporary Registers (Not Saved Across Function Calls)
`$s0-$s7`	`$16-$23`	Saved Registers (Saved Across Function Calls)
`$t8-$t9`	`$24-$25`	Temporary Registers (Not Saved Across Function Calls)
`$k0-$k1`	`$26-$27`	OS Kernel Registers
`$gp`	`$28`	Global Pointer
`$sp`	`$29`	Stack Pointer
`$fp`	`$30`	Frame Pointer
`$ra`	`$31`	Return Address

Table 2 MIPS ALU Instructions

Instruction	Operation	Notes
`ADD, ADDI`	`rd <- rs + rt`	`Generate Overflow Exception`
	`rt <- rs + sext_imm`	`Not Implemented, No O/S`
`ADDU, ADDIU`	`rd <- rs + rt`	`No Overflow Generated`
	`rt <- rs + sext_imm`	
`SUB, SUBU`	`rd <- rs - rt`	`Overflow Not Implemented`
`SLT, SLTI`	`rd <- (rs < rt)`	`Set Less Than`
	`rt <- (rs < sext_imm)`	`Result 1 True, 0 False`
`SLTU, SLTIU`	`rd <- (0||rs < 0||rt)`	`Set Less Than Unsigned`
	`rt <- (0||rs < 0||sext_imm)`	`Result 1 True, 0 False`
`AND, ANDI`	`rd <- rs & rt`	`Immediate Zero Extended`
	`rt <- rs & zext_imm`	
`OR, ORI`	`rd <- rs | rt`	`Immediate Zero Extended`
	`rt <- rs | zext_imm`	
`LUI`	`rt <- imm << 16`	`Load Upper Immediate`
`XOR, XORI`	`rd <- rs xor rt`	`Immediate Zero Extended`
	`rt <- rs xor zext_imm`	
`NOR`	`rd <- ~(rs | rt)`	
`SLL, SLLV`	`rd <- rt << sa`	`Shift Left Logical`
	`rd <- rt << rs`	
`SRL, SRLV`	`rd <- rt >> sa`	`Shift Right Logical`
	`rd <- rt >> rs`	`Zero Extend Shift`
`SRA, SRAV`	`rd <- rt >>> sa`	`Shift Right Arithmetic`
	`rd <- rt >>> rs`	`Sign Extend Shift`
`MULT, MULTU`	`HI,LO <- rs * rt`	`Upper 32 Bits of Product in`
	`HI,LO <- 0||rs * 0||rt`	`Hi, Lower 32 Bits in LO`
`DIV, DIVU`	`HI <- rs / rt`	`HI Gets Division Result`
	`LO <- rs % rt`	`LO Gets Remainder`
	`HI <- 0||rs / 0||rt`	
	`LO <- 0||rs % 0||rt`	
`MFHI, MFLO`	`rd <- HI`	`Move From HI`
	`rd <- LO`	`Move From LO`

Table 3 MIPS Memory Instructions

Instruction	Operation	Notes
LW	rt <- mem[base+offset]	Load Word

LH, LHU	rt <- sext(mem[base+offset]&FFFF)
	rt <- zext(mem[base+offset]&FFFF)
		Load Half Word

LB, LBU	rt <- sext(mem[base+offset]&FF)
	rt <- zext(mem[base+offset]&FF)
		Load Byte

SW	mem[base+offset] <- rt	Store Word
SH	mem[base+offset] <- rt[15:0]	Store Half Word
SB	mem[base+offset] <- rt[7:0]	Store Byte

Table 4 MIPS Control Flow Instructions

Instruction	Operation	Notes
J	PC <- PC+4[31:28]||instr_index||00	Jump to Address, Has Delay Slot
JAL	$ra <- PC+8 PC <- PC+4[31:28]||instr_index||00	Jump to Address and Link, Has Delay Slot
JR	PC <- rs	Jump to Address in Register, Has Delay Slot

JALR	rd <- PC+8
	PC <- rs
		Jump to Address in Register and Link, Has Delay Slot

BNE	if(rs != rt) PC <- PC + 4 + offset	Branch Not Equal, Has Delay Slot
BEQ	if(rs == rt) PC <- PC + 4 + offset	Branch Equal, Has Delay Slot
BLTZ	if(rs < 0) PC <- PC + 4 + offset	Branch Less Than Zero, Has Delay Slot
BGEZ	if(rs >= 0) PC <- PC + 4 + offset	Branch Greater Than or Equal to Zero, Has Delay Slot
BLEZ	if(rs <= 0) PC <- PC + 4 + offset	Branch Less Than or Equal to Zero, Has Delay Slot
BGTZ	if(rs > 0) PC <- PC + 4 + offset	Branch Greater Than Zero, Has Delay Slot

BGEZAL	if(rs >= 0) PC <- PC + 4 + offset
	ra <- PC + 8
		Branch Greater Than or Equal to Zero and Link, Has Delay Slot

BLTZAL	if(rs > 0) PC <- PC + 4 + offset
	ra <- PC + 8
		Branch Less Than Zero and Link, Has Delay Slot

SYSCALL	Perform System Call Exception	Emulated in C Library, Updates Register File, Flushes Pipeline to Emulate Exception Vector

II. Design Details

The high-level data-path for the CPU is shown in **Figure 1**. Due to the extremely large number of busses passing between modules, the exact names and widths are omitted to more clearly show the functionality. Individually, each component is mostly straightforward. However, the timing of interactions between each part and the many corner cases (several unfortunately discovered during debugging) that arise yield significant complexity over the previously described CPU's. Module interactions are described in detail in their respective sections.

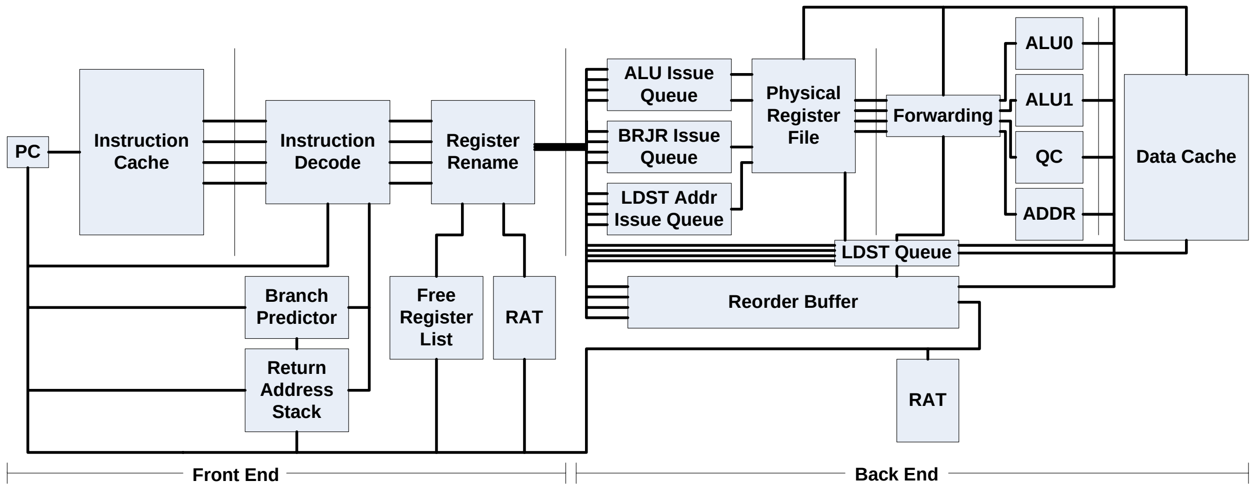

Figure 1 High Level Datapath

II.A. Front End

The front end of the CPU is responsible for fetching instructions, decoding these instructions, and renaming them such that the backend can issue them as dependencies allow. Speculative control flow is done in the front end to provide a constant stream of instructions. Upon resolving a speculated flow direction, the reorder buffer (ROB) can direct the front end to stop what it is doing and redirect its fetching down the correct path. The front end of the processor will have to stall if the back end runs out of space to buffer instructions, the free list runs out of registers to provide to the renamer, or the instruction cache misses.

II.A.1. Program Counter (PC) / Instruction Cache (I$)

The program counter is still used to index into the instruction memory as before and maintain the current state of the processor. However, the complexity of updating the PC has increased. The instruction decoder provides the control as to how to get the next PC. It can either stay the same on an instruction cache miss, be calculated based off of the type of instructions in the decoder, or get updated from the ROB on a flush. If the decoder finds that a control flow instruction is being decoded, it may update the PC with a jump target, a branch target or the fall through target depending on the branch predictor's prediction, or a return address stack prediction if a jump register is decoded. Otherwise, the PC will be updated with the address of the instruction following the last one decoded. If the processor has to stall because the renamer runs out of free registers or the ROB is full, the design freezes the front end. However, the PC must still be updated based of the instructions in the decoder and that target may change based on prediction updates coming from retired branches in the ROB. This condition was discovered experimentally when a branch predicted taken, updated the PC with the target, and stalled.

When the stall was resolved, the prediction had changed to not taken and that prediction was carried with the branch into the ROB, even though it had set the PC to the taken path.

The instruction cache interface remains similar to the previous implementations. Changes were made to support fetching four instructions per cycle. Again, no requirement is made that the instructions reside in the same cache line. With four addresses being accessed, the chance of a miss increases. Despite fetching four instructions, we can only have at most two misses because of the size of the cache line used. Both misses must be handled before fetching can continue. Because data memory accesses and instruction memory accesses no longer stall each other, it is possible to have an instruction cache miss come part way through a data cache miss. In this case, the instruction cache miss is deferred until the data cache miss has been resolved. The design uses a blocking cache interface that can only handle one miss of any type. When the processor decides to flush because of a misprediction during an instruction cache miss, the flush has to be deferred until the miss is resolved. Although this miss is clearly from the wrong path, there is still a high probability that the code will be needed soon due to spatial locality. Small direct mapped caches are used to re-align the comparison with the previous processors and maintain the assumption of single cycle accesses. The assumption that two cache lines can be accessed simultaneously can be easily supported by banking the caches, such that the addresses which resolved to different lines also access different banks.

II.A.2. Instruction Decode

As its name suggests, the instruction decoder is responsible for interpreting the 4 byte instruction word and extracting useful information for the instruction. In hardware, it is implemented as a large lookup table. Four lookup tables proceed in parallel to set the proper control signals for each instruction. Information extracted from decoding an instruction includes which registers it uses, what type of instruction it is, which issue queue to place it in, what operand the corresponding execution unit should perform, and calculating targets and immediates. Once the four instructions have been decoded, they proceed to a mini-scheduling piece of logic that determines which instructions can go to rename of those fetched. This mini-state machine also makes the decision of where to fetch from next. Instructions that need special consideration are discussed in the next two sections.

II.A.2.a. Multiply / Divide Expansion

Only multiplies will be discussed for brevity, though the discussion applies to both multiplies and divides. In the previous two CPU’s, there existed two dedicated registers for storing the results of multiplies. Also, subsequent multiplies could stop short and overwrite currently executing multiplies. However, now that instructions are being executed out of order, we need a mechanism for providing consumers with the proper multiply. This is handled by renaming the HI and LO registers as if they were part of the normal 32 registers in the ISA. This creates the issue of needing two registers for one instruction. The solution is to expand each multiply into two instructions where one writes the HI register and the other writes the LO register. Because of the limitation of renaming four instructions per cycle, the expansion of a multiply will force another instruction out of the current block going to rename and that a multiply cannot be decoded as the fourth instruction of the block. In either case, the dropped instruction (or more if two multiplies are expanded) will be fetched again in the next cycle. While a multiply now takes up more buffer and queue space, instructions only wanting to read the result of HI or LO can execute once that half of the multiply has completed.

II.A.2.b. Control Flow Reordering

This discussion applies to both branch and jump delay slots, but the term branch delay slot will be used. While the branch delay slot was useful in boosting performance of a pipelined processor, it now becomes a nuisance. There is no longer a need for it as we can predict the next instruction and execute instructions out of order. Therefore, the last part of the decode logic attempts to reverse the delay slot and have it execute ahead of the branch. This is legal because the ISA specifies that logically the branch delay slot be executed before the effects of the branch are seen. To minimize wasted ROB space and the number of ports needed to write link values, only one control flow instruction will be renamed each cycle. Any subsequent instructions wait until the next cycle if they still remain in the predicted path.

Swapping the delay slot and the branch simplifies flushing on a misprediction because the delay slot will already have been committed when the branch reaches the ROB head. This requires that the delay slot be fetched in the same block as the branch and so a branch cannot be the last instruction in the block. However, this is a useful restriction even without reordering the instructions because otherwise it would be possible for the branch to reach the ROB head while the delay slot was still waiting on an instruction cache miss to be resolved, further complicating flushing.

Due to compiler optimizations such as code boosting to fill the delay slot, we occasionally have the situation where the delay slot instruction’s result will change the branch direction if allowed to be renamed first as this will cause the branch to use its value that should have happened after the branch evaluated. Therefore, we always rename the branch first, although the delay slot is inserted ahead of it into the ROB. In the case where there is a data dependency, another problem arrives. If the delay slot reaches the ROB head before branch evaluates, it will mark the branch’s operands as unready, for reasons to be discussed in the commit description, and the branch will wait at the ROB head forever. It must wait for the branch to be resolved before the delay slot can commit when there is a dependency.

II.A.3. Branch Prediction

The performance of a processor that executes instructions speculatively is only as good as its ability to predict the correct path to take. To this end, one static and three dynamic branch predictors were implemented. Finally, a pseudo fifth predictor was implemented that simply takes a vote among the three dynamic predictors. The static predictor always predicts backward branching branches to be taken under the assumption that they are loops and loops are usually ‘taken’. Forward branching branches are predicted ‘not taken’ because they are typically if statements, and there seems to be a slight tendency for them to be ‘not taken’ (ie. execute the “if” not the “else”). This static prediction is based solely on the sign of the offset.

Figure 2 shows the dynamic branch predictors. Predictor ‘a’ is a bimodal predictor. Bit 11:2 of the program counter of the branch are used to index a table of two bit saturating counters with 1024 entries (where the most significant bit of the counter corresponds to the prediction). A count of 3 or 2 indicates ‘taken’, where 1 or 0 indicates ‘not taken’. The bimodal predictor is rather simple to in concept and implementation but has the lowest prediction accuracy of the three. This is because it only considers the past history of the branch (or other branches that map to the same entry) and makes the assumption that past behavior will dictate future behavior. While in many cases this is correct, it has many serious flaws. For example, a branch that toggles between 'taken' and 'not taken' can force this predictor to miss 100% of the time if it is in a weakly taken (2) or not taken (1) state.

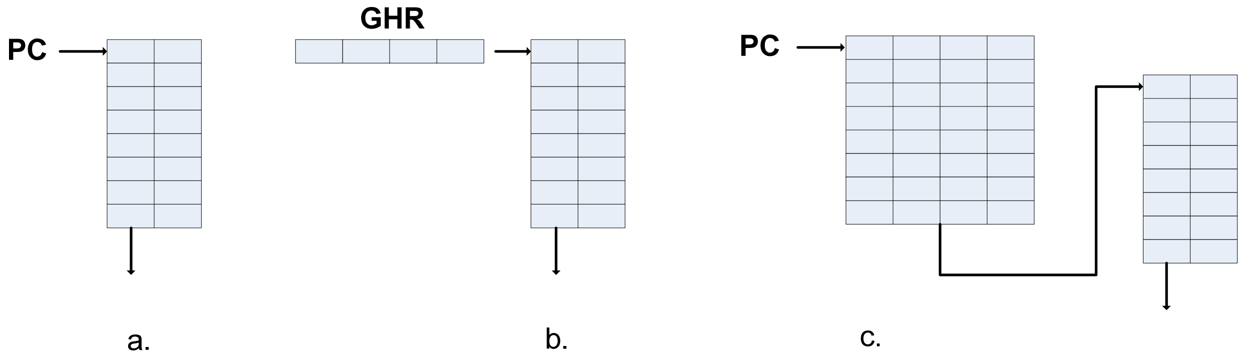

Figure 2 Branch Predictors

Predictor 'b' is a GAg predictor. A 10 bit global history register (GHR) is used to a table of the same size with two bit saturating counters. The GHR is a shift register that is updated by every branch. The assumption here is that branches correlate and the direction of the current branch can be determined by the previous branches. Consider the following code segment:

```
if(a){
…
b = 1;
}
else b = 0;
…
if(b){
…
```

Clearly, the direction taken by the second "if" statement is controlled by that of the first. This is where a correlating branch predictor will perform better than one that only considers its own past history.

Predictor 'c' is a two level SAg predictor that exploits self-correlation. While it has the highest accuracy, it is the slowest and consumes the most space. The lower bits of the PC are used to index a table of 1024 entries, each of which is a 10 bit shift register. The value of the shift register is used to index a table to 1024 2-bit saturating counters.

All branch predictors are updated upon retiring a branch so that only correct path branches influence the predictor accuracy. This introduces a possibly large update latency but also prevents mispath branches from influencing the prediction of other branches. All these predictors are also vulnerable to aliasing so that other branches that happen to map to the same entry will perturb the other predictions.

II.A.4. Return Address Stack (RAS)

The return address stack is used for predicting jump register instructions. The assumption is that most jump register instructions use the return address register and most of the time the return register was previously set by a link instruction. Therefore, each link value is pushed onto the stack and popped off when a jump register instruction is decoded. Typically function calls are made with a jump and link instruction and returned from with a jump register instruction. This is the type of sequence where the return address stack is most useful. To try to maintain accuracy in the event of deep recursion, a counter is kept to track the number of missed link values and the corresponding number of jump register instructions do not pop a value from the stack. Also, since mispredicted path instructions will push and pop the stack before we realize they were mispredicted, the reorder buffer tracks these at commit and informs the return address stack whether its stack pointer is too high or too low. While this doesn't solve all of the mispredictions, it does help improve accuracy enough to warrant implementation.

II.A.5. Register Rename

Register renaming resolves the WAW and WAR hazards introduced by executing instructions out of order. Each logical register is mapped to a physical register that is currently holding the value for that logical register as stored in the register allocation table (RAT). Renaming works because we have twice as many physical registers as logical registers (64 physical registers). Each physical register is written to only once while it is live and the backend of the CPU works solely with physical register until commit where the logical register is needed again. Therefore there are no WAW or WAR hazards. Consumers do not read the register until it has been written to, controlled by the ready register list discussed later. Every instruction is given a destination register from the free list, provided one is available, and the RAT is updated with the new mapping. Because instructions in the same fetch block can have any number of dependencies, renaming happens in the order instructions were fetched. Because this RAT is working with speculative values, it must be updated on a flush with the actual correct mappings as stored in the retirement RAT. At reset, both RAT's are mapped so that each logical register corresponds to the physical register with the same number. An example renaming is as follows:

```
Free List: p10, p32, p27, p9
RAT Mappings: r1 : p3 , r2: p1, r3: p33, r4: p23
Instructions before Rename (Note the destination is the first register):
      add r5, r3, r1
      sub r6, r2, r4
      add r5, r5, r6
Instructions after Rename
      add p10, p33, p3
      sub p32, p1,  p23
      add p27, p10, p32
```

II.A.6. Free Register List

The free register list is a ring buffer that gets freed registers from the reorder buffer and gives them to the renamer. Because there are 64 physical registers in this implementation and 34 logical registers (32 + HI + LO), there are a maximum of 30 free registers. If a large enough number of instructions write values and are given a register from the free list, the free list can run out requiring the processor to stall until enough free registers have been recovered for renaming to continue.

II.B. Back End

The back end of the CPU is responsible for taking the renamed instructions and issuing them to be executed as their operands become available. The only dependency that remains is the RAW dependency which is ultimately the limiting factor in how fast we can execute a program. Other independent instructions are issued to execute while we wait on other dependencies. The sequential ordering of the program is maintained through the reorder buffer which commits instructions in order as they complete execution. The major components of the back end are the issue and load-store queues, the reorder buffer, the four execution units (2 ALUs, 1 Address Calculation Unit, 1 Branch/Jump Register Resolver), forwarding muxes, the physical register file, the retirement RAT, and the data cache.

II.B.1. Issue Queues

There are three issue queues in the processor: one for the two ALU's and one for each of the other execution units. The ALU queue can issue two instructions to be executed every cycle whereas the other queues can only issue one. Each queue snoops on the bus of instructions coming out of rename to see if their queue matches. If so, the instruction is inserted into the issue queue at the first available entry. NOPS and syscalls do not require any execution unit, and so they are placed only into the ROB and not an issue queue. The decision of which instruction to issue next is made by searching the queue for the first instruction (or two for the ALU's) that has all of its operands ready. It is then issued to read from the register file.

After instructions are issued, there is now a hole in the issue queue that the issued instruction previously occupied. Because we both insert and issue from the front of the queue, a newer instruction may get inserted ahead of an older one and force it to wait, thereby also delaying all dependant instructions. This will shortly be resolved because the instruction will eventually stall at the ROB head and be the only instruction left in the queue that can be issued. However, to give priority to older instructions, the issue queues are compacted after issuing an instruction. Starting at the front of the queue, each instruction moves up a space if the entry in front of it is no longer valid. For the ALU queue, this means that all possible gaps may not be filled because two instructions can issue at once. In most cases, performance is improved greatly by giving priority to older instructions. In the rare event that an issue queue fills, the processor would have to stall. Because this stall would have identical results to an ROB full stall, the issue queues are instead sized such that they will never fill completely.

II.B.2. Physical Register File

The physical register file works the same as the register file in the other processors except that it contains double the number of registers and has more ports to support all of the execution units. Two read ports and one write port is dedicated to each ALU. The branch/jump register execution unit also has two read ports but does not need a write port. One write port is dedicated to storing link values. One read port is dedicated to the address calculation unit and another to producing values for stores to write. The final write port is used for load data. This totals to eight read ports and four write ports. Writes are performed on the positive clock edge if an enable signal is set for the corresponding write port. Reads are performed after instructions are issued on the negative clock edge. As mentioned previously, there are 64 physical registers in the register file, each 32 bits wide. As with the logical register file, one physical register is hardcoded to zero for instructions that compare with zero or to cause an instruction to be a NOP by setting its destination to the zero register.

II.B.3. Ready Register List

To control the execution of instructions, we need to know which have their operands ready. This is maintained through the ready register list, really a 64 bit bit-vector of ready registers. After ALU operations are issued, the ready bit of their destination register is set. This allows dependant instructions to issue in the following cycle, maintaining the back to back dependent instruction execution of the other CPU's. Link values are marked as ready as they enter the ROB. Load values are marked as ready as they are performed, either through data forwarding or cache accesses. As instructions retire, the physical register previously mapped to same destination as the instruction retiring is now marked as unready and returned to the free list because we are guaranteed no other instruction will need its value as it will be mapped to the currently retiring instruction's register. Flushed instructions return their register to the free list and mark their destination as unready instead of the previous writer. Because an instruction may be marked as ready and then flushed in the same cycle and marked as unready, clearing the ready bit masks out setting the ready bit. The hardcoded zero is always marked as ready even if an instruction tries to clear it by writing to the zero register as a NOP.

II.B.4. Execution Units

Each execution unit performs independently of the others because the issue logic guarantees there are no RAW hazards between currently executing instructions. While it is possible to send all instructions through the general purpose ALU's with a few additions, the importance of resolving control flow and memory accesses as quickly as possible results in them each having their own execution unit and issue queue. The number of execution units dedicated to each type of instruction roughly corresponds to the average occurrence of the instruction type in a typical program.

II.B.4.a. Arithmetic Logic (ALU) Units

The two general purpose ALU's perform all of the instructions indicated in **Table 2**. They are essentially the same ALU's used in the previous processors with the change that multiplies, divides, and MFHI/MFLO are now performed in the ALU instead of in separate logic. The ALU performs every operation every cycle and selects the result for the operation indicated by the instruction currently executing. Which register to use and whether or not to use the immediate value passed in were all determined previously in decode. The operands for ALU instructions can come from either the register file or from forwarded values from the execution performed in the previous cycle. The ALU result is written back to the register file on the next positive edge. There are two ALU's because the majority of instructions are processed by one of the ALU's, and there is usually sufficient parallelism to issue two independent instructions every cycle. A third ALU is not added because control flow and memory access instructions are already separated out, and a third ALU would add two more read and one more write ports to the register file.

II.B.4.b. Branch / Jump Register Resolution Unit

The branch comparison unit, also referred to as the quick compare (QC) unit, tests expression conditions between the two source operands of a branch or one operand and zero. The decision to take a branch or not is written back to the reorder buffer entry corresponding to the branch. When the branch gets to the ROB head and the predicted and actual decisions do not match, we have to flush the pipeline and fetch the correct target. Jump register instructions also pass through this unit but no computation is performed. It is rather a convenient reuse of register file ports that might otherwise sit idle. Jump register instructions are not frequent enough to warrant their own ports in the already heavily loaded register file.

II.B.4.c. Load-Store Address Resolution Unit

Because memory accesses can incur a very high latency if they miss, it is important to resolve their effective address as soon as possible. This is simply an addition of a base register and a signed offset contained in the instruction. Once this value has been calculated, it is written to the load-store queue. This allows stores to forward data and loads to issue before they reach the ROB head as will be discussed shortly.

II.B.5. Data Forwarding

To allow dependent instructions to issue back to back, it is necessary to forward the data values currently being written to the register file to the execution units because the values were not present when the instruction read the register file. The notion of getting the newest value is no longer a problem because the renamer already handled this by guaranteeing that each register is only written once during its lifetime. Each of the eight read ports on the register file has a corresponding forwarding unit. Each forwarding unit compares the register that was intended to be read with each of the four write-back paths' destination register. If the register number matches, the value being currently written back is used instead of the one fetched from the register file. Because write-backs happen a cycle earlier than in the pipelined or superscalar processors, there is only one cycle worth of addresses to compare against but there are now four sources of forwarding.

II.B.6. Load-Store Queue / Data Cache (D$)

The load store queue was not initially implemented in the processor. Loads were performed when they reached the ROB head like stores are. However, performance suffered tremendously by waiting that long to perform the load. There are typically many instructions dependant on the load result --- the longer the load waits, the longer its consumers must wait. Furthermore, if the load gets to the ROB head and then misses, the processor must wait even longer to execute the dependant instructions.

This lead to the load-store queue being implemented. Unlike registers, memory dependencies cannot be resolved in rename because their effective addresses are not yet known. It is therefore essential that the load store queue ordering maintain sequential consistency semantics. As loads and stores enter the back end from rename, they are added to the end of the load store queue in program order. Each cycle, queue entries snoop the result of the address calculation unit to see if their corresponding entry in the address queue has just calculated the effective address. If so, the address is added to the queue entry. Also each cycle, we search the queue for a store that does not yet have the data that it will write to memory. Because there is only one register read port for store values, only one store can request its data per cycle. This store is always the one closest to the head of the queue (the oldest store). Stores in the queue that have both their address and data can search the rest of the queue for a load of the same size to the same address. If a match is found, the data is forwarded to that load to save a cache access. This search is stopped when a store to the same address or a store with an unresolved address is found. A conservative forwarding policy of stopping the search when a store to the same word is found was implemented. A more aggressive policy would do per byte aliasing checks; however, the assumption that words are typically accessed with word accesses and bytes accessed with byte accesses is made. Significant speedups can be achieved even with this conservative forwarding policy. Stores must still wait to write to the cache until they reach the head of the reorder buffer to guarantee that they are not speculative because there is no easy way of undoing a store that shouldn't have happened.

Once a load has its address, it can issue a cache access or receive forwarded data. Priority of cache accesses is given to the oldest loads. If no load or store cache access is performed because the operation reached the ROB head without going to memory, then another load that has its address but doesn't have its data yet can use the cache for its access provided there isn't an outstanding miss. Forwarded data also needs to be written back to the register file. Waiting until the load reaches the ROB head is as bad as not forwarding at all. Therefore, the queue is searched for either a load with its data but not written back or a load with its address but no data. The oldest one found is issued. Even while a store is using the data cache, we can write back a load that had its data forwarded. In essence, this performs two memory operations simultaneously despite having only a single cache port.

The data cache is essentially the same as in the previous processors. Dirty lines must be written back on an eviction before the request line can be filled. While the load store queue speeds up loads significantly, stores may still suffer long latency misses at the ROB head without prefetching, which is not implemented. A data cache miss can be nearly twice as bad as an instruction cache miss because of the possibility of having to wait to write back dirty data to the next level of memory.

II.B.7. Reorder Buffer (ROB)

The reorder buffer is a 32 entry circular buffer responsible for maintaining instructions in program order despite being executed out of order. Relevant information about each instruction coming out of rename is inserted into the ROB at the tail. Instructions are retired in program order when they reach the ROB head, or rather the ROB head reaches them. If the ROB fills up, the processor must stall until entries are freed. Entries are marked as ready as their operations complete elsewhere in the processor. Only instructions that are marked as either ready or flushed can be retired or committed. This implementation has the capability of retiring four instructions per cycle to match the fetch width and number of execution units.

II.B.8. Commit / Pipeline Flushing

Retiring or committing instructions in order is critical for proper program execution. When an instruction reaches the ROB head, it is checked to see if it is marked as either ready or flushed. If it is marked as neither, retirement must stall until the instruction can be retired.   When an instruction that writes to a register reaches the ROB head is ready and is not flushed, we must reclaim the physical register of the previous instruction that wrote to the same logical register as it is only at this point that we can guarantee that it will no longer be needed. The now free register is sent to the free list, marked as unready, and the retirement RAT is updated to show the new mapping. The retirement RAT contains the mappings of logical to physical registers of all instructions so far and is guaranteed not to be speculative. When a branch or jump register reaches the ROB head, we check to see if the predicted path was indeed the correct one. If so, the instruction is retired as normal. If not, the ROB, issue and load-store queues, and currently decoding instructions are flushed, the rename RAT is replaced with the correct information from the retirement RAT, and the PC is set to the correct target. As mentioned previously, if a delay slot reaches the ROB head that overwrites an operand of its control flow instruction, it must wait until both instructions are ready before committing.

Memory operations that have yet to access the cache or have gotten forwarded data but have yet to write it to the register file are performed when they reach the ROB head. Because of resource limitations, only one such instruction can be performed and retired each cycle. System calls that reach the ROB head stall there for one cycle to allow any writers to the logical register file (as determined by the retirement RAT) to complete before emulating the system call with C libraries. The ROB is the flushed following this emulation because the values in the register file previously read are now stale and this more accurately simulates jumping to O/S code and returning to the user code following the system call. Finally, registers allocated to flushed instructions must be reclaimed and added to the free list, and marked as unready.

III. Differences with MIPS R10000

While this processor design is based generally off the MIPS R10000, there are some design differences that should be noted. The R10000 implements the full 64-bit MIPS 4 ISA including floating point instructions which are absent from this implementation. While the fetch width is the same, it issues instructions to five execution units (two of which are floating point) and implements non-blocking, set associative caches. Because more than one process runs on the CPU at a time, it implements TLB's to cache the page table. To buffer against instruction cache misses and to save instructions not able to be decoded in a given cycle, there is a fetch buffer to store up to eight instructions. Mispredicted branches are allowed to selectively flush the pipeline and queues as soon as they are determined to be mispredicted. This is done by tagging each instruction with which branch of up to 4 speculated branches that the instruction depends on. The rename RAT is placed on a branch stack as branches are predicted so that it can be easily restored upon a flush. The branch predictor used is the bimodal predictor described previously. Branches are resolved in the integer ALU whereas a separate comparison unit is used in my implementation.

IV. Results

Performance results for the processor were collected to compare with the other processors and different configurations of the out of order processor. Unless the test indicates that an item is varying, the implementation discussed above is used. Both instruction and data caches are 16KB direct mapped caches with 32 byte blocks. The SAg predictor is used in all cases unless otherwise stated.

IV.A. Test Cases

The test cases presented here are a subset of those used to verify functionality. Many of those tests tested very specific instructions and were too short for performance to be reasonable. This is because short programs have a very large number of cold cache misses, and there is not sufficient time to train branch predictors. All test cases presented here were written in C and compile with a gcc cross compiler on the Linuxpool. Several other assembly language programs were written to test functionality as well but they are not shown here. **Table 5** describes the mini-benchmarks used.

Table 5 Performance Benchmarks

array_rewrite	On even loop iterations the first array is copied to the second array, On odd iterations the array is copied back from the second array
bigfib	Calculates the Fibonacci sequence to 6765
hello	Prints Hello World and then performs a tight nested loop sequence
msort	Merge sort for an array of size 500
nested_branches	A series of correlated branches in nested loops
nqueen	Nqueens algorithm for board of size 7
recursion_loops	Repeated recursive calls that have variable loop lengths
towers	Towers of Hanoi for 6 disks

IV.B. Comparison with Five Stage Pipeline & Two-Way Superscalar

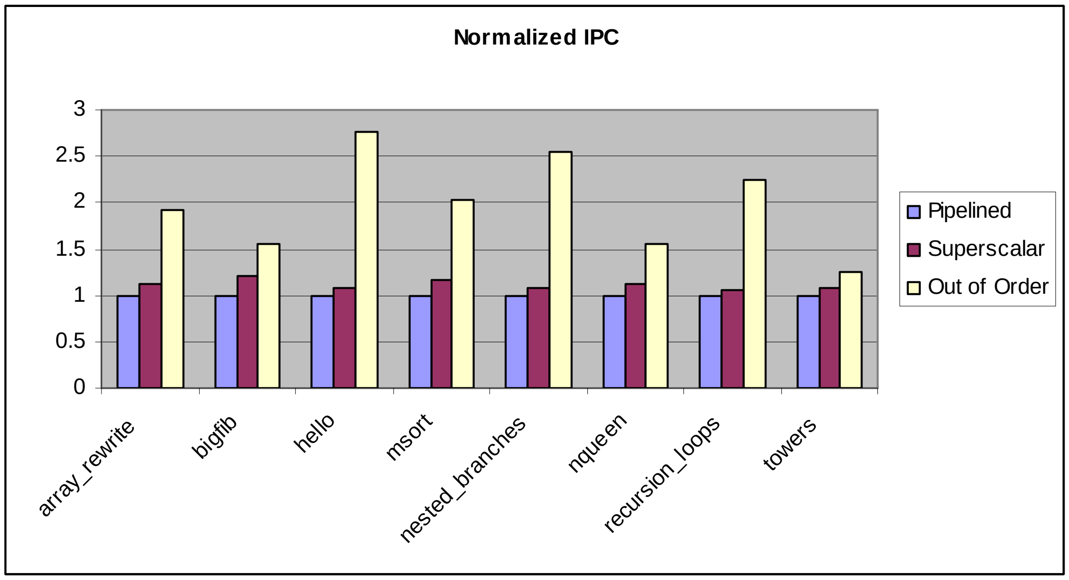

Figure 3 Three CPU Performance Comparison

As the above figure indicates, the out of order processor outperforms the other two processors for all benchmarks tested. This is due largely in part to the predictability of the branches in these benchmarks. Though not described in this graph, “towers” executes less than one instruction per cycle for all implementations. This is because there are an extremely large number of instruction cache misses and the program does not run for long enough to recover from the performance loss of the stalls. Normalized IPC is used in these performance tests because the clock speed can be set somewhat arbitrarily in the simulator and a better measure of relative performance is how much work is done each cycle.

IV.C. Reorder Buffer Size

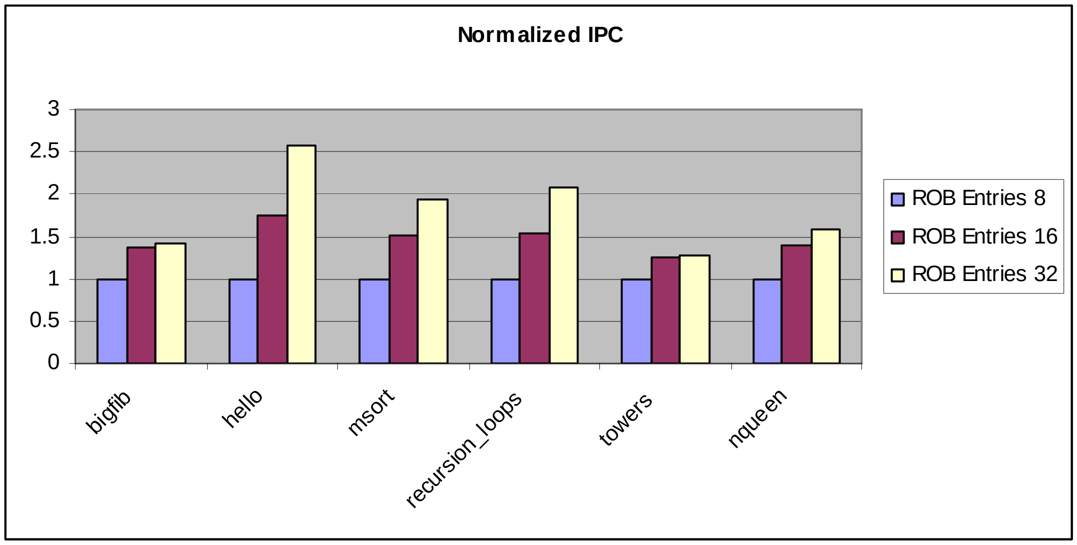

Figure 4 Reorder Buffer Size Performance Comparison

Surprisingly, cutting down the ROB size did not affect some benchmarks very much. Again using "towers" as an example, this is because the instruction cache could not supply enough instructions to fill the ROB anyway. "Nqueen" and "bigfib" also show little improvement going from 16 to 32 entries which would indicate that they empty the ROB fast enough to not stall often due to the ROB being full. This could be due to a favorable instruction stream executing on the processor or that the instruction memory again can't supply enough instructions to keep the back end busy. A third explanation may be that many branches are mispredicted and the ROB does not have a chance to fill before getting flushed, which may be the case for "bigfib," which has relatively bad predictor accuracy as shown next. The most likely explanation is a combination of not being able to fill the ROB due to instruction cache misses and flushes.

IV.D. Branch Predictors

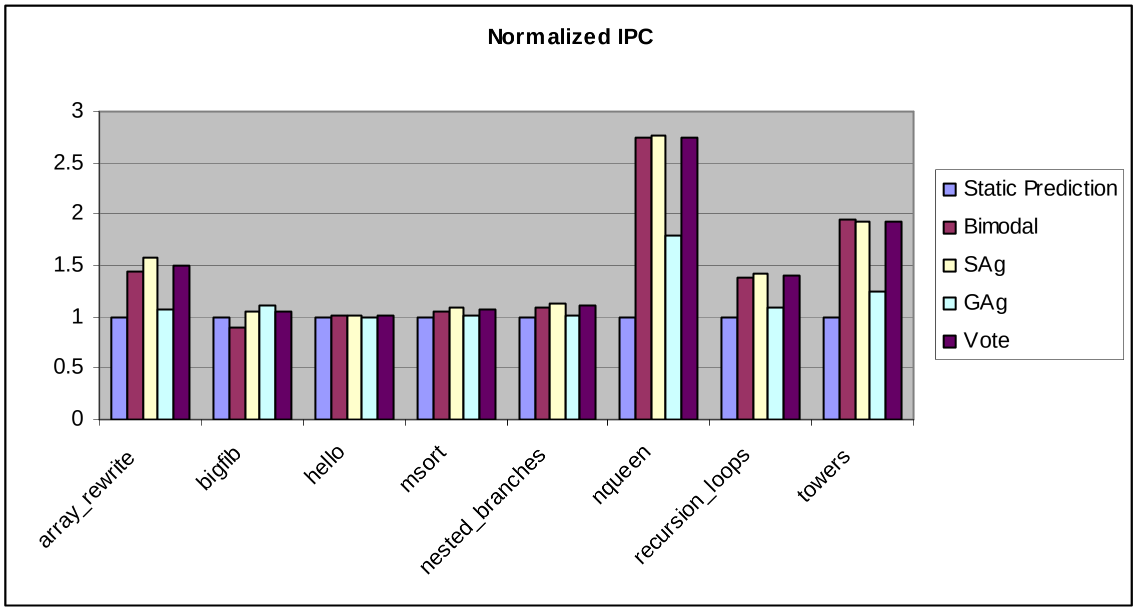

Figure 5 Branch Predictor Performance Comparison

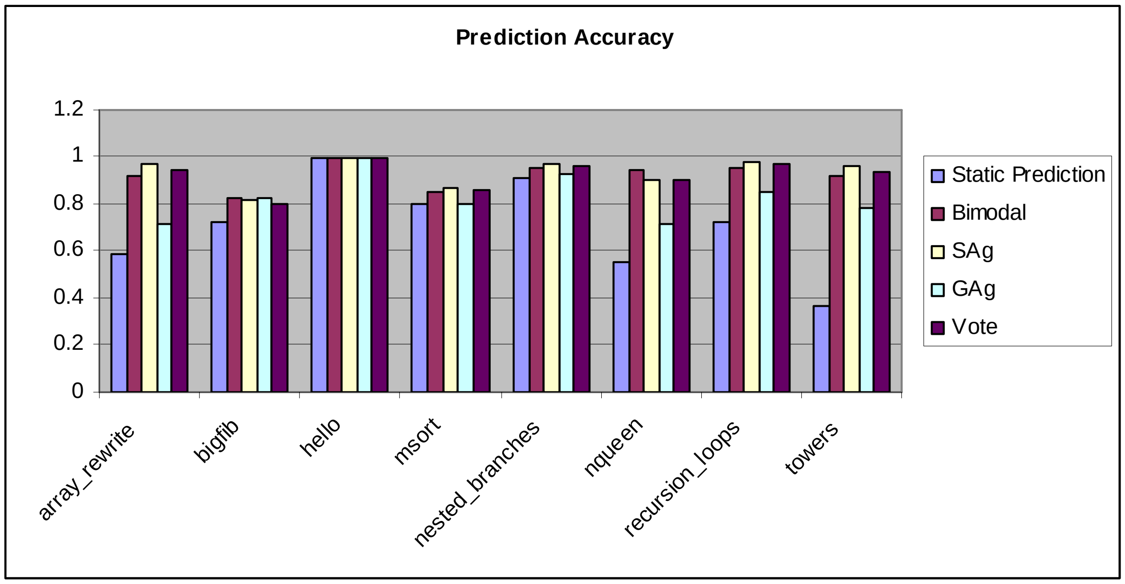

Figure 6 Branch Predictor Accuracy

As can be seen from the above graphs, using a static prediction scheme tends to perform pretty poorly. Not surprisingly, the majority vote of the three dynamic predictors is near the top but never really the best. As discussed previously, different branch predictors predict better for different branch behaviors and so taking a vote smoothes out the bad predictions but can also suppresses the predictor that best predicts a given branch. Also of concern in branch accuracy is that only correct path branches are considered in accuracy calculations and updating the predictors. Intuitively, it seems correct to suppress the results of a branch that you were never supposed to reach, but doing so may prevent warming up a predictor to a branch that you will soon reach.

IV.E. Retirement Rate

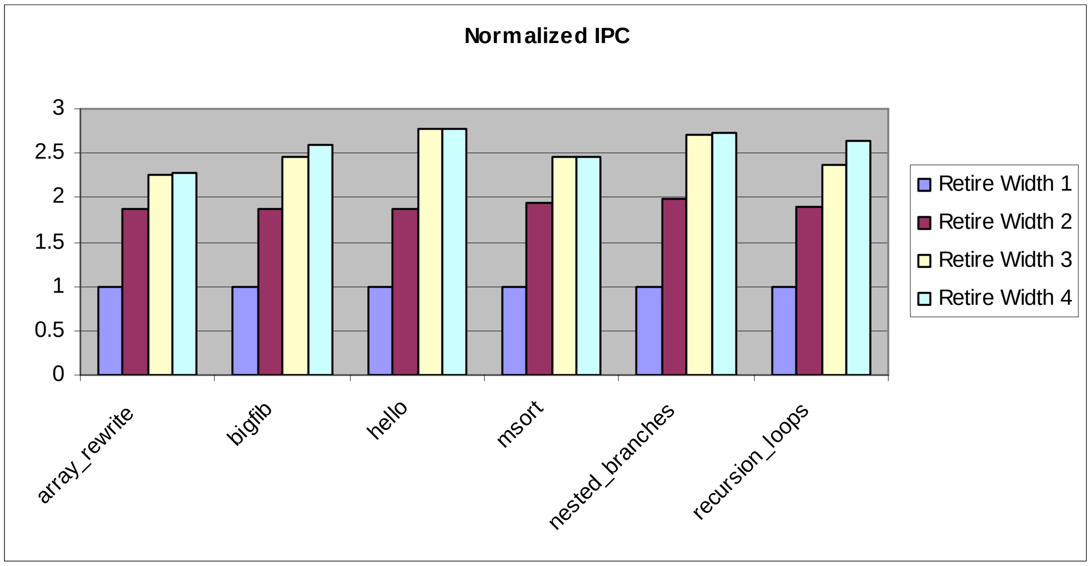

Figure 7 Retirement Width Performance Comparison

The final performance test performed was to see the effect of varying the retirement width. Retirement width one caps the maximum IPC at 1 and is slower than the superscalar processor in most cases; it can be slower than the pipelined processor due to instruction cache misses and the overhead involved in filling the ROB. As seen with varying the ROB size, often times the full capabilities of the ROB are not needed. It would take a near optimal flow of instructions to keep the entire busy at maximum performance for the length of the program. Cache limitations and predictor accuracy prevent this for all but the most trivial programs.

V. Conclusion

V.A. Design Difficulties

The debugging stage for this processor was much more tedious than expected. While each component has fairly predictable behavior in isolation, the interactions and timing between the modules makes it difficult to predict all possible corner cases of events that may occur. Also, the unexpected insertion of instructions that would affect a branch outcome by the compiler into the delay slot caused much grief. Because of the massive number of signals in the processor, it was nearly impossible to debug what was going wrong by looking at the waveforms only as was possible in the previous processors. I found it easier to generate traces of instructions retired and values written, and compare those to the pipelined implementation to find the point in time at which the out of order processor goofed and then follow the relevant signals in the simulator waveform. Such debugging would have been very difficult if not impossible if the implementation was attempted on only hardware from the start.

V.B. Possible Improvements

There are four main improvements that could be applied. Increasing the fetch width, execution width, or retire width would improve performance so they are not considered, nor is simply increasing buffer/queue sizes. The first useful improvement would be to use a set associative cache. This would greatly reduce all misses except cold misses which are unavoidable (but could be reduced with bigger block size). The second improvement that could be implemented is to implement a branch stack as the R10000 does. This would require much more flush logic to selectively flush instructions but has the benefit of reducing the amount of time spent on the wrong path which could pollute the cache. To this end, the third improvement would be to implement a better branch predictor. While some programs designed to work well with the branch predictors implemented worked very well, they also performed poorly on some more realistic applications. Finally, non-blocking caches would improve memory throughput greatly in the presence of a miss in both the instruction and data caches. This is especially useful when a flush is required but the instruction cache is already servicing a miss and so the redirection of the PC must wait.

V.C. Final Remarks

Overall, the out of order implementation satisfied the project goals quite well. I was able to fetch, decode, rename, issue, execute, and retire up to four instructions per cycle and achieved a much higher IPC than previous processors implemented in computer architecture classes. Last minute changes, such as the inclusion of a load store queue, significantly boosted performance.

VI. References

[1] Yeager, Kenneth C. “The MIPS R10000 Superscalar Microprocessor,” IEEE Micro 1996

VII. Appendix

The appendix is organized as follows. First, the simulator output and source code for the performance benchmarks shown above. Array_rewrite, hello, nested_branches, and recursion_loops were written for this project. The other four came from ECE475 projects. Next, the Verilog source code for the project is listed. The header files mips.h and cache.h were heavily modified from ECE475; the source code was written from scratch with the exception of mips.v and memory.v which were heavily modified from their ECE475 versions. ECE475 Verilog code bears the appropriate copyright notice.

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Loaded Executable `test/array_rewrite'
-----------------------------------------------------------------------
Boot code entry point: 0xbfc00000
User code entry point: 0x00400290
Stack pointer: 0x7fffefff
-----------------------------------------------------------------------

Testing Array Rewrite
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49

EXIT called, code: 0
-----------------------------------
Program Terminated.
Retired     172648 instructions,     195685 entered ROB
Number of flushes:       1007
Ran for                95731 cycles
ICache Accesses:      60509
ICache Misses:        430
DCache Accesses:      49185
DCache Misses:         55
Average Fullness:
ROB                  23.9
ALUQ                    0
CFQ                    0
LDSTAQ                    0
LDSTQ                   6.9
Freelist(avg. free regs)                    0
RAS                   2.0
Max Fullness:
ALUQ          6
CFQ          2
LDSTAQ          5
LDSTQ          20
RAS          8
BR Predictions:      10384
Incorrect:        330
JR Predictions:       1139
Incorrect:        571
Correct Path:
Mem      52832
CF      16649
Sys        106
ALU & NOPS     103061
Retired 0 1 2 3 4:      32162       8337      12979       7587      34657
Renamed 0 1 2 3 4:      36864          0      14210      11331      33318
Cycles Stalled:
IStall      11701
Rename        115
ROB      23390
Forwarded Data:       5841
Wrote Forwards Early:       5789
Performed Early Loads:      26786
Forwarded Data But No Early Write:          0
ROB Head Loads:       5892
```

```
Loads Already Written Before Retire:      30433
-----------------------------------
int q[50];

int main ()
{
  int i,j,t,z[50];
  printf("Testing Array Rewrite\n");


  for(i=0;i<50;i++){
        z[i] = 0;
        q[i] = i;
  }
  for(i=0;i<50;i++){
          for(j=0;j<50;j++){
                 if(z[j]){
                      t = z[j];
                      z[j] = q[j];
                      q[j]= t;
                 }else{
                      t = q[j];
                      q[j] = z[j];
                      z[j]= t;
                 }
              }
          printf("%d ",i);
  }

        printf("\n");
  exit(0);
}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Loaded Executable `test/bigfib'
-----------------------------------------------------------------------
Boot code entry point: 0xbfc00000
User code entry point: 0x00400420
Stack pointer: 0x7fffefff
-----------------------------------------------------------------------

Series:  1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
EXIT called, code: 99
-----------------------------------
Program Terminated.
Retired    3715759 instructions,    5729728 entered ROB
Number of flushes:      74062
Ran for              2388991 cycles
ICache Accesses:    1877710
ICache Misses:        321
DCache Accesses:    1019551
DCache Misses:         64
Average Fullness:
ROB                  26.1
ALUQ                    0
CFQ                   0
LDSTAQ                   0
LDSTQ                   9.2
Freelist(avg. free regs)                    0
RAS                  13.2
Max Fullness:
ALUQ          4
CFQ          2
LDSTAQ          3
LDSTQ         20
RAS         22
BR Predictions:     209709
Incorrect:      37495
JR Predictions:     115069
Incorrect:      36523
```

```
Correct Path:
Mem    1418908
CF     554721
Sys         44
ALU & NOPS    1742086
Retired 0 1 2 3 4:     348541     520711     294758     280403     944569
Renamed 0 1 2 3 4:     656904          0     503665     191258    1037156
Cycles Stalled:
IStall       8882
Rename         29
ROB     500394
Forwarded Data:     461079
Wrote Forwards Early:     445431
Performed Early Loads:     115643
Forwarded Data But No Early Write:        119
ROB Head Loads:     327506
Loads Already Written Before Retire:     514881
-----------------------------------

int rfib (int x)
{
  if (x == 0 || x == 1) return x;
  else return rfib(x-1) + rfib(x-2);
}

int ifib (int n)
{
  int x, y, z, i;

  if (n == 0 || n == 1) return n;
  x = 0; y = 1;
  for (i=2; i <= n; i++) {
       z = x + y;
       x = y;
       y = z;
  }
  return y;
}

int main (void)
{
 int d;
 int i;
 d = 20;
 printf ("Series: ");
 for (i=1; i <= d; i++) {
       printf (" %d", rfib(i));
       if (ifib (i) != rfib (i)) {
              printf ("\n*** yow *** You're hosed\n");
       }
 }
 printf ("\n");
 exit(99);
}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Loaded Executable `test/hello'
-----------------------------------------------------------------------
Boot code entry point: 0xbfc00000
User code entry point: 0x00400290
Stack pointer: 0x7fffefff
-----------------------------------------------------------------------

Hello World.
0 1 2 3 4 5 6 7 8 9
 r:100000
EXIT called, code: 0
-----------------------------------
Program Terminated.
Retired    1509920 instructions,    1516437 entered ROB
```

```
Number of flushes:        316
Ran for               548540 cycles
ICache Accesses:     505587
ICache Misses:        262
DCache Accesses:     203961
DCache Misses:         80
Average Fullness:
ROB                  28.5
ALUQ                    0
CFQ                   0
LDSTAQ                   0
LDSTQ                   9.4
Freelist(avg. free regs)                    0
RAS                   1.0
Max Fullness:
ALUQ          4
CFQ          2
LDSTAQ          3
LDSTQ         20
RAS          9
BR Predictions:     101283
Incorrect:        136
JR Predictions:        274
Incorrect:        152
Correct Path:
Mem     503563
CF     201880
Sys         28
ALU & NOPS     804449
Retired 0 1 2 3 4:      42720     101417     100945        682     302767
Renamed 0 1 2 3 4:      43440          0     201437     101057     202598
Cycles Stalled:
IStall       7345
Rename          3
ROB      35795
Forwarded Data:     300127
Wrote Forwards Early:     300098
Performed Early Loads:       1573
Forwarded Data But No Early Write:          6
ROB Head Loads:        872
Loads Already Written Before Retire:     301169
-----------------------------------
```

```
int main ()
{
  int i,j,r;
       printf("Hello World.\n");
       for(i=0;i<10;i++){
              for(j=0;j<10000;j++){
                     r++;
              }
              printf("%d ",i);
       }
       printf("\n r:%d\n",r);

  exit(0);
}
```

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Loaded Executable `test/msort'
----------------------------------------------------------------------
Boot code entry point: 0xbfc00000
User code entry point: 0x004008b8
Stack pointer: 0x7fffefff
----------------------------------------------------------------------

Array sorted correctly.

EXIT called, code: 0
-----------------------------------
```

```
Program Terminated.
Retired     378137 instructions,     457452 entered ROB
Number of flushes:       2760
Ran for               190350 cycles
ICache Accesses:     136264
ICache Misses:        190
DCache Accesses:     125980
DCache Misses:        164
Average Fullness:
ROB                  28.0
ALUQ                    0
CFQ                    0
LDSTAQ                   0
LDSTQ                  8.7
Freelist(avg. free regs)                    0
RAS                   1.0
Max Fullness:
ALUQ          4
CFQ          2
LDSTAQ          3
LDSTQ         17
RAS         10
BR Predictions:      20580
Incorrect:       2744
JR Predictions:         25
Incorrect:         12
Correct Path:
Mem     125190
CF      31285
Sys          4
ALU & NOPS     221658
Retired 0 1 2 3 4:      44177      10892      30269      34006      70997
Renamed 0 1 2 3 4:      59539          0      23384      18992      88427
Cycles Stalled:
IStall       5432
Rename          0
ROB      48914
Forwarded Data:      11443
Wrote Forwards Early:      11299
Performed Early Loads:      94571
Forwarded Data But No Early Write:        139
ROB Head Loads:      11664
Loads Already Written Before Retire:      93642
-----------------------------------
#define TYPE int
#define SIZE 500

TYPE x[SIZE];
TYPE y[SIZE];

void init (void)
{
   int i;

   x[0] = 3;
   x[1] = 7;
   for (i=1; i < SIZE-1; i++) {
      x[i+1] = x[i]+(x[i-1]^128);
   }
}

void dumpoutput (void)
{
   int i;
   int flag = 0;

   for (i=1; i < SIZE; i++) {
      if (x[i] < x[i-1]) {
        flag = 1;
        printf ("Error at position %d\n", i);
      }
   }
   if (!flag) printf ("Array sorted correctly.\n");
}


void mergesort (unsigned long n, TYPE *array, TYPE *array2)
{
  TYPE *p, *q, *r;
  unsigned long int i, j, k, l, m, t, k1;
  unsigned long log2;

  if (n==0) return;

  p = array; q = array2;
  for(log2=0,i=1; i < n; i*=2, log2++) ;
  for (i=1, k=2; i <= log2; i++, k*=2) {
        for (j=0; j < n; j += k) {
                l = j; m = j+(k>>1);
                if ((n-j) < k) k1 = n-j; else k1 = k;
                for (t=0; t < k1; t++)
                        if (l < (j+(k>>1)))
                                if (m < j+k1)
                                        if (p[l] < p[m]) q[j+t] = p[l++];
                                        else q[j+t] = p[m++];
                                else
                                        q[j+t] = p[l++];
                        else
                                q[j+t] = p[m++];
        }
        r = p; p = q; q = r;
  }
  if (array != p) for (i=0; i < n; i++) *array++ = *p++;
}


int main (void)
{
   init ();
   mergesort (SIZE, x, y);
   dumpoutput ();
   exit(0);
}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Loaded Executable `test/nested_branches'
-----------------------------------------------------------------------
Boot code entry point: 0xbfc00000
User code entry point: 0x00400290
Stack pointer: 0x7fffefff
-----------------------------------------------------------------------

Testing Nested Branches
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49

EXIT called, code: 0
-----------------------------------
Program Terminated.
Retired    1469454 instructions,    1573207 entered ROB
Number of flushes:       3894
Ran for               583748 cycles
ICache Accesses:     462081
ICache Misses:        424
DCache Accesses:     232784
DCache Misses:        141
Average Fullness:
ROB                  28.5
ALUQ                   0
CFQ                   0
LDSTAQ                   0
```

```
LDSTQ                   9.7
Freelist(avg. free regs)                    0
RAS                   1.1
Max Fullness:
ALUQ          4
CFQ          0
LDSTAQ          3
LDSTQ          20
RAS          8
BR Predictions:     111014
Incorrect:       3216
JR Predictions:       1139
Incorrect:        572
Correct Path:
Mem     517568
CF     212731
Sys        106
ALU & NOPS     739049
Retired 0 1 2 3 4:      27910     199412      17715      16444     322258
Renamed 0 1 2 3 4:     129153          0     115918      13305     325364
Cycles Stalled:
IStall      11530
Rename        114
ROB     110071
Forwarded Data:     295427
Wrote Forwards Early:     294436
Performed Early Loads:      24752
Forwarded Data But No Early Write:         23
ROB Head Loads:       3219
Loads Already Written Before Retire:     309513
-----------------------------------
int main ()
{
  int i,j,k,a,b,c;
  printf("Testing Nested Branches\n");

  a = 0;
  b = 0;
  c = 0;

  for(i=0;i<50;i++){
         for(j=0;j<i*3;j++){
               a++;
               if(a&1)b=0;
               else b=1;

               if(b)c++;
               else {
                      for(k=i;k>0;k--)a++;
               }
         }
       printf("%d ",i);
  }
       printf("\n");
  exit(0);
}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Loaded Executable `test/nqueen'
----------------------------------------------------------------------
Boot code entry point: 0xbfc00000
User code entry point: 0x00400290
Stack pointer: 0x7fffefff
----------------------------------------------------------------------

  0   |   (0,0), (1,2), (2,4), (3,6), (4,1), (5,3), (6,5),
  1   |   (0,0), (1,3), (2,6), (3,2), (4,5), (5,1), (6,4),
  2   |   (0,0), (1,4), (2,1), (3,5), (4,2), (5,6), (6,3),
  3   |   (0,0), (1,5), (2,3), (3,1), (4,6), (5,4), (6,2),
  4   |   (0,1), (1,3), (2,0), (3,6), (4,4), (5,2), (6,5),
  5   |   (0,1), (1,3), (2,5), (3,0), (4,2), (5,4), (6,6),
  6   |   (0,1), (1,4), (2,0), (3,3), (4,6), (5,2), (6,5),
  7   |   (0,1), (1,4), (2,2), (3,0), (4,6), (5,3), (6,5),
  8   |   (0,1), (1,4), (2,6), (3,3), (4,0), (5,2), (6,5),
  9   |   (0,1), (1,5), (2,2), (3,6), (4,3), (5,0), (6,4),
 10   |   (0,1), (1,6), (2,4), (3,2), (4,0), (5,5), (6,3),
 11   |   (0,2), (1,0), (2,5), (3,1), (4,4), (5,6), (6,3),
 12   |   (0,2), (1,0), (2,5), (3,3), (4,1), (5,6), (6,4),
 13   |   (0,2), (1,4), (2,6), (3,1), (4,3), (5,5), (6,0),
 14   |   (0,2), (1,5), (2,1), (3,4), (4,0), (5,3), (6,6),
 15   |   (0,2), (1,6), (2,1), (3,3), (4,5), (5,0), (6,4),
 16   |   (0,2), (1,6), (2,3), (3,0), (4,4), (5,1), (6,5),
 17   |   (0,3), (1,0), (2,2), (3,5), (4,1), (5,6), (6,4),
 18   |   (0,3), (1,0), (2,4), (3,1), (4,5), (5,2), (6,6),
 19   |   (0,3), (1,1), (2,6), (3,4), (4,2), (5,0), (6,5),
 20   |   (0,3), (1,5), (2,0), (3,2), (4,4), (5,6), (6,1),
 21   |   (0,3), (1,6), (2,2), (3,5), (4,1), (5,4), (6,0),
 22   |   (0,3), (1,6), (2,4), (3,1), (4,5), (5,0), (6,2),
 23   |   (0,4), (1,0), (2,3), (3,6), (4,2), (5,5), (6,1),
 24   |   (0,4), (1,0), (2,5), (3,3), (4,1), (5,6), (6,2),
 25   |   (0,4), (1,1), (2,5), (3,2), (4,6), (5,3), (6,0),
 26   |   (0,4), (1,2), (2,0), (3,5), (4,3), (5,1), (6,6),
 27   |   (0,4), (1,6), (2,1), (3,3), (4,5), (5,0), (6,2),
 28   |   (0,4), (1,6), (2,1), (3,5), (4,2), (5,0), (6,3),
 29   |   (0,5), (1,0), (2,2), (3,4), (4,6), (5,1), (6,3),
 30   |   (0,5), (1,1), (2,4), (3,0), (4,3), (5,6), (6,2),
 31   |   (0,5), (1,2), (2,0), (3,3), (4,6), (5,4), (6,1),
 32   |   (0,5), (1,2), (2,4), (3,6), (4,0), (5,3), (6,1),
 33   |   (0,5), (1,2), (2,6), (3,3), (4,0), (5,4), (6,1),
 34   |   (0,5), (1,3), (2,1), (3,6), (4,4), (5,2), (6,0),
 35   |   (0,5), (1,3), (2,6), (3,0), (4,2), (5,4), (6,1),
 36   |   (0,6), (1,1), (2,3), (3,5), (4,0), (5,2), (6,4),
 37   |   (0,6), (1,2), (2,5), (3,1), (4,4), (5,0), (6,3),
 38   |   (0,6), (1,3), (2,0), (3,4), (4,1), (5,5), (6,2),
 39   |   (0,6), (1,4), (2,2), (3,0), (4,5), (5,3), (6,1),

40 solutions found, program terminated successfully.

EXIT called, code: 2
-----------------------------------
Program Terminated.
Retired    1360242 instructions,    1955796 entered ROB
Number of flushes:      23675
Ran for               913062 cycles
ICache Accesses:     629221
ICache Misses:       3325
DCache Accesses:     512470
DCache Misses:         84
Average Fullness:
ROB                  23.9
ALUQ                    0
CFQ                    0
LDSTAQ                   0
LDSTQ                   8.0
Freelist(avg. free regs)                    0
RAS                   8.7
Max Fullness:
ALUQ          4
CFQ          0
LDSTAQ          3
LDSTQ         20
RAS         18
BR Predictions:     138551
Incorrect:      13760
JR Predictions:      19683
Incorrect:       8007
Correct Path:
Mem     501285
CF     203055
Sys       1908
ALU & NOPS     653994
```

```
Retired 0 1 2 3 4:     260501     119256     116263      64119     352914
Renamed 0 1 2 3 4:     327381          0     149026      88844     347803
Cycles Stalled:
IStall      88242
Rename         21
ROB     193904
Forwarded Data:      57937
Wrote Forwards Early:      51432
Performed Early Loads:     300581
Forwarded Data But No Early Write:       5187
ROB Head Loads:      73951
Loads Already Written Before Retire:     284209
-----------------------------------
#define   NUM_QUEENS     7
#define   FALSE          0
#define   TRUE          1

typedef int BoardPosition;

int Threatens(int x, int y, BoardPosition *board, int numPiecesPlaced);
void PrintSolution(BoardPosition *board, int numQueens, int solutionsFound);
void FindSolution(BoardPosition *board, int piecesPlaced, int *solutionsFound);

int main() {
  BoardPosition board[NUM_QUEENS];
  int piecesPlaced = 0;
  int solutionsFound = 0;


  FindSolution(board, piecesPlaced, &solutionsFound);
  printf("\n%d solutions found, program terminated successfully.\n", solutionsFound);

  exit(2);
}

int Threatens(int x, int y, BoardPosition *board, int numPiecesPlaced)  {

  int i = 0;
  int threats = FALSE;   /* set if threat detected */

  int temp;

  while ((i < numPiecesPlaced) && (threats == FALSE)) {
    if (board[i] == y)    /* test rows */
      threats = TRUE;

    /* now test diagonals */
    temp = x-i;
    if ((y == (board[i]-temp)) || (y == (board[i]+temp)))
      threats = TRUE;

    ++i;
  }

  return threats;

}

void FindSolution(BoardPosition *board, int piecesPlaced, int *solutionsFound) {

  int i;

  for (i=0; i<NUM_QUEENS; ++i) {
    if (!Threatens(piecesPlaced, i, board, piecesPlaced)) {
      board[piecesPlaced] = i;   /* record it */

      if (piecesPlaced == (NUM_QUEENS-1)) {
        PrintSolution(board, NUM_QUEENS, *solutionsFound);
        ++(*solutionsFound);
      }
      FindSolution(board, piecesPlaced+1, solutionsFound);
    }
  }

  return;

}

void PrintSolution(BoardPosition *board, int numQueens, int solutionsFound) {

  int i;


  printf("%3d   |   ", solutionsFound);

  for (i = 0; i<numQueens; ++i) {
    printf("(%d,%d), ", i, board[i]);
  }

  printf("\n");
}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Loaded Executable `test/recursion_loops'
----------------------------------------------------------------------
Boot code entry point: 0xbfc00000
User code entry point: 0x00400354
Stack pointer: 0x7fffefff
----------------------------------------------------------------------

Testing Recursion Loops
2:510 4:262652 6:67371514 8:67371512 10:67371510 12:67371508 14:67371506 16:67371504 18:67371502
20:67371500 22:67371498 24:67371496 26:67371494 28:67371492 30:67371490 32:67371488 34:67371486
36:67371484 38:67371482 40:67371480 42:67371478 44:67371476 46:67371474 48:67371472 50:67371470
52:67371468 54:67371466 56:67371464 58:67371462 60:67371460 62:67371458 64:67371456 66:67371454
68:67371452 70:67371450 72:67371448 74:67371446 76:67371444 78:67371442 80:67371440 82:67371438
84:67371436 86:67371434 88:67371432 90:67371430 92:67371428 94:67371426 96:67371424 98:67371422
100:67371420

EXIT called, code: 0
-----------------------------------
Program Terminated.
Retired     329630 instructions,     367972 entered ROB
Number of flushes:       1684
Ran for               153144 cycles
ICache Accesses:     111733
ICache Misses:        623
DCache Accesses:     102871
DCache Misses:        149
Average Fullness:
ROB                  25.0
ALUQ                    0
CFQ                   0
LDSTAQ                  0
LDSTQ                 7.9
Freelist(avg. free regs)                    0
RAS                   3.6
Max Fullness:
ALUQ          4
CFQ         0
LDSTAQ        3
LDSTQ        20
RAS         9
BR Predictions:      20202
Incorrect:        454
JR Predictions:       1954
Incorrect:       1024
Correct Path:
Mem     110919
CF      34960
```

```
Sys        206
ALU & NOPS     183545
Retired 0 1 2 3 4:      38849       9645      27802       4634      72205
Renamed 0 1 2 3 4:      44255          0      20072      27408      61401
Cycles Stalled:
IStall      16844
Rename        318
ROB      24351
Forwarded Data:      11372
Wrote Forwards Early:      11318
Performed Early Loads:      65004
Forwarded Data But No Early Write:         49
ROB Head Loads:       5566
Loads Already Written Before Retire:      73003
-----------------------------------
int q=0;
int rec(int i){
       int j,x;
       if(i<5)x = rec(i+1);
       for(j=0;j<q<<2;j++)x = x | q<<j;
       return x;
}

int main ()
{
  int i,z;
  printf("Testing Recursion Loops\n");


  for(i=0;i<50;i++){
       q++;
       q++;
       z += rec(i);
       printf("%d:%d ",q,z);
  }
       printf("\n");
  exit(0);
}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Loaded Executable `test/towers'
-----------------------------------------------------------------------
Boot code entry point: 0xbfc00000
User code entry point: 0x004003a0
Stack pointer: 0x7fffefff
-----------------------------------------------------------------------

Move disk 1 from peg A to peg B
Move disk 2 from peg A to peg C
Move disk 1 from peg B to peg C
Move disk 3 from peg A to peg B
Move disk 1 from peg C to peg A
Move disk 2 from peg C to peg B
Move disk 1 from peg A to peg B
Move disk 4 from peg A to peg C
Move disk 1 from peg B to peg C
Move disk 2 from peg B to peg A
Move disk 1 from peg C to peg A
Move disk 3 from peg B to peg C
Move disk 1 from peg A to peg B
Move disk 2 from peg A to peg C
Move disk 1 from peg B to peg C
Move disk 5 from peg A to peg B
Move disk 1 from peg C to peg A
Move disk 2 from peg C to peg B
Move disk 1 from peg A to peg B
Move disk 3 from peg C to peg A
Move disk 1 from peg B to peg C
Move disk 2 from peg B to peg A
Move disk 1 from peg C to peg A
Move disk 4 from peg C to peg B
Move disk 1 from peg A to peg B
Move disk 2 from peg A to peg C
Move disk 1 from peg B to peg C
Move disk 3 from peg A to peg B
Move disk 1 from peg C to peg A
Move disk 2 from peg C to peg B
Move disk 1 from peg A to peg B
Move disk 6 from peg A to peg C
Move disk 1 from peg B to peg C
Move disk 2 from peg B to peg A
Move disk 1 from peg C to peg A
Move disk 3 from peg B to peg C
Move disk 1 from peg A to peg B
Move disk 2 from peg A to peg C
Move disk 1 from peg B to peg C
Move disk 4 from peg B to peg A
Move disk 1 from peg C to peg A
Move disk 2 from peg C to peg B
Move disk 1 from peg A to peg B
Move disk 3 from peg C to peg A
Move disk 1 from peg B to peg C
Move disk 2 from peg B to peg A
Move disk 1 from peg C to peg A
Move disk 5 from peg B to peg C
Move disk 1 from peg A to peg B
Move disk 2 from peg A to peg C
Move disk 1 from peg B to peg C
Move disk 3 from peg A to peg B
Move disk 1 from peg C to peg A
Move disk 2 from peg C to peg B
Move disk 1 from peg A to peg B
Move disk 4 from peg A to peg C
Move disk 1 from peg B to peg C
Move disk 2 from peg B to peg A
Move disk 1 from peg C to peg A
Move disk 3 from peg B to peg C
Move disk 1 from peg A to peg B
Move disk 2 from peg A to peg C
Move disk 1 from peg B to peg C

EXIT called, code: 0
-----------------------------------
Program Terminated.
Retired     156503 instructions,     233348 entered ROB
Number of flushes:       3467
Ran for                164497 cycles
ICache Accesses:       79145
ICache Misses:        2313
DCache Accesses:       59867
DCache Misses:         331
Average Fullness:
ROB                   16.9
ALUQ                    0
CFQ                    0
LDSTAQ                   0
LDSTQ                   5.2
Freelist(avg. free regs)                  0
RAS                    8.7
Max Fullness:
ALUQ          4
CFQ          0
LDSTAQ          3
LDSTQ         20
RAS         16
BR Predictions:       21491
Incorrect:        833
JR Predictions:        4163
Incorrect:       2128
Correct Path:
Mem      54237
```

```
CF      31551
Sys        506
ALU & NOPS      70209
Retired 0 1 2 3 4:      81173      18837      16710       9982      37786
Renamed 0 1 2 3 4:      90822          0      23594      14132      35941
Cycles Stalled:
IStall      58667
Rename          0
ROB      27028
Forwarded Data:       1574
Wrote Forwards Early:       1571
Performed Early Loads:      26194
Forwarded Data But No Early Write:          0
ROB Head Loads:      11084
Loads Already Written Before Retire:      20564
-----------------------------------
char *A = "A";
char *B = "B";
char *C = "C";

char s[5];

#define TOWER_SIZE 6

void
printmessage(int n, char *from_peg, char *to_peg)
{
  printf("Move disk %d from peg %s to peg %s\n",
        n, from_peg, to_peg);
}

void
towers(int n, char *from_peg, char *to_peg, char *aux_peg)
{
  if (n == 1) {
    printmessage(n, from_peg, to_peg);
    return;
  };

  towers(n-1, from_peg, aux_peg, to_peg);

  printmessage(n, from_peg, to_peg);

  towers(n-1, aux_peg, to_peg, from_peg);
}

int
main(int argc, char **argv)
{
  towers(TOWER_SIZE, A, C, B);
  exit(0);
}
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

******************************************
cpu.v

`include "mips.h"

module cpu(CLK,RESET,Bus,Addr,Write,Read,Valid);
input   CLK;                  // Clock
input   RESET;                // Master reset
input   Valid;                // Input from mem system on fills
inout  [31:0] Bus;                    // Data Bus between cpu & memory
wire   [31:0] Bus;
output [31:0] Addr;                   // Address Bus between cpu & memory
wire   [31:0] Addr;
output   Read;                // Bus Read
wire     Read;
output   Write;               // Bus Write
wire     Write;
```

```
// Stats
reg  [31:0] numLoads;
reg  [31:0] numStores;
reg  [31:0] numDMisses;
reg  [31:0] numIMisses;
reg  [31:0] num_instructions_retired;
reg  [31:0] num_instructions_inserted;
reg  [31:0] num_flushes;
reg  [31:0] num_instructions_fetched;
reg  [31:0] num_instructions_decoded;
reg  [31:0] num_branches;
reg  [31:0] num_jrs;
reg  [31:0] num_correct_br_pred;
reg  [31:0] num_correct_jr_pred;
reg  [31:0]
num_data_forwarded,num_data_forwarded_written_ahead_of_time,num_early_load_accesses,num_data_forw
arded_not_written,num_rob_head_loads,num_loads_written_before_retire;
reg  [31:0] num_cycles_istall,num_cycles_renamestall,num_cycles_robstall;

//Control Regs
wire ROB_Full;
reg  [31:0] PC,nextFetchAt,correctTarget;
wire [31:0] InAddrA,InAddrB,InAddrC,InAddrD;
reg  [31:0] InstrA,InstrB,InstrC,InstrD;
wire [31:0] InInstA,InInstB,InInstC,InInstD;
reg  [31:0] PCA,PCB,PCC,PCD;
reg  [31:0] PC_inc,PC_same,PC_br,PC_jmp;
reg  Istalled;
wire IStall,DStall;
wire renameStall;
reg fullFlush,fullFlushed,robFulled;
reg [5:0] MAP[33:0];//retirement rat, moved out of its own module for timing
wire [31:0] RAS_PC;
wire BR_Pred;

//PC Related Stuff
always @(negedge CLK)begin
	if(RESET)begin
		PC = 32'hbfc00000;
	end
	else begin
		if(IStall || (Istalled&&!fullFlush) /*|| (robFulled&&!fullFlush)*/ /*||
(ROB_Full&&!fullFlush) || (renameStall&&!fullFlush)*/)begin
			PC = PC;
		end
		else if(fullFlush)begin
			PC = correctTarget;
		end
		else begin
			PC = nextFetchAt;
		end
	end
end

//ICache
assign InAddrA = PC;
assign InAddrB = PC+4;
assign InAddrC = PC+8;
assign InAddrD = PC+12;

always @(posedge CLK)begin
	if(RESET)begin
			PCA = InAddrA;
			PCB = InAddrB;
			PCC = InAddrC;
			PCD = InAddrD;
			InstrA = InInstA;
			InstrB = InInstB;
			InstrC = InInstC;
			InstrD = InInstD;
```

```
                Istalled = 1;
                num_cycles_istall = 0;
                num_cycles_renamestall = 0;
                num_cycles_robstall = 0;
end
else begin
        if(IStall)num_cycles_istall = num_cycles_istall + 1;
        if(renameStall)num_cycles_renamestall = num_cycles_renamestall+1;
        if(ROB_Full)num_cycles_robstall = num_cycles_robstall+1;

        if(IStall)begin
                if(fullFlush)begin
                        PCA = 0;
                        PCB = 0;
                        PCC = 0;
                        PCD = 0;
                        InstrA = 0;
                        InstrB = 0;
                        InstrC = 0;
                        InstrD = 0;
                end
                else begin
                        if(ROB_Full||renameStall)begin
                                PCA = PCA;
                                PCB = PCB;
                                PCC = PCC;
                                PCD = PCD;
                                InstrA = InstrA;
                                InstrB = InstrB;
                                InstrC = InstrC;
                                InstrD = InstrD;
                        end
                        else begin
                                PCA = InAddrA;
                                PCB = InAddrB;
                                PCC = InAddrC;
                                PCD = InAddrD;
                                InstrA = InInstA;
                                InstrB = InInstB;
                                InstrC = InInstC;
                                InstrD = InInstD;
                        end
                end
                Istalled = IStall;
                fullFlushed = fullFlush;
                robFulled = ROB_Full;
        end
        else begin
                if(fullFlush)begin
                        PCA = 0;
                        PCB = 0;
                        PCC = 0;
                        PCD = 0;
                        InstrA = 0;
                        InstrB = 0;
                        InstrC = 0;
                        InstrD = 0;
                end
                else begin
                        if((ROB_Full||renameStall)&&!Istalled)begin
                                PCA = PCA;
                                PCB = PCB;
                                PCC = PCC;
                                PCD = PCD;
                                InstrA = InstrA;
                                InstrB = InstrB;
                                InstrC = InstrC;
                                InstrD = InstrD;
                        end
                        else begin
                                PCA = InAddrA;
```

```
                                    PCB = InAddrB;
                                    PCC = InAddrC;
                                    PCD = InAddrD;
                                    InstrA = InInstA;
                                    InstrB = InInstB;
                                    InstrC = InInstC;
                                    InstrD = InInstD;
                            end
                    end

                    Istalled = IStall;
                    fullFlushed = fullFlush;
                    robFulled = ROB_Full;
            end
        end
end

reg [7:0] memOp;
reg isMemRetire;
reg [31:0] MAR;
reg [31:0] SMDR;
wire [31:0] cacheOut;
wire [31:0] store_value,pre_store_value;
wire isLoad;
mem i_cache_d_cache(
 .CLK(CLK),
 .memOperation(memOp),
 .isMem(isMemRetire),
 .RESET(RESET),
 .MAR(MAR),
 .Valid(Valid),
 .SMDR(SMDR),
 .IaddrA(InAddrA),
 .IaddrB(InAddrB),
 .IaddrC(InAddrC),
 .IaddrD(InAddrD),
 .Bus(Bus),
 .Read(Read),
 .Write(Write),
 .Addr(Addr),
 .cacheOut(cacheOut),
 .IinA(InInstA),
 .IinB(InInstB),
 .IinC(InInstC),
 .IinD(InInstD),
 .Istall(IStall),
 .Dstall(DStall),
 .isLoad(isLoad)
);

//Decode / rename / predict
reg [31:0] CF_PC,Link_PC;
reg push_link,pop_link,br_dir;
reg [2:0] numInstr,r_numInstr;
wire [2:0] numRenamed;
//Values coming out of decoder
wire [7:0] opA1,opA2,opB1,opB2,opC1,opC2,opD1,opD2;
wire [1:0] queueA,queueB,queueC,queueD;
wire [5:0] rsA,rsB,rsC,rsD,rtA,rtB,rtC,rtD;
wire [5:0] rdA1,rdA2,rdB1,rdB2,rdC1,rdC2,rdD1,rdD2;
wire usesImmA,usesImmB,usesImmC,usesImmD;
wire [31:0] immA,immB,immC,immD;
wire isSysA,isMemA,isMDA,isALUA,isSysB,isMemB,isMDB,isALUB;
wire isSysC,isMemC,isMDC,isALUC,isSysD,isMemD,isMDD,isALUD;
wire [2:0] isCFA,isCFB,isCFC,isCFD;
wire [31:0] targetA,targetB,targetC,targetD;
//Values going to rename
reg [7:0] d_opA,d_opB,d_opC,d_opD;
reg [1:0] d_queueA,d_queueB,d_queueC,d_queueD;
reg [5:0] d_rsA,d_rsB,d_rsC,d_rsD,d_rtA,d_rtB,d_rtC,d_rtD;
reg [5:0] d_rdA,d_rdB,d_rdC,d_rdD;
reg d_usesImmA,d_usesImmB,d_usesImmC,d_usesImmD;
reg [31:0] d_immA,d_immB,d_immC,d_immD;
reg d_isSysA,d_isMemA,d_isMDA,d_isALUA,d_isSysB,d_isMemB,d_isMDB,d_isALUB;
reg d_isSysC,d_isMemC,d_isMDC,d_isALUC,d_isSysD,d_isMemD,d_isMDD,d_isALUD;
reg [2:0] d_isCFA,d_isCFB,d_isCFC,d_isCFD;
reg [31:0] d_targetA,d_targetB,d_targetC,d_targetD;
reg [31:0] d_PCA,d_PCB,d_PCC,d_PCD;
reg d_pred_A,d_pred_B,d_pred_C,d_pred_D;
reg [31:0] d_instrA,d_instrB,d_instrC,d_instrD;
reg [31:0] r_instrA,r_instrB,r_instrC,r_instrD;
//temporary swap values
reg [7:0] t_op;
reg [1:0] t_queue;
reg [5:0] t_rs;
reg [5:0] t_rt;
reg [5:0] t_rd;
reg t_usesImm;
reg [31:0] t_imm;
reg t_isSys,t_isMem,t_isMD,t_isALU;
reg [2:0] t_isCF;
reg [31:0] t_target;
reg [31:0] t_PC;
reg t_pred;
reg [31:0] t_instr;
reg ab_swap,bc_swap,cd_swap;
//Set up instructions from decode to go into rename and rest of cpu
//mult/div expanded to two instructions
//control flow ends decode unless untaken branch and no other branches in fetch group
//swap delay slot with CF to make it easier to flush after misprediction gets to ROB head
always @(*)begin
        numInstr = 4;
        nextFetchAt = PCD+4;
        //assign initial values
        d_opA = opA1;               d_opB = opB1;               d_opC = opC1;
        d_opD = opD1;
        d_queueA = queueA;          d_queueB = queueB;          d_queueC = queueC;
        d_queueD = queueD;
        d_rsA = rsA;                d_rsB = rsB;                d_rsC = rsC;
        d_rsD = rsD;
        d_rtA = rtA;                d_rtB = rtB;                d_rtC = rtC;
        d_rtD = rtD;
        d_rdA = rdA1;               d_rdB = rdB1;               d_rdC = rdC1;
        d_rdD = rdD1;
        d_usesImmA = usesImmA; d_usesImmB = usesImmB; d_usesImmC = usesImmC; d_usesImmD = usesImmD;
        d_immA = immA;              d_immB = immB;              d_immC = immC;
        d_immD = immD;
        d_isSysA = isSysA;          d_isSysB = isSysB;          d_isSysC = isSysC;
        d_isSysD = isSysD;
        d_isMemA = isMemA;          d_isMemB = isMemB;          d_isMemC = isMemC;
        d_isMemD = isMemD;
        d_isMDA = isMDA;            d_isMDB = isMDB;            d_isMDC = isMDC;
        d_isMDD = isMDD;
        d_isALUA = isALUA;          d_isALUB = isALUB;          d_isALUC = isALUC;
        d_isALUD = isALUD;
        d_isCFA = isCFA;            d_isCFB = isCFB;            d_isCFC = isCFC;
        d_isCFD = isCFD;
        d_targetA = targetA;   d_targetB = targetB;   d_targetC = targetC;   d_targetD = targetD;
        d_PCA = PCA;                d_PCB = PCB;                d_PCC = PCC;
        d_PCD = PCD;
        d_pred_A = 0;               d_pred_B = 0;               d_pred_C = 0;
        d_pred_D = 0;
        d_instrA = InstrA; d_instrB = InstrB; d_instrC = InstrC; d_instrD = InstrD;
        ab_swap = 0; bc_swap = 0; cd_swap = 0;

        //expand multiplies/divides
        if(d_isMDA)begin
                if(d_isMDB)begin
                //mult/div back to back, expand both, a to a/b, b to c/d
                        d_opB = opA2;               d_opC = opB1;               d_opD = opB2;
                        d_queueB = queueA;          d_queueC = queueB;          d_queueD =
queueB;
```

```
                        d_rsB = rsA;                  d_rsC = rsB;                  d_rsD = rsB;
                        d_rtB = rtA;                  d_rtC = rtB;                  d_rtD = rtB;
                        d_rdB = rdA2;                 d_rdC = rdB1;                 d_rdD = rdB2;
                        d_usesImmB = usesImmA; d_usesImmC = usesImmB; d_usesImmD = usesImmB;
                        d_immB = immA;                d_immC = immB;                d_immD = immB;
                        d_isSysB = isSysA;            d_isSysC = isSysB;            d_isSysD =
isSysB;
                        d_isMemB = isMemA;            d_isMemC = isMemB;            d_isMemD =
isMemB;
                        d_isMDB = isMDA;              d_isMDC = isMDB;              d_isMDD =
isMDB;
                        d_isALUB = isALUA;            d_isALUC = isALUB;            d_isALUD =
isALUB;
                        d_isCFB = isCFA;              d_isCFC = isCFB;              d_isCFD =
isCFB;
                        d_targetB = targetA;   d_targetC = targetB;   d_targetD = targetB;
                        d_PCB = PCA;                  d_PCC = PCB;                  d_PCD = PCB;
                        d_instrB = InstrA;     d_instrC = InstrB;     d_instrD = InstrB;
                        nextFetchAt = PCB+4;
                end
                else if(d_isMDC)begin
                //won't be room for both md's to be expanded
                //A expands to A and B, B to C, D empty because can't expand that mult
                        d_opB = opA2;                 d_opC = opB1;                 d_opD = 0;
                        d_queueB = queueA;            d_queueC = queueB;            d_queueD = 0;
                        d_rsB = rsA;                  d_rsC = rsB;                  d_rsD = 0;
                        d_rtB = rtA;                  d_rtC = rtB;                  d_rtD = 0;
                        d_rdB = rdA2;                 d_rdC = rdB1;                 d_rdD = 0;
                        d_usesImmB = usesImmA; d_usesImmC = usesImmB; d_usesImmD = 0;
                        d_immB = immA;                d_immC = immB;                d_immD = 0;
                        d_isSysB = isSysA;            d_isSysC = isSysB;            d_isSysD = 0;
                        d_isMemB = isMemA;            d_isMemC = isMemB;            d_isMemD = 0;
                        d_isMDB = isMDA;              d_isMDC = isMDB;              d_isMDD = 0;
                        d_isALUB = isALUA;            d_isALUC = isALUB;            d_isALUD = 0;
                        d_isCFB = isCFA;              d_isCFC = isCFB;              d_isCFD = 0;
                        d_targetB = targetA;   d_targetC = targetB;   d_targetD = 0;
                        d_PCB = PCA;                  d_PCC = PCB;                  d_PCD = 0;
                        d_instrB = InstrA;     d_instrC = InstrB;     d_instrD = 0;
                        nextFetchAt = PCB+4;
                        numInstr = 3;
                end
                else begin
                //only one md, first slot, if there was one in last slot it got pushed out
                //A expands to A and B, B to C, C to D D pushed out
                        d_opB = opA2;                 d_opC = opB1;                 d_opD = opC1;
                        d_queueB = queueA;            d_queueC = queueB;            d_queueD =
queueC;
                        d_rsB = rsA;                  d_rsC = rsB;                  d_rsD = rsC;
                        d_rtB = rtA;                  d_rtC = rtB;                  d_rtD = rtC;
                        d_rdB = rdA2;                 d_rdC = rdB1;                 d_rdD = rdC1;
                        d_usesImmB = usesImmA; d_usesImmC = usesImmB; d_usesImmD = usesImmC;
                        d_immB = immA;                d_immC = immB;                d_immD = immC;
                        d_isSysB = isSysA;            d_isSysC = isSysB;            d_isSysD =
isSysC;
                        d_isMemB = isMemA;            d_isMemC = isMemB;            d_isMemD =
isMemC;
                        d_isMDB = isMDA;              d_isMDC = isMDB;              d_isMDD =
isMDC;
                        d_isALUB = isALUA;            d_isALUC = isALUB;            d_isALUD =
isALUC;
                        d_isCFB = isCFA;              d_isCFC = isCFB;              d_isCFD =
isCFC;
                        d_targetB = targetA;   d_targetC = targetB;   d_targetD = targetC;
                        d_PCB = PCA;                  d_PCC = PCB;                  d_PCD = PCC;
                        d_instrB = InstrA;     d_instrC = InstrB;     d_instrD = InstrC;
                        nextFetchAt = PCC+4;
                end
        end
        else if(d_isMDB)begin
                if(d_isMDC)begin
```

```
                        //Mult in 2nd spot takes B and C, pushes D out, C can't expand so D empty,
A same
                        d_opC = opB2;                   d_opD = 0;
                        d_queueC = queueB;              d_queueD = 0;
                        d_rsC = rsB;                    d_rsD = 0;
                        d_rtC = rtB;                    d_rtD = 0;
                        d_rdC = rdB2;                   d_rdD = 0;
                        d_usesImmC = usesImmB;  d_usesImmD = 0;
                        d_immC = immB;                  d_immD = 0;
                        d_isSysC = isSysB;              d_isSysD = 0;
                        d_isMemC = isMemB;              d_isMemD = 0;
                        d_isMDC = isMDB;                d_isMDD = 0;
                        d_isALUC = isALUB;              d_isALUD = 0;
                        d_isCFC = isCFB;                d_isCFD = 0;
                        d_targetC = targetB;    d_targetD = 0;
                        d_PCC = PCB;                    d_PCD = 0;
                        d_instrC = InstrB;      d_instrD = 0;
                        nextFetchAt = PCB+4;
                        numInstr = 3;
                end
                else begin
                        //only B is mult, a same, b to b and c, c to d
                        d_opC = opB2;                   d_opD = opC1;
                        d_queueC = queueB;              d_queueD = queueC;
                        d_rsC = rsB;                    d_rsD = rsC;
                        d_rtC = rtB;                    d_rtD = rtC;
                        d_rdC = rdB2;                   d_rdD = rdC1;
                        d_usesImmC = usesImmB;  d_usesImmD = usesImmC;
                        d_immC = immB;                  d_immD = immC;
                        d_isSysC = isSysB;              d_isSysD = isSysC;
                        d_isMemC = isMemB;              d_isMemD = isMemC;
                        d_isMDC = isMDB;                d_isMDD = isMDC;
                        d_isALUC = isALUB;              d_isALUD = isALUC;
                        d_isCFC = isCFB;                d_isCFD = isCFC;
                        d_targetC = targetB;    d_targetD = targetC;
                        d_PCC = PCB;                    d_PCD = PCC;
                        d_instrC = InstrB;      d_instrD = InstrC;
                        nextFetchAt = PCC+4;
                end
        end
        else if(d_isMDC)begin
                        //a same, b same, c expands to c and d
                        d_opD = opC2;
                        d_queueD = queueC;
                        d_rsD = rsC;
                        d_rtD = rtC;
                        d_rdD = rdC2;
                        d_usesImmD = usesImmC;
                        d_immD = immC;
                        d_isSysD = isSysC;
                        d_isMemD = isMemC;
                        d_isMDD = isMDC;
                        d_isALUD = isALUC;
                        d_isCFD = isCFC;
                        d_targetD = targetC;
                        d_PCD = PCC;
                        d_instrD = InstrC;
                        nextFetchAt = PCC+4;
        end
        else if(d_isMDD)begin
                        //can't expand a mult in slot d, defer to next cycle
                        d_opD = 0;
                        d_queueD = 0;
                        d_rsD = 0;
                        d_rtD = 0;
                        d_rdD = 0;
                        d_usesImmD = 0;
                        d_immD = 0;
                        d_isSysD = 0;
                        d_isMemD = 0;
                        d_isMDD = 0;
                        d_isALUD = 0;
                        d_isCFD = 0;
                        d_targetD = 0;
                        d_PCD = 0;
                        d_instrD = 0;
                        numInstr = 3;
                        nextFetchAt = PCC+4;

                        if(d_isCFC[0])begin
                                d_opC = 0;
                                d_queueC =0;
                                d_rsC =0;
                                d_rtC =0;
                                d_rdC =0;
                                d_usesImmC =0;
                                d_immC =0;
                                d_isSysC =0;
                                d_isMemC =0;
                                d_isMDC =0;
                                d_isALUC =0;
                                d_isCFC =0;
                                d_targetC =0;
                                d_PCC =0;
                                d_instrC = 0;
                                numInstr = 2;
                                nextFetchAt = PCB+4;
                        end
         end

         //handle control flow instructions
         //swap with delay slot,no more instructions decoded if pred not taken
         //CF can't be last instruction fetched to make sure we get the delay slot

         push_link = 0;
         pop_link = 0;

         if(d_isCFA[0])begin
                 if(d_isMDB)begin
                        //Swap A and C
                        t_op = d_opA;
                        t_queue = d_queueA;
                        t_rs = d_rsA;
                        t_rt = d_rtA;
                        t_rd = d_rdA;
                        t_usesImm = d_usesImmA;
                        t_imm = d_immA;
                        t_isSys = d_isSysA;
                        t_isMem = d_isMemA;
                        t_isMD = d_isMDA;
                        t_isALU = d_isALUA;
                        t_isCF = d_isCFA;
                        t_target = d_targetA;
                        t_PC = d_PCA;
                        t_pred = 0;
                        t_instr = d_instrA;

                        d_opA = d_opC;                        d_opC = t_op;
                        d_queueA = d_queueC;          d_queueC = t_queue;
                        d_rsA = d_rsC;                        d_rsC = t_rs;
                        d_rtA = d_rtC;                        d_rtC = t_rt;
                        d_rdA = d_rdC;                        d_rdC = t_rd;
                        d_usesImmA = d_usesImmC;      d_usesImmC = t_usesImm;
                        d_immA = d_immC;                      d_immC = t_imm;
                        d_isSysA = d_isSysC;          d_isSysC = t_isSys;
                        d_isMemA = d_isMemC;          d_isMemC = t_isMem;
                        d_isMDA = d_isMDC;                    d_isMDC = t_isMD;
                        d_isALUA = d_isALUC;          d_isALUC = t_isALU;
                        d_isCFA = d_isCFC;                    d_isCFC = t_isCF;
                        d_targetA = d_targetC;        d_targetC = t_target;
                        d_PCA = d_PCC;                        d_PCC = t_PC;
                        d_pred_A = 0;                         d_pred_C = t_pred;
```

```
			d_instrA = d_instrC;		d_instrC = t_instr;
			CF_PC = d_PCC;
			Link_PC = d_immC;
			if(d_isCFC[2] && d_rdC == 31)push_link = 1;
			if((d_opC == `select_qc_jr || d_opC == 
`select_qc_jalr)&&d_rsC==31)pop_link = 1;
			br_dir = d_isCFC[1];

			if(d_opC == `select_qc_jr || d_opC == `select_qc_jalr)begin
				nextFetchAt = RAS_PC;
				d_targetC = RAS_PC;
			end
			else if(d_opC == `select_qc_j || d_opC == `select_qc_jal)begin
				nextFetchAt = d_targetC;
			end
			else begin
				if(BR_Pred)begin
					nextFetchAt = d_targetC;
					d_pred_C = 1;
				end
				else begin
					nextFetchAt = d_PCC + 8;
					d_pred_C = 0;
				end
			end
			//wouldnt have to kill if predicted not taken and is not CF

				d_opD = 0;
				d_queueD = 0;
				d_rsD = 0;
				d_rtD = 0;
				d_rdD = 0;
				d_usesImmD = 0;
				d_immD = 0;
				d_isSysD = 0;
				d_isMemD = 0;
				d_isMDD = 0;
				d_isALUD = 0;
				d_isCFD = 0;
				d_targetD = 0;
				d_PCD = 0;
				d_instrD = 0;
				numInstr = 3;

		end
		else begin
			//Swap A and B
			t_op = d_opA;
			t_queue = d_queueA;
			t_rs = d_rsA;
			t_rt = d_rtA;
			t_rd = d_rdA;
			t_usesImm = d_usesImmA;
			t_imm = d_immA;
			t_isSys = d_isSysA;
			t_isMem = d_isMemA;
			t_isMD = d_isMDA;
			t_isALU = d_isALUA;
			t_isCF = d_isCFA;
			t_target = d_targetA;
			t_PC = d_PCA;
			t_pred = 0;
			t_instr = d_instrA;
			//Swap A and B
			ab_swap = 1;
			d_opA = d_opB;			d_opB = t_op;
			d_queueA = d_queueB;		d_queueB = t_queue;
			d_rsA = d_rsB;			d_rsB = t_rs;
			d_rtA = d_rtB;			d_rtB = t_rt;
			d_rdA = d_rdB;			d_rdB = t_rd;
			d_usesImmA = d_usesImmB;	d_usesImmB = t_usesImm;
```

```
                        d_immA = d_immB;                     d_immB = t_imm;
                        d_isSysA = d_isSysB;          d_isSysB = t_isSys;
                        d_isMemA = d_isMemB;          d_isMemB = t_isMem;
                        d_isMDA = d_isMDB;                   d_isMDB = t_isMD;
                        d_isALUA = d_isALUB;          d_isALUB = t_isALU;
                        d_isCFA = d_isCFB;                   d_isCFB = t_isCF;
                        d_targetA = d_targetB;        d_targetB = t_target;
                        d_PCA = d_PCB;                       d_PCB = t_PC;
                        d_pred_A = 0;                        d_pred_B = t_pred;
                        d_instrA = d_instrB;          d_instrB = t_instr;
                        CF_PC = d_PCB;
                        Link_PC = d_immB;
                        if(d_isCFB[2] && d_rdB == 31)push_link = 1;
                        if((d_opB == `select_qc_jr || d_opB ==
`select_qc_jalr)&&d_rsB==31)pop_link = 1;
                        br_dir = d_isCFB[1];

                        if(d_opB == `select_qc_jr || d_opB == `select_qc_jalr)begin
                               nextFetchAt = RAS_PC;
                               d_targetB = RAS_PC;
                               d_pred_B = 1;
                        end
                        else if(d_opB == `select_qc_j || d_opB == `select_qc_jal)begin
                               nextFetchAt = d_targetB;
                               d_pred_B = 1;
                        end
                        else begin
                               if(BR_Pred)begin
                                      nextFetchAt = d_targetB;
                                      d_pred_B = 1;
                               end
                               else begin
                                      nextFetchAt = d_PCB + 8;
                                      d_pred_B = 0;
                               end
                        end
                        //wouldnt have to kill if predicted not taken and is not CF

                               d_opC = 0;
                               d_queueC =0;
                               d_rsC =0;
                               d_rtC =0;
                               d_rdC =0;
                               d_usesImmC =0;
                               d_immC =0;
                               d_isSysC =0;
                               d_isMemC =0;
                               d_isMDC =0;
                               d_isALUC =0;
                               d_isCFC =0;
                               d_targetC =0;
                               d_PCC =0;
                               d_instrC = 0;
                               d_opD = 0;
                               d_queueD = 0;
                               d_rsD = 0;
                               d_rtD = 0;
                               d_rdD = 0;
                               d_usesImmD = 0;
                               d_immD = 0;
                               d_isSysD = 0;
                               d_isMemD = 0;
                               d_isMDD = 0;
                               d_isALUD = 0;
                               d_isCFD = 0;
                               d_targetD = 0;
                               d_PCD = 0;
                               d_instrD = 0;
                               numInstr = 2;

                 end
```

```
        end
        else if(d_isCFB[0])begin
                if(d_isMDC)begin
                        //swap b and d
                        t_op = d_opB;
                        t_queue = d_queueB;
                        t_rs = d_rsB;
                        t_rt = d_rtB;
                        t_rd = d_rdB;
                        t_usesImm = d_usesImmB;
                        t_imm = d_immB;
                        t_isSys = d_isSysB;
                        t_isMem = d_isMemB;
                        t_isMD = d_isMDB;
                        t_isALU = d_isALUB;
                        t_isCF = d_isCFB;
                        t_target = d_targetB;
                        t_PC = d_PCB;
                        t_pred = 0;
                        t_instr = d_instrB;

                        d_opB = d_opD;                          d_opD = t_op;
                        d_queueB = d_queueD;            d_queueD = t_queue;
                        d_rsB = d_rsD;                          d_rsD = t_rs;
                        d_rtB = d_rtD;                          d_rtD = t_rt;
                        d_rdB = d_rdD;                          d_rdD = t_rd;
                        d_usesImmB = d_usesImmD;        d_usesImmD = t_usesImm;
                        d_immB = d_immD;                        d_immD = t_imm;
                        d_isSysB = d_isSysD;            d_isSysD = t_isSys;
                        d_isMemB = d_isMemD;            d_isMemD = t_isMem;
                        d_isMDB = d_isMDD;                      d_isMDD = t_isMD;
                        d_isALUB = d_isALUD;            d_isALUD = t_isALU;
                        d_isCFB = d_isCFD;                      d_isCFD = t_isCF;
                        d_targetB = d_targetD;          d_targetD = t_target;
                        d_PCB = d_PCD;                          d_PCD = t_PC;
                        d_pred_B = 0;                           d_pred_D = t_pred;
                        d_instrB = d_instrD;            d_instrD = t_instr;
                        CF_PC = d_PCD;
                        Link_PC = d_immD;
                        if(d_isCFD[2] && d_rdD == 31)push_link = 1;
                        if((d_opD == `select_qc_jr || d_opD ==
`select_qc_jalr)&&d_rsD==31)pop_link = 1;
                        br_dir = d_isCFD[1];

                        if(d_opD == `select_qc_jr || d_opD == `select_qc_jalr)begin
                                nextFetchAt = RAS_PC;
                                d_targetD = RAS_PC;
                        end
                        else if(d_opD == `select_qc_j || d_opD == `select_qc_jal)begin
                                nextFetchAt = d_targetD;
                        end
                        else begin
                                if(BR_Pred)begin
                                        nextFetchAt = d_targetD;
                                        d_pred_D = 1;
                                end
                                else begin
                                        nextFetchAt = d_PCD + 8;
                                end
                        end
                        numInstr = 4;
                end
                else begin
                        t_op = d_opB;
                        t_queue = d_queueB;
                        t_rs = d_rsB;
                        t_rt = d_rtB;
                        t_rd = d_rdB;
                        t_usesImm = d_usesImmB;
                        t_imm = d_immB;
                        t_isSys = d_isSysB;
                        t_isMem = d_isMemB;
                        t_isMD = d_isMDB;
                        t_isALU = d_isALUB;
                        t_isCF = d_isCFB;
                        t_target = d_targetB;
                        t_PC = d_PCB;
                        t_pred = 0;
                        t_instr = d_instrB;
                        //Swap B and C
                        bc_swap = 1;
                        d_opB = d_opC;                      d_opC = t_op;
                        d_queueB = d_queueC;        d_queueC = t_queue;
                        d_rsB = d_rsC;                      d_rsC = t_rs;
                        d_rtB = d_rtC;                      d_rtC = t_rt;
                        d_rdB = d_rdC;                      d_rdC = t_rd;
                        d_usesImmB = d_usesImmC;    d_usesImmC = t_usesImm;
                        d_immB = d_immC;                    d_immC = t_imm;
                        d_isSysB = d_isSysC;        d_isSysC = t_isSys;
                        d_isMemB = d_isMemC;        d_isMemC = t_isMem;
                        d_isMDB = d_isMDC;                  d_isMDC = t_isMD;
                        d_isALUB = d_isALUC;        d_isALUC = t_isALU;
                        d_isCFB = d_isCFC;                  d_isCFC = t_isCF;
                        d_targetB = d_targetC;      d_targetC = t_target;
                        d_PCB = d_PCC;                      d_PCC = t_PC;
                        d_pred_B = 0;                       d_pred_C = t_pred;
                        d_instrB = d_instrC;        d_instrC = t_instr;
                        CF_PC = d_PCC;
                        Link_PC = d_immC;
                        if(d_isCFC[2] && d_rdC == 31)push_link = 1;
                        if((d_opC == `select_qc_jr || d_opC ==
`select_qc_jalr)&&d_rsC==31)pop_link = 1;
                        br_dir = d_isCFC[1];

                        if(d_opC == `select_qc_jr || d_opC == `select_qc_jalr)begin
                                nextFetchAt = RAS_PC;
                                d_targetC = RAS_PC;
                                d_pred_C = 1;
                        end
                        else if(d_opC == `select_qc_j || d_opC == `select_qc_jal)begin
                                nextFetchAt = d_targetC;
                                d_pred_C = 1;
                        end
                        else begin
                                if(BR_Pred)begin
                                        nextFetchAt = d_targetC;
                                        d_pred_C = 1;
                                end
                                else begin
                                        d_pred_C = 0;
                                        nextFetchAt = d_PCC + 8;
                                end
                        end
                        //wouldnt have to kill if predicted not taken and is not CF

                                d_opD = 0;
                                d_queueD = 0;
                                d_rsD = 0;
                                d_rtD = 0;
                                d_rdD = 0;
                                d_usesImmD = 0;
                                d_immD = 0;
                                d_isSysD = 0;
                                d_isMemD = 0;
                                d_isMDD = 0;
                                d_isALUD = 0;
                                d_isCFD = 0;
                                d_targetD = 0;
                                d_PCD = 0;
                                d_instrD = 0;

                                numInstr = 3;
```

```
		end
	end
	else if(d_isCFC[0])begin
		t_op = d_opC;
		t_queue = d_queueC;
		t_rs = d_rsC;
		t_rt = d_rtC;
		t_rd = d_rdC;
		t_usesImm = d_usesImmC;
		t_imm = d_immC;
		t_isSys = d_isSysC;
		t_isMem = d_isMemC;
		t_isMD = d_isMDC;
		t_isALU = d_isALUC;
		t_isCF = d_isCFC;
		t_target = d_targetC;
		t_PC = d_PCC;
		t_pred = 0;
		t_instr = d_instrC;
		//Swap C and D
		cd_swap = 1;
		d_opC = d_opD;				d_opD = t_op;
		d_queueC = d_queueD;		d_queueD = t_queue;
		d_rsC = d_rsD;				d_rsD = t_rs;
		d_rtC = d_rtD;				d_rtD = t_rt;
		d_rdC = d_rdD;				d_rdD = t_rd;
		d_usesImmC = d_usesImmD;	d_usesImmD = t_usesImm;
		d_immC = d_immD;			d_immD = t_imm;
		d_isSysC = d_isSysD;		d_isSysD = t_isSys;
		d_isMemC = d_isMemD;		d_isMemD = t_isMem;
		d_isMDC = d_isMDD;			d_isMDD = t_isMD;
		d_isALUC = d_isALUD;		d_isALUD = t_isALU;
		d_isCFC = d_isCFD;			d_isCFD = t_isCF;
		d_targetC = d_targetD;		d_targetD = t_target;
		d_PCC = d_PCD;				d_PCD = t_PC;
		d_pred_C = 0;				d_pred_D = t_pred;
		d_instrC = d_instrD;		d_instrD = t_instr;
		CF_PC = d_PCD;
		Link_PC = d_immD;
		if(d_isCFD[2] && d_rdD == 31)push_link = 1;
		if((d_opD == `select_qc_jr || d_opD == `select_qc_jalr)&&d_rsD==31)pop_link = 1;
		br_dir = d_isCFD[1];

		if(d_opD == `select_qc_jr || d_opD == `select_qc_jalr)begin
			nextFetchAt = RAS_PC;
			d_targetD = RAS_PC;
		end
		else if(d_opD == `select_qc_j || d_opD == `select_qc_jal)begin
			nextFetchAt = d_targetD;
		end
		else begin
			if(BR_Pred)begin
				nextFetchAt = d_targetD;
				d_pred_D = 1;
			end
			else begin
				nextFetchAt = d_PCD + 8;
			end
		end
	end
	else if(d_isCFD[0])begin
		//defer til next cycle so we can get delay slot at same time
		d_opD = 0;
		d_queueD = 0;
		d_rsD = 0;
		d_rtD = 0;
		d_rdD = 0;
		d_usesImmD = 0;
		d_immD = 0;
		d_isSysD = 0;
```

```
                d_isMemD = 0;
                d_isMDD = 0;
                d_isALUD = 0;
                d_isCFD = 0;
                d_targetD = 0;
                d_PCD = 0;
                d_instrD = 0;
                numInstr = 3;
                nextFetchAt = PCC+4;
        end
        if((Istalled || ROB_Full || fullFlush || fullFlushed ||
renameStall)&&!(IStall&&robFulled&&!ROB_Full))begin
                if(!renameStall)numInstr = 0;
                d_queueA = 0;
                d_queueB = 0;
                d_queueC = 0;
                d_queueD = 0;
                pop_link = 0;
                push_link = 0;
        end
end

//Values after the register separating rename from rob/queue/execute
reg [7:0] r_opA,r_opB,r_opC,r_opD;
reg [1:0] r_queueA,r_queueB,r_queueC,r_queueD;
wire [5:0] r_rsA,r_rsB,r_rsC,r_rsD,r_rtA,r_rtB,r_rtC,r_rtD;
wire [5:0] r_rdA,r_rdB,r_rdC,r_rdD;
reg [5:0] r_lrdA,r_lrdB,r_lrdC,r_lrdD;
reg r_usesImmA,r_usesImmB,r_usesImmC,r_usesImmD;
reg [31:0] r_immA,r_immB,r_immC,r_immD;
reg r_isSysA,r_isMemA,r_isMDA,r_isALUA,r_isSysB,r_isMemB,r_isMDB,r_isALUB;
reg r_isSysC,r_isMemC,r_isMDC,r_isALUC,r_isSysD,r_isMemD,r_isMDD,r_isALUD;
reg [2:0] r_isCFA,r_isCFB,r_isCFC,r_isCFD;
reg [31:0] r_targetA,r_targetB,r_targetC,r_targetD;
reg [31:0] r_PCA,r_PCB,r_PCC,r_PCD;
reg r_pred_A,r_pred_B,r_pred_C,r_pred_D;
wire  ab_dependant,bc_dependant,cd_dependant;
reg r_adep,r_bdep,r_cdep,r_ddep;


//Pipe to next stage
always @(posedge CLK)begin
        r_opA = d_opA;                r_opB = d_opB;                r_opC = d_opC;
        r_opD = d_opD;
        r_lrdA = d_rdA;                       r_lrdB = d_rdB;                       r_lrdC =
d_rdC;                r_lrdD = d_rdD;
        r_queueA = d_queueA;   r_queueB = d_queueB;   r_queueC = d_queueC;   r_queueD = d_queueD;
        r_usesImmA = d_usesImmA;r_usesImmB = d_usesImmB;r_usesImmC = d_usesImmC;r_usesImmD =
d_usesImmD;
        r_immA = d_immA;              r_immB = d_immB;              r_immC = d_immC;
        r_immD = d_immD;
        r_isSysA = d_isSysA;   r_isSysB = d_isSysB;   r_isSysC = d_isSysC;   r_isSysD = d_isSysD;
        r_isMemA = d_isMemA;   r_isMemB = d_isMemB;   r_isMemC = d_isMemC;   r_isMemD = d_isMemD;
        r_isMDA = d_isMDA;            r_isMDB = d_isMDB;            r_isMDC = d_isMDC;
        r_isMDD = d_isMDD;
        r_isCFA = d_isCFA;            r_isCFB = d_isCFB;            r_isCFC = d_isCFC;
        r_isCFD = d_isCFD;
        r_isALUA = d_isALUA;   r_isALUB = d_isALUB;   r_isALUC = d_isALUC;   r_isALUD = d_isALUD;
        r_targetA = d_targetA;  r_targetB = d_targetB;  r_targetC = d_targetC;  r_targetD =
d_targetD;
        r_PCA = d_PCA;                r_PCB = d_PCB;                r_PCC = d_PCC;
        r_PCD = d_PCD;
        r_pred_A = d_pred_A;   r_pred_B = d_pred_B;   r_pred_C = d_pred_C;   r_pred_D = d_pred_D;
        r_instrA = d_instrA; r_instrB = d_instrB; r_instrC = d_instrC; r_instrD = d_instrD;
        r_adep = ab_dependant;
        r_bdep = bc_dependant;
        r_cdep = cd_dependant;
        r_ddep = 0;
        if((fullFlush||ROB_Full || renameStall ||Istalled ||
fullFlushed)&&!(IStall&&robFulled&&!ROB_Full))begin
                r_numInstr = 0;
```

```
               r_queueA = 0;  r_queueB = 0;  r_queueC = 0;  r_queueD = 0;
       end
       else r_numInstr = numInstr;
end

decode decodeA(
 .instr(InstrA),
 .operation(opA1),
 .operation2(opA2),
 .queue(queueA),
 .rs(rsA),
 .rt(rtA),
 .rd(rdA1),
 .rd2(rdA2),
 .usesImmediate(usesImmA),
 .immediate(immA),
 .isSyscall(isSysA),
 .isMem(isMemA),
 .isMultDiv(isMDA),
 .isCF(isCFA),
 .isALU(isALUA),
 .PC(PCA),
 .fullFlush(1'b0),
 .target(targetA)
);
decode decodeB(
 .instr(InstrB),
 .operation(opB1),
 .operation2(opB2),
 .queue(queueB),
 .rs(rsB),
 .rt(rtB),
 .rd(rdB1),
 .rd2(rdB2),
 .usesImmediate(usesImmB),
 .immediate(immB),
 .isSyscall(isSysB),
 .isMem(isMemB),
 .isMultDiv(isMDB),
 .isCF(isCFB),
 .isALU(isALUB),
 .PC(PCB),
 .fullFlush(1'b0),
 .target(targetB)
);
decode decodeC(
 .instr(InstrC),
 .operation(opC1),
 .operation2(opC2),
 .queue(queueC),
 .rs(rsC),
 .rt(rtC),
 .rd(rdC1),
 .rd2(rdC2),
 .usesImmediate(usesImmC),
 .immediate(immC),
 .isSyscall(isSysC),
 .isMem(isMemC),
 .isMultDiv(isMDC),
 .isCF(isCFC),
 .isALU(isALUC),
 .PC(PCC),
 .fullFlush(1'b0),
 .target(targetC)
);
decode decodeD(
 .instr(InstrD),
 .operation(opD1),
 .operation2(opD2),
 .queue(queueD),
 .rs(rsD),
 .rt(rtD),
 .rd(rdD1),
 .rd2(rdD2),
 .usesImmediate(usesImmD),
 .immediate(immD),
 .isSyscall(isSysD),
 .isMem(isMemD),
 .isMultDiv(isMDD),
 .isCF(isCFD),
 .isALU(isALUD),
 .PC(PCD),
 .fullFlush(1'b0),
 .target(targetD)
);

wire [5:0] free0,free1,free2,free3;
wire [7:0] numFree;

reg [203:0] remapped;

always @(*)begin
        remapped[5:0] = MAP[0];
        remapped[11:6] = MAP[1];
        remapped[17:12] = MAP[2];
        remapped[23:18] = MAP[3];
        remapped[29:24] = MAP[4];
        remapped[35:30] = MAP[5];
        remapped[41:36] = MAP[6];
        remapped[47:42] = MAP[7];
        remapped[53:48] = MAP[8];
        remapped[59:54] = MAP[9];
        remapped[65:60] = MAP[10];
        remapped[71:66] = MAP[11];
        remapped[77:72] = MAP[12];
        remapped[83:78] = MAP[13];
        remapped[89:84] = MAP[14];
        remapped[95:90] = MAP[15];
        remapped[101:96] = MAP[16];
        remapped[107:102] = MAP[17];
        remapped[113:108] = MAP[18];
        remapped[119:114] = MAP[19];
        remapped[125:120] = MAP[20];
        remapped[131:126] = MAP[21];
        remapped[137:132] = MAP[22];
        remapped[143:138] = MAP[23];
        remapped[149:144] = MAP[24];
        remapped[155:150] = MAP[25];
        remapped[161:156] = MAP[26];
        remapped[167:162] = MAP[27];
        remapped[173:168] = MAP[28];
        remapped[179:174] = MAP[29];
        remapped[185:180] = MAP[30];
        remapped[191:186] = MAP[31];
        remapped[197:192] = MAP[32];
        remapped[203:198] = MAP[33];
end

rename regRename(
 .RESET(RESET),
 .CLK(CLK),
 .fullFlush(fullFlush),
 .iStalled(Istalled),
 .numInstr(numInstr),
 .replacementMAP(remapped),
 .numFree(numFree),
 .renameStall(renameStall),
 .numRenamed(numRenamed),
 .inRS0(d_rsA),
 .inRT0(d_rtA),
 .inRD0(d_rdA),
 .outRS0(r_rsA),
 .outRT0(r_rtA),
 .outRD0(r_rdA),
 .free0(free0),
 .inRS1(d_rsB),
 .inRT1(d_rtB),
 .inRD1(d_rdB),
 .outRS1(r_rsB),
 .outRT1(r_rtB),
 .outRD1(r_rdB),
 .free1(free1),
 .inRS2(d_rsC),
 .inRT2(d_rtC),
 .inRD2(d_rdC),
 .outRS2(r_rsC),
 .outRT2(r_rtC),
 .outRD2(r_rdC),
 .free2(free2),
 .inRS3(d_rsD),
 .inRT3(d_rtD),
 .inRD3(d_rdD),
 .outRS3(r_rsD),
 .outRT3(r_rtD),
 .outRD3(r_rdD),
 .free3(free3),
 .ab_swap(ab_swap),
 .bc_swap(bc_swap),
 .cd_swap(cd_swap),
 .ab_dependant(ab_dependant),
 .bc_dependant(bc_dependant),
 .cd_dependant(cd_dependant)
);
reg [5:0] ROB_free0,ROB_free1,ROB_free2,ROB_free3;
reg [2:0] numAddFree;
freelist free_bird(
 .RESET(RESET),
 .CLK(CLK),
 .numRequest(numRenamed),
 .numFree(numFree),
 .free0(free0),
 .free1(free1),
 .free2(free2),
 .free3(free3),
 .numAdd(numAddFree),
 .nowFree0(ROB_free0),
 .nowFree1(ROB_free1),
 .nowFree2(ROB_free2),
 .nowFree3(ROB_free3)
);

reg updatePredictor_WE;
reg [31:0]updatePredictorPC;
reg updatePrediction;

branch_predictor br_predictor(
 .RESET(RESET),
 .CLK(CLK),
 .PC(CF_PC),
 .Prediction(BR_Pred),
 .UpdateEnable(updatePredictor_WE),
 .UpdatePC(updatePredictorPC),
 .UpdateValue(updatePrediction),
 .PredictorSelect(`select_pred_tpred),
 .Direction(br_dir)
);
reg [2:0] incorrect_pushes,incorrect_pops;
return_address_stack ras(
 .RESET(RESET),
 .CLK(CLK),
 .push(push_link),
 .pop(pop_link),
 .LinkPC(Link_PC),
 .PredPC(RAS_PC),
 .incorrect_pushes(incorrect_pushes),
 .incorrect_pops(incorrect_pops)
);
//Reorder Buffer / Ready Register list
wire [63:0] readyList;
reg [5:0] store_reg;

reg link_WE;
reg [5:0] link_reg;
reg [31:0] link_value;
reg [63:0] link_ready;
reg load_WE;
reg [5:0] load_reg;
reg [31:0] load_value;
reg [63:0] load_ready;

reg [4:0] ROB_Head;//Pointer to head of ROB
reg [4:0] ROB_Tail;
reg [5:0] ROB_Free;//Number of free entries

assign ROB_Full = ROB_Free < 4;
reg ROB_Ready[31:0];//Is the entry ready to commit
reg ROB_Flushed[31:0];//Is the entry flushed
reg [7:0] ROB_Op[31:0];//Operation the entry performs
reg [5:0] ROB_LogReg[31:0];//Logical destination
reg [5:0] ROB_PhyReg[31:0];//Physical destination
reg ROB_Pred[31:0];//Direction Prediction
reg [31:0] ROB_Pred_Target[31:0];//Predicted target
reg ROB_Dir[31:0];//Actual direction
reg [31:0] ROB_EffAddr_Target_PC8[31:0];//Effective address for mem, actual target for jr's, or
pc+8 of br
reg ROB_isSys[31:0];
reg ROB_isMem[31:0];
reg [2:0] ROB_isCF[31:0];
reg [31:0] ROB_debug_instr[31:0];
reg [31:0] ROB_debug_PC[31:0];
reg [2:0] numRetired;
reg [63:0] ROB_unready0,ROB_unready1,ROB_unready2,ROB_unready3;
reg [5:0] ROB_pr0,ROB_pr1,ROB_pr2,ROB_pr3,ROB_lr0,ROB_lr1,ROB_lr2,ROB_lr3;
reg [4:0] ROB_slot0,ROB_slot1,ROB_slot2,ROB_slot3;
reg retireDone,prem;
reg ROB_Dep[31:0];

reg didAStore,didALoad;
wire [31:0] ALU0_out,ALU1_out,EffAddr,BRJR_target;
wire BRJR_taken;
reg ALU0_WE,ALU1_WE;
reg [5:0] ex_rdA,ex_rdB;
reg [31:0] exv_rdA,exv_rdB;
reg LDST_Done,BRJR_Done;
reg [4:0] ex_ROBA,ex_ROBB,ex_ROBC,ex_ROBD;
reg ex_BRJR_taken;
reg [31:0] ex_EffAddr,ex_BRJR_target;

integer i,j;

reg [5:0] old_load_reg, delayed_load_reg;
reg DStalled;
reg loadDone,ugh;

always @(posedge CLK)begin
        if(RESET)begin
                load_ready = 0;
                ugh = 0;
        end
        else begin
                if(!DStall && loadDone == 0)begin
                        loadDone = 1;
                        load_ready = 1<<load_reg;
                end
```

```
               else load_ready = 0;

        end
        DStalled = DStall;
        old_load_reg = load_reg;
end

wire badnews;
assign badnews = ROB_Free[5]==1;
reg gotFlushed,bingo,doingSyscall;
reg [3:0] oneCycleDelay;

reg [5:0] LDSTQ_Addr_Reg [31:0];
reg [5:0] LDSTQ_Data_Reg [31:0];
reg [31:0] LDSTQ_Addr [31:0];
reg [31:0] LDSTQ_Data [31:0];
reg LDSTQ_hasAddr [31:0];
reg LDSTQ_Valid [31:0];
reg LDSTQ_isStore [31:0];
reg LDSTQ_isLoad [31:0];
reg LDSTQ_hasData [31:0];
reg [4:0] LDSTQ_ROB [31:0];
reg [4:0] LDSTQ_nextFree;
reg [2:0] LDSTQ_Size [31:0];
reg LDSTQ_wroteValue [31:0];
reg [31:0] pre_load;
reg    pre_load_WE;
reg stop_forwarding;
reg data_snooped;
reg [4:0] data_snooper;
reg unresolved_store;
reg earlyaccess;
reg [31:0] LDSTQ_max_fullness,LDSTQ_total_occupancy,ROB_total_occupancy;
reg [31:0] num_BR_predictions,num_JR_predictions,num_BR_incorrect,num_JR_incorrect;
reg [31:0] num_mem,num_cf,num_sys,num_alu_other;
reg [31:0] num_times_retire[4:0];
reg [31:0] num_times_rename[4:0];
reg [31:0] num_d_accesses;

//ROB and LDSTQ (the one for cache access not address calc)
always @(negedge CLK)begin
        //Insert new instr's into ROB
        if(RESET)begin
               ROB_Tail = 0;
               ROB_Head = 0;
               ROB_Ready[0] = 0;
               ROB_Free = 32;
               fullFlush = 0;
               isMemRetire = 0;
               ROB_slot0 = 0;
               ROB_slot1 = 1;
               ROB_slot2 = 2;
               ROB_slot3 = 3;
               ROB_unready3 = 0;
               ROB_unready2 = 0;
               ROB_unready1 = 0;
               ROB_unready0 = 0;
               ROB_free0 = 0;
               ROB_free1 = 0;
               ROB_free2 = 0;
               ROB_free3 = 0;
               doingSyscall = 0;
               num_instructions_retired = 0;
               num_instructions_inserted = 0;
               num_flushes = 0;
               updatePredictor_WE = 0;
               updatePredictorPC = 0;
               updatePrediction = 0;
               oneCycleDelay = 1;
               loadDone = 1;
               for(i=0;i<34;i=i+1)MAP[i]=i;
```

```
				LDSTQ_nextFree = 0;
				for(i=0;i<32;i=i+1)LDSTQ_Valid[i] = 0;
				pre_load = 0;
				pre_load_WE = 0;
				stop_forwarding = 0;
				data_snooped = 0;
				data_snooper = 0;
				unresolved_store = 0;
				num_data_forwarded=0;
				num_data_forwarded_written_ahead_of_time=0;
				num_early_load_accesses=0;
				num_data_forwarded_not_written=0;
				num_loads_written_before_retire=0;
				num_rob_head_loads=0;
				LDSTQ_max_fullness=0;
				LDSTQ_total_occupancy=0;
				ROB_total_occupancy = 0;
				num_BR_predictions=0;
				num_JR_predictions=0;
				num_BR_incorrect=0;
				num_JR_incorrect=0;
				num_mem = 0;
				num_cf = 0;
				num_sys = 0;
				num_alu_other = 0;
				num_times_retire[0]=0;
				num_times_retire[1]=0;
				num_times_retire[2]=0;
				num_times_retire[3]=0;
				num_times_retire[4]=0;
				num_times_rename[0]=0;
				num_times_rename[1]=0;
				num_times_rename[2]=0;
				num_times_rename[3]=0;
				num_times_rename[4]=0;
				num_d_accesses = 0;
				incorrect_pushes = 0;
				incorrect_pops = 0;
		end
		else begin
				if(r_numInstr>=1)begin
						//ROB
						if(r_queueA == 0 || r_isMemA)ROB_Ready[ROB_Tail] = 1;//Is the entry ready
to commit
						else ROB_Ready[ROB_Tail] = 0;
						ROB_Flushed[ROB_Tail] = fullFlush;//Is the entry flushed
						ROB_Op[ROB_Tail] = r_opA;//Operation the entry performs
						if(r_isMemA && (r_opA==`select_mem_sw || r_opA==`select_mem_sh ||
r_opA==`select_mem_sb))begin
								ROB_PhyReg[ROB_Tail] = r_rtA;//Physical destination
								ROB_LogReg[ROB_Tail] = 0;
						end
						else begin
								ROB_PhyReg[ROB_Tail] = r_rdA;//Physical destination
								ROB_LogReg[ROB_Tail] = r_lrdA;//Logical destination
						end
						ROB_Pred[ROB_Tail] = r_pred_A;//Direction Prediction
						ROB_Pred_Target[ROB_Tail] = r_targetA;//Predicted target
						ROB_Dir[ROB_Tail] = r_pred_A;//Actual direction
						ROB_debug_PC[ROB_Tail] = r_PCA;
						ROB_EffAddr_Target_PC8[ROB_Tail] = r_PCA+8;//Effective address for mem,
actual target for jr's, or pc+8 of br
						ROB_isSys[ROB_Tail] = r_isSysA;
						ROB_isMem[ROB_Tail] = r_isMemA;
						ROB_isCF[ROB_Tail] = r_isCFA;
						ROB_Dep[ROB_Tail] = r_adep;
						ROB_debug_instr[ROB_Tail] = r_instrA;
						ROB_Tail = ROB_Tail + 1;
						ROB_Free = ROB_Free - 1; num_instructions_inserted =
num_instructions_inserted + 1;
```

```
                        //LDSTQ
                        if(r_isMemA)begin
                                LDSTQ_Addr_Reg[LDSTQ_nextFree] = r_rsA;
                                if((r_opA==`select_mem_sw || r_opA==`select_mem_sh ||
r_opA==`select_mem_sb))begin
                                        LDSTQ_Data_Reg[LDSTQ_nextFree] = r_rtA;
                                        LDSTQ_isStore[LDSTQ_nextFree] = 1;
                                        LDSTQ_isLoad[LDSTQ_nextFree] = 0;
                                        if(r_opA==`select_mem_sw)begin
                                                LDSTQ_Size[LDSTQ_nextFree] = 4;
                                        end
                                        else if(r_opA==`select_mem_sh)begin
                                                LDSTQ_Size[LDSTQ_nextFree] = 2;
                                        end
                                        else begin
                                                LDSTQ_Size[LDSTQ_nextFree] = 1;
                                        end
                                end
                                else begin
                                        LDSTQ_Data_Reg[LDSTQ_nextFree] = r_rdA;
                                        LDSTQ_isStore[LDSTQ_nextFree] = 0;
                                        LDSTQ_isLoad[LDSTQ_nextFree] = 1;
                                        if(r_opA==`select_mem_lw)begin
                                                LDSTQ_Size[LDSTQ_nextFree] = 4;
                                        end
                                        else if(r_opA==`select_mem_lh||r_opA==`select_mem_lhu)begin
                                                LDSTQ_Size[LDSTQ_nextFree] = 2;
                                        end
                                        else begin
                                                LDSTQ_Size[LDSTQ_nextFree] = 1;
                                        end
                                end
                                LDSTQ_wroteValue[LDSTQ_nextFree] = 0;
                                LDSTQ_hasAddr[LDSTQ_nextFree] = 0;
                                LDSTQ_Valid[LDSTQ_nextFree] = 1;
                                LDSTQ_hasData[LDSTQ_nextFree] = 0;
                                LDSTQ_ROB[LDSTQ_nextFree] = ROB_Tail - 1;
                                LDSTQ_nextFree = LDSTQ_nextFree + 1;
                        end
                end
                if(r_numInstr>=2)begin
                        //ROB
                        if(r_queueB == 0 || r_isMemB)ROB_Ready[ROB_Tail] = 1;//Is the entry ready
to commit
                        else ROB_Ready[ROB_Tail] = 0;
                        ROB_Flushed[ROB_Tail] = fullFlush;//Is the entry flushed
                        ROB_Op[ROB_Tail] = r_opB;//Operation the entry performs
                        if(r_isMemB && (r_opB==`select_mem_sw || r_opB==`select_mem_sh ||
r_opB==`select_mem_sb))begin
                                ROB_PhyReg[ROB_Tail] = r_rtB;//Physical destination
                                ROB_LogReg[ROB_Tail] = 0;
                        end
                        else begin
                                ROB_PhyReg[ROB_Tail] = r_rdB;//Physical destination
                                ROB_LogReg[ROB_Tail] = r_lrdB;//Logical destination
                        end
                        ROB_Pred[ROB_Tail] = r_pred_B;//Direction Prediction
                        ROB_Pred_Target[ROB_Tail] = r_targetB;//Predicted target
                        ROB_Dir[ROB_Tail] = r_pred_B;//Actual direction
                        ROB_debug_PC[ROB_Tail] = r_PCB;
                        ROB_EffAddr_Target_PC8[ROB_Tail] = r_PCB+8;//Effective address for mem,
actual target for jr's, or pc+8 of br
                        ROB_isSys[ROB_Tail] = r_isSysB;
                        ROB_isMem[ROB_Tail] = r_isMemB;
                        ROB_isCF[ROB_Tail] = r_isCFB;
                        ROB_Dep[ROB_Tail] = r_bdep;
                        ROB_debug_instr[ROB_Tail] = r_instrB;
                        ROB_Tail = ROB_Tail + 1;
                        ROB_Free = ROB_Free - 1; num_instructions_inserted =
num_instructions_inserted + 1;
```

```
                        //LDSTQ
                        if(r_isMemB)begin
                                LDSTQ_Addr_Reg[LDSTQ_nextFree] = r_rsB;
                                if((r_opB==`select_mem_sw || r_opB==`select_mem_sh ||
r_opB==`select_mem_sb))begin
                                        LDSTQ_Data_Reg[LDSTQ_nextFree] = r_rtB;
                                        LDSTQ_isStore[LDSTQ_nextFree] = 1;
                                        LDSTQ_isLoad[LDSTQ_nextFree] = 0;
                                        if(r_opB==`select_mem_sw)begin
                                                LDSTQ_Size[LDSTQ_nextFree] = 4;
                                        end
                                        else if(r_opB==`select_mem_sh)begin
                                                LDSTQ_Size[LDSTQ_nextFree] = 2;
                                        end
                                        else begin
                                                LDSTQ_Size[LDSTQ_nextFree] = 1;
                                        end
                                end
                                else begin
                                        LDSTQ_Data_Reg[LDSTQ_nextFree] = r_rdB;
                                        LDSTQ_isStore[LDSTQ_nextFree] = 0;
                                        LDSTQ_isLoad[LDSTQ_nextFree] = 1;
                                        if(r_opB==`select_mem_lw)begin
                                                LDSTQ_Size[LDSTQ_nextFree] = 4;
                                        end
                                        else if(r_opB==`select_mem_lh||r_opB==`select_mem_lhu)begin
                                                LDSTQ_Size[LDSTQ_nextFree] = 2;
                                        end
                                        else begin
                                                LDSTQ_Size[LDSTQ_nextFree] = 1;
                                        end
                                end
                                LDSTQ_wroteValue[LDSTQ_nextFree] = 0;
                                LDSTQ_hasAddr[LDSTQ_nextFree] = 0;
                                LDSTQ_Valid[LDSTQ_nextFree] = 1;
                                LDSTQ_hasData[LDSTQ_nextFree] = 0;
                                LDSTQ_ROB[LDSTQ_nextFree] = ROB_Tail - 1;
                                LDSTQ_nextFree = LDSTQ_nextFree + 1;
                        end
                end
                if(r_numInstr>=3)begin
                        //ROB
                        if(r_queueC == 0 || r_isMemC)ROB_Ready[ROB_Tail] = 1;//Is the entry ready
to commit
                        else ROB_Ready[ROB_Tail] = 0;
                        ROB_Flushed[ROB_Tail] = fullFlush;//Is the entry flushed
                        ROB_Op[ROB_Tail] = r_opC;//Operation the entry performs
                        if(r_isMemC && (r_opC==`select_mem_sw || r_opC==`select_mem_sh ||
r_opC==`select_mem_sb))begin
                                ROB_PhyReg[ROB_Tail] = r_rtC;//Physical destination
                                ROB_LogReg[ROB_Tail] = 0;
                        end
                        else begin
                                ROB_PhyReg[ROB_Tail] = r_rdC;//Physical destination
                                ROB_LogReg[ROB_Tail] = r_lrdC;//Logical destination
                        end
                        ROB_Pred[ROB_Tail] = r_pred_C;//Direction Prediction
                        ROB_Pred_Target[ROB_Tail] = r_targetC;//Predicted target
                        ROB_Dir[ROB_Tail] = r_pred_C;//Actual direction
                        ROB_debug_PC[ROB_Tail] = r_PCC;
                        ROB_EffAddr_Target_PC8[ROB_Tail] = r_PCC+8;//Effective address for mem,
actual target for jr's, or pc+8 of br
                        ROB_isSys[ROB_Tail] = r_isSysC;
                        ROB_isMem[ROB_Tail] = r_isMemC;
                        ROB_isCF[ROB_Tail] = r_isCFC;
                        ROB_Dep[ROB_Tail] = r_cdep;
                        ROB_debug_instr[ROB_Tail] = r_instrC;
                        ROB_Tail = ROB_Tail + 1;
                        ROB_Free = ROB_Free - 1; num_instructions_inserted =
num_instructions_inserted + 1;
```

```
            //LDSTQ
            if(r_isMemC)begin
                LDSTQ_Addr_Reg[LDSTQ_nextFree] = r_rsC;
                if((r_opC==`select_mem_sw || r_opC==`select_mem_sh ||
r_opC==`select_mem_sb))begin
                    LDSTQ_Data_Reg[LDSTQ_nextFree] = r_rtC;
                    LDSTQ_isStore[LDSTQ_nextFree] = 1;
                    LDSTQ_isLoad[LDSTQ_nextFree] = 0;
                    if(r_opC==`select_mem_sw)begin
                        LDSTQ_Size[LDSTQ_nextFree] = 4;
                    end
                    else if(r_opC==`select_mem_sh)begin
                        LDSTQ_Size[LDSTQ_nextFree] = 2;
                    end
                    else begin
                        LDSTQ_Size[LDSTQ_nextFree] = 1;
                    end
                end
                else begin
                    LDSTQ_Data_Reg[LDSTQ_nextFree] = r_rdC;
                    LDSTQ_isStore[LDSTQ_nextFree] = 0;
                    LDSTQ_isLoad[LDSTQ_nextFree] = 1;
                    if(r_opC==`select_mem_lw)begin
                        LDSTQ_Size[LDSTQ_nextFree] = 4;
                    end
                    else if(r_opC==`select_mem_lh||r_opC==`select_mem_lhu)begin
                        LDSTQ_Size[LDSTQ_nextFree] = 2;
                    end
                    else begin
                        LDSTQ_Size[LDSTQ_nextFree] = 1;
                    end
                end
                LDSTQ_wroteValue[LDSTQ_nextFree] = 0;
                LDSTQ_hasAddr[LDSTQ_nextFree] = 0;
                LDSTQ_Valid[LDSTQ_nextFree] = 1;
                LDSTQ_hasData[LDSTQ_nextFree] = 0;
                LDSTQ_ROB[LDSTQ_nextFree] = ROB_Tail - 1;
                LDSTQ_nextFree = LDSTQ_nextFree + 1;
            end
        end
        if(r_numInstr>=4)begin
            //ROB
            if(r_queueD == 0 || r_isMemD)ROB_Ready[ROB_Tail] = 1;//Is the entry ready
to commit
            else ROB_Ready[ROB_Tail] = 0;
            ROB_Flushed[ROB_Tail] = fullFlush;//Is the entry flushed
            ROB_Op[ROB_Tail] = r_opD;//Operation the entry performs
            if(r_isMemD && (r_opD==`select_mem_sw || r_opD==`select_mem_sh ||
r_opD==`select_mem_sb))begin
                ROB_PhyReg[ROB_Tail] = r_rtD;//Physical destination
                ROB_LogReg[ROB_Tail] = 0;
            end
            else begin
                ROB_PhyReg[ROB_Tail] = r_rdD;//Physical destination
                ROB_LogReg[ROB_Tail] = r_lrdD;//Logical destination
            end
            ROB_Pred[ROB_Tail] = r_pred_D;//Direction Prediction
            ROB_Pred_Target[ROB_Tail] = r_targetD;//Predicted target
            ROB_Dir[ROB_Tail] = r_pred_D;//Actual direction
            ROB_debug_PC[ROB_Tail] = r_PCD;
            ROB_EffAddr_Target_PC8[ROB_Tail] = r_PCD+8;//Effective address for mem,
actual target for jr's, or pc+8 of br
            ROB_isSys[ROB_Tail] = r_isSysD;
            ROB_isMem[ROB_Tail] = r_isMemD;
            ROB_isCF[ROB_Tail] = r_isCFD;
            ROB_Dep[ROB_Tail] = r_ddep;
            ROB_debug_instr[ROB_Tail] = r_instrD;
            ROB_Tail = ROB_Tail + 1;
            ROB_Free = ROB_Free - 1; num_instructions_inserted =
num_instructions_inserted + 1;
```

```
                //LDSTQ
                if(r_isMemD)begin
                        LDSTQ_Addr_Reg[LDSTQ_nextFree] = r_rsD;
                        if((r_opD==`select_mem_sw || r_opD==`select_mem_sh ||
r_opD==`select_mem_sb))begin
                                LDSTQ_Data_Reg[LDSTQ_nextFree] = r_rtD;
                                LDSTQ_isStore[LDSTQ_nextFree] = 1;
                                LDSTQ_isLoad[LDSTQ_nextFree] = 0;
                                if(r_opD==`select_mem_sw)begin
                                        LDSTQ_Size[LDSTQ_nextFree] = 4;
                                end
                                else if(r_opD==`select_mem_sh)begin
                                        LDSTQ_Size[LDSTQ_nextFree] = 2;
                                end
                                else begin
                                        LDSTQ_Size[LDSTQ_nextFree] = 1;
                                end
                        end
                        else begin
                                LDSTQ_Data_Reg[LDSTQ_nextFree] = r_rdD;
                                LDSTQ_isStore[LDSTQ_nextFree] = 0;
                                LDSTQ_isLoad[LDSTQ_nextFree] = 1;
                                if(r_opD==`select_mem_lw)begin
                                        LDSTQ_Size[LDSTQ_nextFree] = 4;
                                end
                                else if(r_opD==`select_mem_lh||r_opD==`select_mem_lhu)begin
                                        LDSTQ_Size[LDSTQ_nextFree] = 2;
                                end
                                else begin
                                        LDSTQ_Size[LDSTQ_nextFree] = 1;
                                end
                        end
                        LDSTQ_wroteValue[LDSTQ_nextFree] = 0;
                        LDSTQ_hasAddr[LDSTQ_nextFree] = 0;
                        LDSTQ_Valid[LDSTQ_nextFree] = 1;
                        LDSTQ_hasData[LDSTQ_nextFree] = 0;
                        LDSTQ_ROB[LDSTQ_nextFree] = ROB_Tail - 1;
                        LDSTQ_nextFree = LDSTQ_nextFree + 1;
                end
        end
        num_times_rename[r_numInstr]=num_times_rename[r_numInstr]+1;
        ROB_slot0 = ROB_Tail;
        ROB_slot1 = ROB_Tail+1;
        ROB_slot2 = ROB_Tail+2;
        ROB_slot3 = ROB_Tail+3;
        //Update Entries in the ROB
        if(!fullFlush && !gotFlushed)begin
                if(ALU0_WE && ROB_Flushed[ex_ROBA]==0)begin
                        ROB_Ready[ex_ROBA]=1;
                end
                if(ALU1_WE && ROB_Flushed[ex_ROBB]==0)begin
                        ROB_Ready[ex_ROBB]=1;
                end
                if(LDST_Done && ROB_Flushed[ex_ROBC]==0)begin
                        //ROB_Ready[ex_ROBC]=1;
                        //ROB_EffAddr_Target_PC8[ex_ROBC] = ex_EffAddr;
                end
                if(BRJR_Done && ROB_Flushed[ex_ROBD]==0)begin
                        ROB_Ready[ex_ROBD]=1;
                        ROB_Dir[ex_ROBD] = ex_BRJR_taken;
                        if(ex_BRJR_taken)ROB_EffAddr_Target_PC8[ex_ROBD] = ex_BRJR_target;
                end
        end
        LDSTQ_total_occupancy = LDSTQ_total_occupancy + LDSTQ_nextFree;
        ROB_total_occupancy = ROB_total_occupancy + (32-ROB_Free);
        if(LDSTQ_nextFree>LDSTQ_max_fullness)LDSTQ_max_fullness = LDSTQ_nextFree;
        //LDSTQ
        //Snoop in Address
        for(i=0;i<32;i=i+1)begin
                if(LDSTQ_Valid[i] && LDSTQ_ROB[i]==ex_ROBC && LDST_Done)begin
                        LDSTQ_hasAddr[i] = 1;
                        LDSTQ_Addr[i] = ex_EffAddr;
                end
        end
        //Snoop in data
        if(data_snooped == 1)begin
                LDSTQ_Data[data_snooper] = store_value;
                if(LDSTQ_Valid[data_snooper])LDSTQ_hasData[data_snooper] = 1;
                data_snooped = 0;
        end
        for(i=0;i<32;i=i+1)begin
                if(data_snooped == 0 && LDSTQ_Valid[i]==1 && LDSTQ_hasData[i]==0 &&
LDSTQ_isStore[i] == 1 && (readyList>>LDSTQ_Data_Reg[i])&1'b1)begin
                        store_reg = LDSTQ_Data_Reg[i];
                        data_snooped = 1;
                        data_snooper = i;
                end
        end
        stop_forwarding = 0;
        //Forward Data, perhaps too conservatively
        for(i=0;i<32;i=i+1)begin
                if(LDSTQ_isStore[i] && LDSTQ_Valid[i] && LDSTQ_hasAddr[i] &&
LDSTQ_hasData[i])begin
                        stop_forwarding = 0;
                        for(j=i+1;j<32;j=j+1)begin
                                if(LDSTQ_isStore[j] &&
(((LDSTQ_Addr[j]&32'hFFFFFFFC)==(LDSTQ_Addr[i]&32'hFFFFFFFC)) || LDSTQ_hasAddr[j]==0))begin
                                        stop_forwarding = 1;
                                end
                                if(!stop_forwarding && LDSTQ_isLoad[j] && LDSTQ_Valid[j] &&
LDSTQ_hasAddr[j] && !LDSTQ_hasData[j] && LDSTQ_Addr[i]==LDSTQ_Addr[j] &&
LDSTQ_Size[i]==LDSTQ_Size[j])begin
                                        LDSTQ_hasData[j] = 1;
                                        num_data_forwarded = num_data_forwarded + 1;
                                        if(LDSTQ_Size[j]==4)LDSTQ_Data[j] = LDSTQ_Data[i];
                                        else if(LDSTQ_Size[j]==2)begin
                                                LDSTQ_Data[j] = LDSTQ_Data[i]&32'h0000ffff;
                                                if(ROB_Op[LDSTQ_ROB[j]]==`select_mem_lh &&
LDSTQ_Data[j][15])LDSTQ_Data[j] = LDSTQ_Data[j] | 32'hffff0000;
                                        end
                                        else begin
                                                LDSTQ_Data[j] = LDSTQ_Data[i]&32'h000000ff;
                                                if(ROB_Op[LDSTQ_ROB[j]]==`select_mem_lb&&
LDSTQ_Data[j][7])LDSTQ_Data[j] = LDSTQ_Data[j] | 32'hffffff00;
                                        end
                                end
                        end
                end
        end

        pre_load = 0;
        pre_load_WE = 0;


        //Retire Instructions at ROB head
        retireDone = 0;
        prem = 0;
        if(DStall || !loadDone || (ROB_Head==ROB_Tail&&ROB_Free!=0) ||
(ROB_Dep[ROB_Head]&&!ROB_Ready[(ROB_Head+1)&5'b11111]&&!ROB_Flushed[ROB_Head]))begin
                retireDone = 1;
                prem = 1;
        end
        numRetired = 0;//for retirement rat
        numAddFree = 0;//for free list
        didALoad = 0;
        if(!DStall)isMemRetire = 0;

        if(!DStall)load_reg = 0;
        //store_reg = 0;
        ROB_unready3 = 0;
        ROB_unready2 = 0;
        ROB_unready1 = 0;
```

```
                ROB_unready0 = 0;
                ROB_free0 = 0;
                ROB_free1 = 0;
                ROB_free2 = 0;
                ROB_free3 = 0;
                doingSyscall = 0;
                updatePredictor_WE = 0;
                incorrect_pushes = 0;
                incorrect_pops = 0;
                if(!IStall && !ROB_Full/* && !Istalled*/)fullFlush = 0;
                //retire up to 4 instructions per cycle
                if(retireDone != 1 && (ROB_Ready[ROB_Head] == 1 || ROB_Flushed[ROB_Head] ==
1))begin
                        if(ROB_isSys[ROB_Head] && ROB_Flushed[ROB_Head] == 0)begin
                        if(oneCycleDelay!=0)begin
                                oneCycleDelay = oneCycleDelay - 1;
                                retireDone = 1;
                        end
                        else begin
                                num_sys = num_sys + 1;
                                oneCycleDelay = 1;
                                num_instructions_retired = num_instructions_retired  + 1;numRetired
= numRetired + 1;
                                /**retireDone = 1;**/doingSyscall = 1;
                                num_flushes = num_flushes + 1;fullFlush =
1;for(i=0;i<32;i=i+1)ROB_Flushed[i] = 1;
                                correctTarget = ROB_debug_PC[ROB_Head]+4;

                                ROB_Ready[ROB_Head] = 0;ROB_Flushed[ROB_Head] = 0;
                                ROB_LogReg[ROB_Head] =  0;ROB_PhyReg[ROB_Head] = 0;ROB_Head =
ROB_Head + 1;
                                ROB_free0 = 0;
                                            begin
                                               // store register file state to the C
interface functions!
                                               for (j=1; j<32;j=j+1)
                                                        begin
                                                           // Changed to setreg instead of
setregvalue
                                                           $syscall_setreg
(j,CPU.reg_file.RegFile[CPU.MAP[j]]);
                                                         end
                                               $syscall_setpc (ROB_debug_PC[ROB_Head]);
        // store PC

                                               // do the system call
                                               if ($emulate_syscall) begin
                                                         // exit called
                                                         $display ("-------------------------
----------");

                                                         $display ("Program Terminated.");
                                                         $display ("Retired %d instructions,
%d entered ROB",num_instructions_retired,num_instructions_inserted);
                                                         $display ("Number of flushes:
%d",num_flushes);
                                                         $display ("Ran for %d
cycles",$time/`cycle);
                                                         $display ("ICache Accesses: %d
\nICache Misses: %d",  i_cache_d_cache.num_icache_accesses,numIMisses);
                                                         $display ("DCache Accesses: %d
\nDCache Misses: %d",  num_d_accesses,numDMisses);
                                                         $display ("Average Fullness: \nROB
%d \nALUQ %d \nCFQ %d \nLDSTAQ %d \nLDSTQ %d \nFreelist(avg. free regs)
%d",(10*ROB_total_occupancy)/($time/`cycle),(10*ALUqueue.total_occupancy)/($time/`cycle),(10*BRJR
queue.total_occupancy)/($time/`cycle),(10*LDSTqueue.total_occupancy)/($time/`cycle),(10*LDSTQ_tot
al_occupancy)/($time/`cycle),(10*free_bird.numFree)/($time/`cycle));
                                                         $display ("RAS
%d",(10*ras.total_occupancy)/($time/`cycle));
                                                         $display ("Max Fullness: \nALUQ %d
\nCFQ %d \nLDSTAQ %d \nLDSTQ %d\nRAS
%d",ALUqueue.max_fullness,BRJRqueue.max_fullness,LDSTqueue.max_fullness,LDSTQ_max_fullness,ras.ma
x_fullness);
                                                             $display ("BR Predictions: %d
\nIncorrect: %d\nJR Predictions: %d \nIncorrect:
%d",num_BR_predictions,num_BR_incorrect,num_JR_predictions,num_JR_incorrect);
                                                             $display ("Correct Path: \nMem %d
\nCF %d \nSys %d \nALU & NOPS %d",num_mem,num_cf,num_sys,num_instructions_retired-
(num_mem+num_cf+num_sys));
                                                             $display ("Retired 0 1 2 3 4: %d %d
%d %d
%d",num_times_retire[0],num_times_retire[1],num_times_retire[2],num_times_retire[3],num_times_ret
ire[4]);
                                                             $display ("Renamed 0 1 2 3 4: %d %d
%d %d
%d",num_times_rename[0],num_times_rename[1],num_times_rename[2],num_times_rename[3],num_times_ren
ame[4]);
                                                             $display ("Cycles Stalled: \nIStall
%d \nRename %d \nROB %d",num_cycles_istall,num_cycles_renamestall,num_cycles_robstall);
                                                             $display ("Forwarded Data: %d\nWrote
Forwards Early: %d\nPerformed Early Loads: %d\nForwarded Data But No Early Write: %d\nROB Head
Loads: %d\nLoads Already Written Before Retire:
%d",num_data_forwarded,num_data_forwarded_written_ahead_of_time,num_early_load_accesses,num_data_
forwarded_not_written,num_rob_head_loads,num_loads_written_before_retire);
                                                             $display ("------------------------
----------");


                                                             $finish;
                                                    end

                                                    // restore register file state from the C
interface
                                                    for (j=1; j<32;j=j+1)
                                                             begin
                                                                // Changed to syscall_getreg
instead of syscall_regvalue

       CPU.reg_file.RegFile[CPU.MAP[j]] = $syscall_getreg(j);
                                                             end
                                                 end
                              end

                       end
                       else if(ROB_isCF[ROB_Head][0] && ROB_Flushed[ROB_Head] == 0)begin
                              if(/**!IStall &&**/ updatePredictor_WE==0)begin
                                     num_cf = num_cf + 1;
                                     if(ROB_Op[ROB_Head]==`select_qc_j ||
ROB_Op[ROB_Head]==`select_qc_jal)begin
                                            //regular jumps cant be wrong
                                     end
                                     else if(ROB_Op[ROB_Head]==`select_qc_jalr ||
ROB_Op[ROB_Head]==`select_qc_jr)begin
                                            num_JR_predictions = num_JR_predictions + 1;

       if(ROB_Pred_Target[ROB_Head]!=ROB_EffAddr_Target_PC8[ROB_Head])begin
                                                   num_JR_incorrect = num_JR_incorrect + 1;
                                                   correctTarget =
ROB_EffAddr_Target_PC8[ROB_Head];
                                                   num_flushes = num_flushes + 1;fullFlush =
1;for(i=0;i<32;i=i+1)ROB_Flushed[i] = 1;
                                                   /**retireDone = 1;**/
                                            end
                                     end
                                     else begin
                                            if(ROB_Dir[ROB_Head]!=ROB_Pred[ROB_Head])begin
                                                   num_BR_incorrect = num_BR_incorrect + 1;
                                                   if(ROB_Dir[ROB_Head])correctTarget =
ROB_Pred_Target[ROB_Head];
                                                   else correctTarget =
ROB_EffAddr_Target_PC8[ROB_Head];
                                                   num_flushes = num_flushes + 1;fullFlush =
1;for(i=0;i<32;i=i+1)ROB_Flushed[i] = 1;
```

```
                                                        /**retireDone = 1;**/
                                                end
                                                num_BR_predictions = num_BR_predictions + 1;

                                                updatePredictor_WE = 1;
                                                updatePredictorPC = ROB_debug_PC[ROB_Head];
                                                updatePrediction = ROB_Dir[ROB_Head];
                                        end
                                        //link
                                        if(ROB_isCF[ROB_Head][2])begin
                                                ROB_free0 = MAP[ROB_LogReg[ROB_Head]];
                                                ROB_unready0 = 1<<MAP[ROB_LogReg[ROB_Head]];
                                                MAP[ROB_LogReg[ROB_Head]] = ROB_PhyReg[ROB_Head];
                                        end
                                        else ROB_free0 = 0;

                                        ROB_Ready[ROB_Head] = 0;ROB_Flushed[ROB_Head] = 0;
                                        ROB_LogReg[ROB_Head] =  0;ROB_PhyReg[ROB_Head] = 0;ROB_Head
= ROB_Head + 1;
                                        num_instructions_retired = num_instructions_retired  +
1;numRetired = numRetired + 1;

                                end
                                else begin
                                        retireDone = 1;
                                end
                        end
                        else if(ROB_isMem[ROB_Head] && ROB_Flushed[ROB_Head] == 0)begin
                                if(isMemRetire==0 && (((ROB_Op[ROB_Head]==`select_mem_sw ||
ROB_Op[ROB_Head]==`select_mem_sh ||
ROB_Op[ROB_Head]==`select_mem_sb)&&(LDSTQ_hasAddr[0]&&LDSTQ_hasData[0]))
                                ||(!(ROB_Op[ROB_Head]==`select_mem_sw ||
ROB_Op[ROB_Head]==`select_mem_sh ||
ROB_Op[ROB_Head]==`select_mem_sb)&&LDSTQ_hasAddr[0]&&(loadDone==1))))begin
                                        if(!(ROB_Op[ROB_Head]==`select_mem_sw ||
ROB_Op[ROB_Head]==`select_mem_sh || ROB_Op[ROB_Head]==`select_mem_sb))begin
                                        //load
                                                if(LDSTQ_hasData[0] && LDSTQ_wroteValue[0])begin
                                                        num_loads_written_before_retire =
num_loads_written_before_retire+1;

                                                end
                                                else if(LDSTQ_hasData[0])begin
                                                //got data through forwarding but didnt get to write
yet
                                                        pre_load = LDSTQ_Data[0];
                                                        load_reg = LDSTQ_Data_Reg[0];
                                                        pre_load_WE = 1;
                                                        loadDone = 0;
                                                        num_data_forwarded_not_written =
num_data_forwarded_not_written + 1;
                                                end
                                                else begin
                                                //doesnt have its data, need to issue cache access
                                                        isMemRetire = 1;
                                                        memOp = ROB_Op[ROB_Head];
                                                        MAR = LDSTQ_Addr[0];
                                                        load_reg = ROB_PhyReg[ROB_Head];
                                                        loadDone = 0;
                                                        num_rob_head_loads = num_rob_head_loads + 1;
                                                        num_d_accesses = num_d_accesses + 1;
                                                end
                                                ROB_free0 = MAP[ROB_LogReg[ROB_Head]];
                                                ROB_unready0 = 1<<MAP[ROB_LogReg[ROB_Head]];
                                                MAP[ROB_LogReg[ROB_Head]] = ROB_PhyReg[ROB_Head];
                                        end
                                        else begin
                                        //store
                                                SMDR = LDSTQ_Data[0];
                                                isMemRetire = 1;
                                                memOp = ROB_Op[ROB_Head];
                                                MAR = LDSTQ_Addr[0];
                                                ROB_free0 = 0;
                                                num_d_accesses = num_d_accesses + 1;
                                        end
                                        //shift LDSTQ forward
                                        for(i=0;i<31;i=i+1)begin
                                                LDSTQ_Addr_Reg[i] = LDSTQ_Addr_Reg[i+1];
                                                LDSTQ_Data_Reg[i] = LDSTQ_Data_Reg[i+1];
                                                LDSTQ_Addr[i] = LDSTQ_Addr[i+1];
                                                LDSTQ_Data[i] = LDSTQ_Data[i+1];
                                                LDSTQ_hasAddr[i] = LDSTQ_hasAddr[i+1];
                                                LDSTQ_Valid[i] = LDSTQ_Valid[i+1];
                                                LDSTQ_isStore[i] = LDSTQ_isStore[i+1];
                                                LDSTQ_isLoad[i] = LDSTQ_isLoad[i+1];
                                                LDSTQ_hasData[i] = LDSTQ_hasData[i+1];
                                                LDSTQ_ROB[i] = LDSTQ_ROB[i+1];
                                                LDSTQ_Size[i] = LDSTQ_Size[i+1];
                                                LDSTQ_wroteValue[i] = LDSTQ_wroteValue[i+1];
                                        end
                                        LDSTQ_Addr_Reg[31]=0;
                                        LDSTQ_Data_Reg[31]=0;
                                        LDSTQ_Addr[31]=0;
                                        LDSTQ_Data[31]=0;
                                        LDSTQ_hasAddr[31]=0;
                                        LDSTQ_Valid[31]=0;
                                        LDSTQ_isStore[31]=0;
                                        LDSTQ_isLoad[31]=0;
                                        LDSTQ_hasData[31]=0;
                                        LDSTQ_ROB[31]=0;
                                        LDSTQ_Size[31]=0;
                                        LDSTQ_wroteValue[31]=0;
                                        LDSTQ_nextFree = LDSTQ_nextFree - 1;
                                        data_snooper = data_snooper - 1;
                                        num_mem = num_mem + 1;
                                        num_instructions_retired = num_instructions_retired  + 
1;numRetired = numRetired + 1;

                                        /**retireDone = 1;**/
                                        ROB_Ready[ROB_Head] = 0;ROB_Flushed[ROB_Head] = 0;
                                        ROB_LogReg[ROB_Head] =  0;ROB_PhyReg[ROB_Head] = 0;ROB_Head 
= ROB_Head + 1;
                                end
                                else begin
                                        retireDone = 1;
                                        ROB_free0 = 0;
                                end
                        end
                        else begin
                                if(ROB_Flushed[ROB_Head]==0)num_instructions_retired = 
num_instructions_retired  + 1;
                                if(ROB_isCF[ROB_Head][0])begin

        if(ROB_isCF[ROB_Head][2]&&ROB_LogReg[ROB_Head]==31)incorrect_pushes = incorrect_pushes + 
1;
                                        if((ROB_Op[ROB_Head]==`select_qc_jalr || 
ROB_Op[ROB_Head]==`select_qc_jr)&&(ROB_debug_instr[ROB_Head][`rs]==31))incorrect_pops = 
incorrect_pops + 1;
                                end
                                numRetired = numRetired + 1;
                                if(ROB_Flushed[ROB_Head]==0)ROB_free0 = MAP[ROB_LogReg[ROB_Head]];

                                else if(ROB_LogReg[ROB_Head]!=0)ROB_free0 = ROB_PhyReg[ROB_Head];
                                if(ROB_Flushed[ROB_Head]==0)ROB_unready0 = 
1<<MAP[ROB_LogReg[ROB_Head]];
                                else if(ROB_LogReg[ROB_Head]!=0)ROB_unready0 = 
1<<ROB_PhyReg[ROB_Head];
                                if(ROB_Flushed[ROB_Head]==0)MAP[ROB_LogReg[ROB_Head]] = 
ROB_PhyReg[ROB_Head];
                                if(ROB_Flushed[ROB_Head]==0)begin
                                end
                                ROB_Ready[ROB_Head] = 0;ROB_Flushed[ROB_Head] = 0;
```

```
                              ROB_LogReg[ROB_Head] =  0;ROB_PhyReg[ROB_Head] = 0;ROB_Head =
ROB_Head + 1;
                       end
               end
               else begin
                       ROB_free0 = 0;
               end
               //retireDone = 1;

               if(ROB_Head==ROB_Tail ||
(ROB_Dep[ROB_Head]&&!ROB_Ready[(ROB_Head+1)&5'b11111]&&!ROB_Flushed[ROB_Head]))retireDone = 1;
               if(retireDone != 1 && (ROB_Ready[ROB_Head] == 1 || ROB_Flushed[ROB_Head] ==
1))begin
                       if(ROB_isSys[ROB_Head] && ROB_Flushed[ROB_Head] == 0)begin
                       if(oneCycleDelay!=0)begin
                              oneCycleDelay = oneCycleDelay - 1;
                              retireDone = 1;
                       end
                       else begin
                              num_sys = num_sys + 1;
                              oneCycleDelay = 1;
                              num_instructions_retired = num_instructions_retired  + 1;numRetired
= numRetired + 1;
                              /**retireDone = 1;**/doingSyscall = 1;
                              num_flushes = num_flushes + 1;fullFlush =
1;for(i=0;i<32;i=i+1)ROB_Flushed[i] = 1;
                              correctTarget = ROB_debug_PC[ROB_Head]+4;

                              ROB_Ready[ROB_Head] = 0;ROB_Flushed[ROB_Head] = 0;
                              ROB_LogReg[ROB_Head] =  0;ROB_PhyReg[ROB_Head] = 0;ROB_Head =
ROB_Head + 1;
                              ROB_free1 = 0;
                                             begin
                                               // store register file state to the C
interface functions!
                                               for (j=1; j<32;j=j+1)
                                                     begin
                                                       // Changed to setreg instead of
setregvalue
                                                       $syscall_setreg
(j,CPU.reg_file.RegFile[CPU.MAP[j]]);
                                                     end
                                               $syscall_setpc (ROB_debug_PC[ROB_Head]);
       // store PC

                                               // do the system call
                                               if ($emulate_syscall) begin
                                                     // exit called
                                                     $display ("------------------------
----------");
                                                     $display ("Program Terminated.");
                                                     $display ("Retired %d instructions,
%d entered ROB",num_instructions_retired,num_instructions_inserted);
                                                     $display ("Number of flushes:
%d",num_flushes);
                                                     $display ("Ran for %d
cycles",$time/`cycle);
                                                     $display ("ICache Accesses: %d
\nICache Misses: %d",  i_cache_d_cache.num_icache_accesses,numIMisses);
                                                     $display ("DCache Accesses: %d
\nDCache Misses: %d",  num_d_accesses,numDMisses);
                                                     $display ("Average Fullness: \nROB
%d \nALUQ %d \nCFQ %d \nLDSTAQ %d \nLDSTQ %d \nFreelist(avg. free regs)
%d",(10*ROB_total_occupancy)/($time/`cycle),(10*ALUqueue.total_occupancy)/($time/`cycle),(10*BRJR
queue.total_occupancy)/($time/`cycle),(10*LDSTqueue.total_occupancy)/($time/`cycle),(10*LDSTQ_tot
al_occupancy)/($time/`cycle),(10*free_bird.numFree)/($time/`cycle));
                                                     $display ("RAS
%d",(10*ras.total_occupancy)/($time/`cycle));
                                                     $display ("Max Fullness: \nALUQ %d
\nCFQ %d \nLDSTAQ %d \nLDSTQ %d\nRAS
%d",ALUqueue.max_fullness,BRJRqueue.max_fullness,LDSTqueue.max_fullness,LDSTQ_max_fullness,ras.ma
x_fullness);
                                                            $display ("BR Predictions: %d
\nIncorrect: %d\nJR Predictions: %d \nIncorrect:
%d",num_BR_predictions,num_BR_incorrect,num_JR_predictions,num_JR_incorrect);
                                                            $display ("Correct Path: \nMem %d
\nCF %d \nSys %d \nALU & NOPS %d",num_mem,num_cf,num_sys,num_instructions_retired-
(num_mem+num_cf+num_sys));
                                                            $display ("Retired 0 1 2 3 4: %d %d
%d %d
%d",num_times_retire[0],num_times_retire[1],num_times_retire[2],num_times_retire[3],num_times_ret
ire[4]);
                                                            $display ("Renamed 0 1 2 3 4: %d %d
%d %d
%d",num_times_rename[0],num_times_rename[1],num_times_rename[2],num_times_rename[3],num_times_ren
ame[4]);
                                                            $display ("Cycles Stalled: \nIStall
%d \nRename %d \nROB %d",num_cycles_istall,num_cycles_renamestall,num_cycles_robstall);
                                                            $display ("Forwarded Data: %d\nWrote
Forwards Early: %d\nPerformed Early Loads: %d\nForwarded Data But No Early Write: %d\nROB Head
Loads: %d\nLoads Already Written Before Retire:
%d",num_data_forwarded,num_data_forwarded_written_ahead_of_time,num_early_load_accesses,num_data_
forwarded_not_written,num_rob_head_loads,num_loads_written_before_retire);
                                                            $display ("------------------------
----------");


                                                            $finish;
                                                  end

                                                  // restore register file state from the C
interface
                                                  for (j=1; j<32;j=j+1)
                                                           begin
                                                              // Changed to syscall_getreg
instead of syscall_regvalue

       CPU.reg_file.RegFile[CPU.MAP[j]] = $syscall_getreg(j);
                                                            end
                                               end
                              end
                       end
                       else if(ROB_isCF[ROB_Head][0] && ROB_Flushed[ROB_Head] == 0)begin
                              if(/**!IStall && **/updatePredictor_WE == 0)begin
                                     num_cf = num_cf + 1;
                                     if(ROB_Op[ROB_Head]==`select_qc_j ||
ROB_Op[ROB_Head]==`select_qc_jal)begin
                                            //regular jumps cant be wrong
                                     end
                                     else if(ROB_Op[ROB_Head]==`select_qc_jalr ||
ROB_Op[ROB_Head]==`select_qc_jr)begin
                                            num_JR_predictions = num_JR_predictions + 1;

       if(ROB_Pred_Target[ROB_Head]!=ROB_EffAddr_Target_PC8[ROB_Head])begin
                                                   num_JR_incorrect = num_JR_incorrect + 1;
                                                   correctTarget =
ROB_EffAddr_Target_PC8[ROB_Head];
                                                   num_flushes = num_flushes + 1;fullFlush =
1;for(i=0;i<32;i=i+1)ROB_Flushed[i] = 1;
                                                   /**retireDone = 1;**/
                                            end
                                     end
                                     else begin
                                            if(ROB_Dir[ROB_Head]!=ROB_Pred[ROB_Head])begin
                                                   num_BR_incorrect = num_BR_incorrect + 1;
                                                   if(ROB_Dir[ROB_Head])correctTarget =
ROB_Pred_Target[ROB_Head];
                                                   else correctTarget =
ROB_EffAddr_Target_PC8[ROB_Head];
                                                   num_flushes = num_flushes + 1;fullFlush =
1;for(i=0;i<32;i=i+1)ROB_Flushed[i] = 1;
                                                   /**retireDone = 1;**/
```

```
                                                            end
                                                            num_BR_predictions = num_BR_predictions + 1;

                                                            updatePredictor_WE = 1;
                                                            updatePredictorPC = ROB_debug_PC[ROB_Head];
                                                            updatePrediction = ROB_Dir[ROB_Head];
                                                    end
                                                    //link
                                                    if(ROB_isCF[ROB_Head][2])begin
                                                            ROB_free1 = MAP[ROB_LogReg[ROB_Head]];
                                                            ROB_unready1 = 1<<MAP[ROB_LogReg[ROB_Head]];
                                                            MAP[ROB_LogReg[ROB_Head]] = ROB_PhyReg[ROB_Head];
                                                    end
                                                    else ROB_free1 = 0;

                                                    ROB_Ready[ROB_Head] = 0;ROB_Flushed[ROB_Head] = 0;
                                                    ROB_LogReg[ROB_Head] =  0;ROB_PhyReg[ROB_Head] = 0;ROB_Head
= ROB_Head + 1;
                                                    num_instructions_retired = num_instructions_retired  +
1;numRetired = numRetired + 1;

                                            end
                                            else begin
                                                    retireDone = 1;
                                            end
                                    end
                                    else if(ROB_isMem[ROB_Head] && ROB_Flushed[ROB_Head] == 0)begin
                                            if(isMemRetire==0 && (((ROB_Op[ROB_Head]==`select_mem_sw ||
ROB_Op[ROB_Head]==`select_mem_sh ||
ROB_Op[ROB_Head]==`select_mem_sb)&&(LDSTQ_hasAddr[0]&&LDSTQ_hasData[0]))
                                            ||(!(ROB_Op[ROB_Head]==`select_mem_sw ||
ROB_Op[ROB_Head]==`select_mem_sh ||
ROB_Op[ROB_Head]==`select_mem_sb)&&LDSTQ_hasAddr[0]&&(loadDone==1))))begin
                                                    if(!(ROB_Op[ROB_Head]==`select_mem_sw ||
ROB_Op[ROB_Head]==`select_mem_sh || ROB_Op[ROB_Head]==`select_mem_sb))begin
                                                    //load
                                                            if(LDSTQ_hasData[0] && LDSTQ_wroteValue[0])begin
                                                            num_loads_written_before_retire =
num_loads_written_before_retire+1;

                                                            end
                                                            else if(LDSTQ_hasData[0])begin
                                                            //got data through forwarding but didnt get to write
yet
                                                                    pre_load = LDSTQ_Data[0];
                                                                    load_reg = LDSTQ_Data_Reg[0];
                                                                    pre_load_WE = 1;
                                                                    loadDone = 0;
                                                                    num_data_forwarded_not_written =
num_data_forwarded_not_written + 1;
                                                            end
                                                            else begin
                                                            //doesnt have its data, need to issue cache access
                                                                    isMemRetire = 1;
                                                                    memOp = ROB_Op[ROB_Head];
                                                                    MAR = LDSTQ_Addr[0];
                                                                    load_reg = ROB_PhyReg[ROB_Head];
                                                                    loadDone = 0;
                                                                    num_rob_head_loads = num_rob_head_loads + 1;
                                                                    num_d_accesses = num_d_accesses + 1;
                                                            end
                                                            ROB_free1 = MAP[ROB_LogReg[ROB_Head]];
                                                            ROB_unready1 = 1<<MAP[ROB_LogReg[ROB_Head]];
                                                            MAP[ROB_LogReg[ROB_Head]] = ROB_PhyReg[ROB_Head];
                                                    end
                                                    else begin
                                                    //store
                                                            SMDR = LDSTQ_Data[0];
                                                            isMemRetire = 1;
                                                            memOp = ROB_Op[ROB_Head];
                                                            MAR = LDSTQ_Addr[0];
```

```
                                                ROB_free1 = 0;
                                                num_d_accesses = num_d_accesses + 1;
                                        end
                                        //shift LDSTQ forward
                                        for(i=0;i<31;i=i+1)begin
                                                LDSTQ_Addr_Reg[i] = LDSTQ_Addr_Reg[i+1];
                                                LDSTQ_Data_Reg[i] = LDSTQ_Data_Reg[i+1];
                                                LDSTQ_Addr[i] = LDSTQ_Addr[i+1];
                                                LDSTQ_Data[i] = LDSTQ_Data[i+1];
                                                LDSTQ_hasAddr[i] = LDSTQ_hasAddr[i+1];
                                                LDSTQ_Valid[i] = LDSTQ_Valid[i+1];
                                                LDSTQ_isStore[i] = LDSTQ_isStore[i+1];
                                                LDSTQ_isLoad[i] = LDSTQ_isLoad[i+1];
                                                LDSTQ_hasData[i] = LDSTQ_hasData[i+1];
                                                LDSTQ_ROB[i] = LDSTQ_ROB[i+1];
                                                LDSTQ_Size[i] = LDSTQ_Size[i+1];
                                                LDSTQ_wroteValue[i] = LDSTQ_wroteValue[i+1];
                                        end
                                        LDSTQ_Addr_Reg[31]=0;
                                        LDSTQ_Data_Reg[31]=0;
                                        LDSTQ_Addr[31]=0;
                                        LDSTQ_Data[31]=0;
                                        LDSTQ_hasAddr[31]=0;
                                        LDSTQ_Valid[31]=0;
                                        LDSTQ_isStore[31]=0;
                                        LDSTQ_isLoad[31]=0;
                                        LDSTQ_hasData[31]=0;
                                        LDSTQ_ROB[31]=0;
                                        LDSTQ_Size[31]=0;
                                        LDSTQ_wroteValue[31]=0;
                                        LDSTQ_nextFree = LDSTQ_nextFree - 1;
                                        data_snooper = data_snooper - 1;

                                        num_mem = num_mem + 1;
                                        num_instructions_retired = num_instructions_retired  +
1;numRetired = numRetired + 1;
                                        /**retireDone = 1;**/
                                        ROB_Ready[ROB_Head] = 0;ROB_Flushed[ROB_Head] = 0;
                                        ROB_LogReg[ROB_Head] =  0;ROB_PhyReg[ROB_Head] = 0;ROB_Head
= ROB_Head + 1;
                                end
                                else begin
                                        retireDone = 1;
                                        ROB_free1 = 0;
                                end
                        end
                        else begin
                                if(ROB_Flushed[ROB_Head]==0)num_instructions_retired =
num_instructions_retired  + 1;
                                if(ROB_isCF[ROB_Head][0])begin

        if(ROB_isCF[ROB_Head][2]&&ROB_LogReg[ROB_Head]==31)incorrect_pushes = incorrect_pushes +
1;
                                        if((ROB_Op[ROB_Head]==`select_qc_jalr ||
ROB_Op[ROB_Head]==`select_qc_jr)&&(ROB_debug_instr[ROB_Head][`rs]==31))incorrect_pops =
incorrect_pops + 1;
                                end
                                numRetired = numRetired + 1;
                                if(ROB_Flushed[ROB_Head]==0)ROB_free1 = MAP[ROB_LogReg[ROB_Head]];
                                else if(ROB_LogReg[ROB_Head]!=0)ROB_free1 = ROB_PhyReg[ROB_Head];
                                if(ROB_Flushed[ROB_Head]==0)ROB_unready1 =
1<<MAP[ROB_LogReg[ROB_Head]];
                                else if(ROB_LogReg[ROB_Head]!=0)ROB_unready1 =
1<<ROB_PhyReg[ROB_Head];
                                if(ROB_Flushed[ROB_Head]==0)MAP[ROB_LogReg[ROB_Head]] =
ROB_PhyReg[ROB_Head];
                                if(ROB_Flushed[ROB_Head]==0)begin

                                        end
                                ROB_Ready[ROB_Head] = 0;ROB_Flushed[ROB_Head] = 0;
```

```
                              ROB_LogReg[ROB_Head] =  0;ROB_PhyReg[ROB_Head] = 0;ROB_Head =
ROB_Head + 1;
                      end
               end
               else begin
                      ROB_free1 = 0;
               end
//             retireDone = 1;
               if(ROB_Head==ROB_Tail ||
(ROB_Dep[ROB_Head]&&!ROB_Ready[(ROB_Head+1)&5'b11111]&&!ROB_Flushed[ROB_Head]))retireDone = 1;
               if(retireDone != 1 && (ROB_Ready[ROB_Head] == 1 || ROB_Flushed[ROB_Head] ==
1))begin
                      if(ROB_isSys[ROB_Head] && ROB_Flushed[ROB_Head] == 0)begin
                      if(oneCycleDelay!=0)begin
                             oneCycleDelay = oneCycleDelay - 1;
                             retireDone = 1;
                      end
                      else begin
                             num_sys = num_sys + 1;
                             oneCycleDelay = 1;
                             num_instructions_retired = num_instructions_retired  + 1;numRetired
= numRetired + 1;
                             /**retireDone = 1;**/doingSyscall = 1;
                             num_flushes = num_flushes + 1;fullFlush =
1;for(i=0;i<32;i=i+1)ROB_Flushed[i] = 1;
                             correctTarget = ROB_debug_PC[ROB_Head]+4;


                             ROB_Ready[ROB_Head] = 0;ROB_Flushed[ROB_Head] = 0;
                             ROB_LogReg[ROB_Head] =  0;ROB_PhyReg[ROB_Head] = 0;ROB_Head =
ROB_Head + 1;
                             ROB_free2 = 0;
                                               begin
                                                  // store register file state to the C
interface functions!
                                                  for (j=1; j<32;j=j+1)
                                                        begin
                                                           // Changed to setreg instead of
setregvalue
                                                           $syscall_setreg
(j,CPU.reg_file.RegFile[CPU.MAP[j]]);
                                                         end
                                                  $syscall_setpc (ROB_debug_PC[ROB_Head]);
       // store PC

                                                  // do the system call
                                                  if ($emulate_syscall) begin
                                                         // exit called
                                                         $display ("------------------------
----------");

                                                         $display ("Program Terminated.");
                                                         $display ("Retired %d instructions,
%d entered ROB",num_instructions_retired,num_instructions_inserted);
                                                         $display ("Number of flushes:
%d",num_flushes);
                                                         $display ("Ran for %d
cycles",$time/`cycle);
                                                         $display ("ICache Accesses: %d
\nICache Misses: %d",  i_cache_d_cache.num_icache_accesses,numIMisses);
                                                         $display ("DCache Accesses: %d
\nDCache Misses: %d",  num_d_accesses,numDMisses);
                                                         $display ("Average Fullness: \nROB
%d \nALUQ %d \nCFQ %d \nLDSTAQ %d \nLDSTQ %d \nFreelist(avg. free regs)
%d",(10*ROB_total_occupancy)/($time/`cycle),(10*ALUqueue.total_occupancy)/($time/`cycle),(10*BRJR
queue.total_occupancy)/($time/`cycle),(10*LDSTqueue.total_occupancy)/($time/`cycle),(10*LDSTQ_tot
al_occupancy)/($time/`cycle),(10*free_bird.numFree)/($time/`cycle));
                                                         $display ("RAS
%d",(10*ras.total_occupancy)/($time/`cycle));
                                                         $display ("Max Fullness: \nALUQ %d
\nCFQ %d \nLDSTAQ %d \nLDSTQ %d\nRAS
%d",ALUqueue.max_fullness,BRJRqueue.max_fullness,LDSTqueue.max_fullness,LDSTQ_max_fullness,ras.ma
x_fullness);
                                                            $display ("BR Predictions: %d
\nIncorrect: %d\nJR Predictions: %d \nIncorrect:
%d",num_BR_predictions,num_BR_incorrect,num_JR_predictions,num_JR_incorrect);
                                                            $display ("Correct Path: \nMem %d
\nCF %d \nSys %d \nALU & NOPS %d",num_mem,num_cf,num_sys,num_instructions_retired-
(num_mem+num_cf+num_sys));
                                                            $display ("Retired 0 1 2 3 4: %d %d
%d %d
%d",num_times_retire[0],num_times_retire[1],num_times_retire[2],num_times_retire[3],num_times_ret
ire[4]);
                                                            $display ("Renamed 0 1 2 3 4: %d %d
%d %d
%d",num_times_rename[0],num_times_rename[1],num_times_rename[2],num_times_rename[3],num_times_ren
ame[4]);
                                                            $display ("Cycles Stalled: \nIStall
%d \nRename %d \nROB %d",num_cycles_istall,num_cycles_renamestall,num_cycles_robstall);
                                                            $display ("Forwarded Data: %d\nWrote
Forwards Early: %d\nPerformed Early Loads: %d\nForwarded Data But No Early Write: %d\nROB Head
Loads: %d\nLoads Already Written Before Retire:
%d",num_data_forwarded,num_data_forwarded_written_ahead_of_time,num_early_load_accesses,num_data_
forwarded_not_written,num_rob_head_loads,num_loads_written_before_retire);
                                                            $display ("------------------------
----------");


                                                            $finish;
                                                  end

                                                  // restore register file state from the C
interface
                                                  for (j=1; j<32;j=j+1)
                                                           begin
                                                              // Changed to syscall_getreg
instead of syscall_regvalue

       CPU.reg_file.RegFile[CPU.MAP[j]] = $syscall_getreg(j);
                                                            end
                                               end
                             end
                      end
                      else if(ROB_isCF[ROB_Head][0] && ROB_Flushed[ROB_Head] == 0)begin
                             if(/**!IStall &&**/ updatePredictor_WE == 0)begin
                                    num_cf = num_cf + 1;
                                    if(ROB_Op[ROB_Head]==`select_qc_j ||
ROB_Op[ROB_Head]==`select_qc_jal)begin
                                           //regular jumps cant be wrong
                                    end
                                    else if(ROB_Op[ROB_Head]==`select_qc_jalr ||
ROB_Op[ROB_Head]==`select_qc_jr)begin
                                           num_JR_predictions = num_JR_predictions + 1;

       if(ROB_Pred_Target[ROB_Head]!=ROB_EffAddr_Target_PC8[ROB_Head])begin
                                                  num_JR_incorrect = num_JR_incorrect + 1;
                                                  correctTarget =
ROB_EffAddr_Target_PC8[ROB_Head];
                                                  num_flushes = num_flushes + 1;fullFlush =
1;for(i=0;i<32;i=i+1)ROB_Flushed[i] = 1;
                                                  /**retireDone = 1;**/
                                           end
                                    end
                                    else begin
                                           if(ROB_Dir[ROB_Head]!=ROB_Pred[ROB_Head])begin
                                                  num_BR_incorrect = num_BR_incorrect + 1;
                                                  if(ROB_Dir[ROB_Head])correctTarget =
ROB_Pred_Target[ROB_Head];
                                                  else correctTarget =
ROB_EffAddr_Target_PC8[ROB_Head];
                                                  num_flushes = num_flushes + 1;fullFlush =
1;for(i=0;i<32;i=i+1)ROB_Flushed[i] = 1;
                                                  /**retireDone = 1;**/
```

```
                                                    end
                                                    num_BR_predictions = num_BR_predictions + 1;

                                                    updatePredictor_WE = 1;
                                                    updatePredictorPC = ROB_debug_PC[ROB_Head];
                                                    updatePrediction = ROB_Dir[ROB_Head];
                                            end
                                            //link
                                            if(ROB_isCF[ROB_Head][2])begin
                                                    ROB_free2 = MAP[ROB_LogReg[ROB_Head]];
                                                    ROB_unready2 = 1<<MAP[ROB_LogReg[ROB_Head]];
                                                    MAP[ROB_LogReg[ROB_Head]] = ROB_PhyReg[ROB_Head];
                                            end
                                            else ROB_free2 = 0;

                                            ROB_Ready[ROB_Head] = 0;ROB_Flushed[ROB_Head] = 0;
                                            ROB_LogReg[ROB_Head] =  0;ROB_PhyReg[ROB_Head] = 0;ROB_Head
= ROB_Head + 1;
                                            num_instructions_retired = num_instructions_retired  +
1;numRetired = numRetired + 1;

                                    end
                                    else begin
                                            retireDone = 1;
                                    end
                            end
                            else if(ROB_isMem[ROB_Head] && ROB_Flushed[ROB_Head] == 0)begin
                                    if(isMemRetire==0 && (((ROB_Op[ROB_Head]==`select_mem_sw ||
ROB_Op[ROB_Head]==`select_mem_sh ||
ROB_Op[ROB_Head]==`select_mem_sb)&&(LDSTQ_hasAddr[0]&&LDSTQ_hasData[0]))
                                    ||(!(ROB_Op[ROB_Head]==`select_mem_sw ||
ROB_Op[ROB_Head]==`select_mem_sh ||
ROB_Op[ROB_Head]==`select_mem_sb)&&LDSTQ_hasAddr[0]&&(loadDone==1))))begin
                                            if(!(ROB_Op[ROB_Head]==`select_mem_sw ||
ROB_Op[ROB_Head]==`select_mem_sh || ROB_Op[ROB_Head]==`select_mem_sb))begin
                                            //load
                                                    if(LDSTQ_hasData[0] && LDSTQ_wroteValue[0])begin
                                                    num_loads_written_before_retire =
num_loads_written_before_retire+1;

                                                    end
                                                    else if(LDSTQ_hasData[0])begin
                                                    //got data through forwarding but didnt get to write
yet
                                                            pre_load = LDSTQ_Data[0];
                                                            load_reg = LDSTQ_Data_Reg[0];
                                                            pre_load_WE = 1;
                                                            loadDone = 0;
                                                            num_data_forwarded_not_written =
num_data_forwarded_not_written + 1;
                                                    end
                                                    else begin
                                                    //doesnt have its data, need to issue cache access
                                                            isMemRetire = 1;
                                                            memOp = ROB_Op[ROB_Head];
                                                            MAR = LDSTQ_Addr[0];
                                                            load_reg = ROB_PhyReg[ROB_Head];
                                                            loadDone = 0;
                                                            num_rob_head_loads = num_rob_head_loads + 1;
                                                            num_d_accesses = num_d_accesses + 1;
                                                    end
                                                    ROB_free2 = MAP[ROB_LogReg[ROB_Head]];
                                                    ROB_unready2 = 1<<MAP[ROB_LogReg[ROB_Head]];
                                                    MAP[ROB_LogReg[ROB_Head]] = ROB_PhyReg[ROB_Head];
                                            end
                                            else begin
                                            //store
                                                    SMDR = LDSTQ_Data[0];
                                                    isMemRetire = 1;
                                                    memOp = ROB_Op[ROB_Head];
                                                    MAR = LDSTQ_Addr[0];
```

```
                                        ROB_free2 = 0;
                                        num_d_accesses = num_d_accesses + 1;
                                end
                                //shift LDSTQ forward
                                for(i=0;i<31;i=i+1)begin
                                        LDSTQ_Addr_Reg[i] = LDSTQ_Addr_Reg[i+1];
                                        LDSTQ_Data_Reg[i] = LDSTQ_Data_Reg[i+1];
                                        LDSTQ_Addr[i] = LDSTQ_Addr[i+1];
                                        LDSTQ_Data[i] = LDSTQ_Data[i+1];
                                        LDSTQ_hasAddr[i] = LDSTQ_hasAddr[i+1];
                                        LDSTQ_Valid[i] = LDSTQ_Valid[i+1];
                                        LDSTQ_isStore[i] = LDSTQ_isStore[i+1];
                                        LDSTQ_isLoad[i] = LDSTQ_isLoad[i+1];
                                        LDSTQ_hasData[i] = LDSTQ_hasData[i+1];
                                        LDSTQ_ROB[i] = LDSTQ_ROB[i+1];
                                        LDSTQ_Size[i] = LDSTQ_Size[i+1];
                                        LDSTQ_wroteValue[i] = LDSTQ_wroteValue[i+1];
                                end
                                LDSTQ_Addr_Reg[31]=0;
                                LDSTQ_Data_Reg[31]=0;
                                LDSTQ_Addr[31]=0;
                                LDSTQ_Data[31]=0;
                                LDSTQ_hasAddr[31]=0;
                                LDSTQ_Valid[31]=0;
                                LDSTQ_isStore[31]=0;
                                LDSTQ_isLoad[31]=0;
                                LDSTQ_hasData[31]=0;
                                LDSTQ_ROB[31]=0;
                                LDSTQ_Size[31]=0;
                                LDSTQ_wroteValue[31]=0;
                                LDSTQ_nextFree = LDSTQ_nextFree - 1;
                                data_snooper = data_snooper - 1;

                                num_mem = num_mem + 1;
                                num_instructions_retired = num_instructions_retired  +
1;numRetired = numRetired + 1;
                                /**retireDone = 1;**/
                                ROB_Ready[ROB_Head] = 0;ROB_Flushed[ROB_Head] = 0;
                                ROB_LogReg[ROB_Head] =  0;ROB_PhyReg[ROB_Head] = 0;ROB_Head
= ROB_Head + 1;
                        end
                        else begin

                                retireDone = 1;
                                ROB_free2 = 0;
                        end
                end
                else begin
                        if(ROB_Flushed[ROB_Head]==0)num_instructions_retired =
num_instructions_retired  + 1;
                        if(ROB_isCF[ROB_Head][0])begin

        if(ROB_isCF[ROB_Head][2]&&ROB_LogReg[ROB_Head]==31)incorrect_pushes = incorrect_pushes +
1;
                                if((ROB_Op[ROB_Head]==`select_qc_jalr ||
ROB_Op[ROB_Head]==`select_qc_jr)&&(ROB_debug_instr[ROB_Head][`rs]==31))incorrect_pops =
incorrect_pops + 1;
                        end
                        numRetired = numRetired + 1;
                        if(ROB_Flushed[ROB_Head]==0)ROB_free2 = MAP[ROB_LogReg[ROB_Head]];
                        else if(ROB_LogReg[ROB_Head]!=0)ROB_free2 = ROB_PhyReg[ROB_Head];
                        if(ROB_Flushed[ROB_Head]==0)ROB_unready2 =
1<<MAP[ROB_LogReg[ROB_Head]];
                        else if(ROB_LogReg[ROB_Head]!=0)ROB_unready2 =
1<<ROB_PhyReg[ROB_Head];
                        if(ROB_Flushed[ROB_Head]==0)MAP[ROB_LogReg[ROB_Head]] =
ROB_PhyReg[ROB_Head];
                        if(ROB_Flushed[ROB_Head]==0)begin

                                end
                        ROB_Ready[ROB_Head] = 0;ROB_Flushed[ROB_Head] = 0;
```

```
                              ROB_LogReg[ROB_Head] =  0;ROB_PhyReg[ROB_Head] = 0;ROB_Head = 
ROB_Head + 1;
                      end
               end
               else begin
                      ROB_free2 = 0;
               end
               if(ROB_Head==ROB_Tail ||
(ROB_Dep[ROB_Head]&&!ROB_Ready[(ROB_Head+1)&5'b11111]&&!ROB_Flushed[ROB_Head]))retireDone = 1;
               if(retireDone != 1 && (ROB_Ready[ROB_Head] == 1 || ROB_Flushed[ROB_Head] == 
1))begin
                      if(ROB_isSys[ROB_Head] && ROB_Flushed[ROB_Head] == 0)begin
                      if(oneCycleDelay!=0)begin
                              oneCycleDelay = oneCycleDelay - 1;
                              retireDone = 1;
                      end
                      else begin
                              num_sys = num_sys + 1;
                              oneCycleDelay = 1;
                              num_instructions_retired = num_instructions_retired  + 1;numRetired 
= numRetired + 1;
                              /**retireDone = 1;**/doingSyscall = 1;
                              num_flushes = num_flushes + 1;fullFlush = 
1;for(i=0;i<32;i=i+1)ROB_Flushed[i] = 1;
                              correctTarget = ROB_debug_PC[ROB_Head]+4;

                              ROB_Ready[ROB_Head] = 0;ROB_Flushed[ROB_Head] = 0;
                              ROB_LogReg[ROB_Head] =  0;ROB_PhyReg[ROB_Head] = 0;ROB_Head = 
ROB_Head + 1;
                              ROB_free3 = 0;
                                          begin
                                            // store register file state to the C 
interface functions!
                                            for (j=1; j<32;j=j+1)
                                                  begin
                                                    // Changed to setreg instead of 
setregvalue
                                                    $syscall_setreg 
(j,CPU.reg_file.RegFile[CPU.MAP[j]]);
                                                  end
                                            $syscall_setpc (ROB_debug_PC[ROB_Head]); 
       // store PC

                                            // do the system call
                                            if ($emulate_syscall) begin
                                                  // exit called
                                                  $display ("-------------------------
----------");
                                                  $display ("Program Terminated.");
                                                  $display ("Retired %d instructions, 
%d entered ROB",num_instructions_retired,num_instructions_inserted);
                                                  $display ("Number of flushes: 
%d",num_flushes);
                                                  $display ("Ran for %d 
cycles",$time/`cycle);
                                                  $display ("ICache Accesses: %d 
\nICache Misses: %d",  i_cache_d_cache.num_icache_accesses,numIMisses);
                                                  $display ("DCache Accesses: %d 
\nDCache Misses: %d",  num_d_accesses,numDMisses);
                                                  $display ("Average Fullness: \nROB 
%d \nALUQ %d \nCFQ %d \nLDSTAQ %d \nLDSTQ %d \nFreelist(avg. free regs) 
%d",(10*ROB_total_occupancy)/($time/`cycle),(10*ALUqueue.total_occupancy)/($time/`cycle),(10*BRJR
queue.total_occupancy)/($time/`cycle),(10*LDSTqueue.total_occupancy)/($time/`cycle),(10*LDSTQ_tot
al_occupancy)/($time/`cycle),(10*free_bird.numFree)/($time/`cycle));
                                                  $display ("RAS 
%d",(10*ras.total_occupancy)/($time/`cycle));
                                                  $display ("Max Fullness: \nALUQ %d 
\nCFQ %d \nLDSTAQ %d \nLDSTQ %d\nRAS 
%d",ALUqueue.max_fullness,BRJRqueue.max_fullness,LDSTqueue.max_fullness,LDSTQ_max_fullness,ras.ma
x_fullness);
```

```
                                                              $display ("BR Predictions: %d
\nIncorrect: %d\nJR Predictions: %d \nIncorrect:
%d",num_BR_predictions,num_BR_incorrect,num_JR_predictions,num_JR_incorrect);
                                                              $display ("Correct Path: \nMem %d
\nCF %d \nSys %d \nALU & NOPS %d",num_mem,num_cf,num_sys,num_instructions_retired-
(num_mem+num_cf+num_sys));
                                                              $display ("Retired 0 1 2 3 4: %d %d
%d %d
%d",num_times_retire[0],num_times_retire[1],num_times_retire[2],num_times_retire[3],num_times_ret
ire[4]);
                                                              $display ("Renamed 0 1 2 3 4: %d %d
%d %d
%d",num_times_rename[0],num_times_rename[1],num_times_rename[2],num_times_rename[3],num_times_ren
ame[4]);
                                                              $display ("Cycles Stalled: \nIStall
%d \nRename %d \nROB %d",num_cycles_istall,num_cycles_renamestall,num_cycles_robstall);
                                                              $display ("Forwarded Data: %d\nWrote
Forwards Early: %d\nPerformed Early Loads: %d\nForwarded Data But No Early Write: %d\nROB Head
Loads: %d\nLoads Already Written Before Retire:
%d",num_data_forwarded,num_data_forwarded_written_ahead_of_time,num_early_load_accesses,num_data_
forwarded_not_written,num_rob_head_loads,num_loads_written_before_retire);
                                                              $display ("------------------------
----------");

                                                              $finish;
                                                    end

                                                    // restore register file state from the C
interface
                                                    for (j=1; j<32;j=j+1)
                                                             begin
                                                                // Changed to syscall_getreg
instead of syscall_regvalue

       CPU.reg_file.RegFile[CPU.MAP[j]] = $syscall_getreg(j);
                                                              end
                                                 end
                              end
                      end
                      else if(ROB_isCF[ROB_Head][0] && ROB_Flushed[ROB_Head] == 0)begin
                              if(/**!IStall &&**/ updatePredictor_WE == 0)begin
                                     num_cf = num_cf + 1;
                                     if(ROB_Op[ROB_Head]==`select_qc_j ||
ROB_Op[ROB_Head]==`select_qc_jal)begin
                                            //regular jumps cant be wrong
                                     end
                                     else if(ROB_Op[ROB_Head]==`select_qc_jalr ||
ROB_Op[ROB_Head]==`select_qc_jr)begin
                                            num_JR_predictions = num_JR_predictions + 1;

       if(ROB_Pred_Target[ROB_Head]!=ROB_EffAddr_Target_PC8[ROB_Head])begin
                                                   num_JR_incorrect = num_JR_incorrect + 1;
                                                   correctTarget =
ROB_EffAddr_Target_PC8[ROB_Head];
                                                   num_flushes = num_flushes + 1;fullFlush =
1;for(i=0;i<32;i=i+1)ROB_Flushed[i] = 1;
                                                   /**retireDone = 1;**/
                                            end
                                     end
                                     else begin
                                            if(ROB_Dir[ROB_Head]!=ROB_Pred[ROB_Head])begin
                                                   num_BR_incorrect = num_BR_incorrect + 1;
                                                   if(ROB_Dir[ROB_Head])correctTarget =
ROB_Pred_Target[ROB_Head];
                                                   else correctTarget =
ROB_EffAddr_Target_PC8[ROB_Head];
                                                   num_flushes = num_flushes + 1;fullFlush =
1;for(i=0;i<32;i=i+1)ROB_Flushed[i] = 1;
                                                   /**retireDone = 1;**/
                                            end
                                            num_BR_predictions = num_BR_predictions + 1;
```

```
                                                updatePredictor_WE = 1;
                                                updatePredictorPC = ROB_debug_PC[ROB_Head];
                                                updatePrediction = ROB_Dir[ROB_Head];
                                        end
                                        //link
                                        if(ROB_isCF[ROB_Head][2])begin
                                                ROB_free3 = MAP[ROB_LogReg[ROB_Head]];
                                                ROB_unready3 = 1<<MAP[ROB_LogReg[ROB_Head]];
                                                MAP[ROB_LogReg[ROB_Head]] = ROB_PhyReg[ROB_Head];
                                        end
                                        else ROB_free3 = 0;

                                        ROB_Ready[ROB_Head] = 0;ROB_Flushed[ROB_Head] = 0;
                                        ROB_LogReg[ROB_Head] =  0;ROB_PhyReg[ROB_Head] = 0;ROB_Head
= ROB_Head + 1;
                                        num_instructions_retired = num_instructions_retired  +
1;numRetired = numRetired + 1;

                                end
                                else begin
                                        retireDone = 1;
                                end
                        end
                        else if(ROB_isMem[ROB_Head] && ROB_Flushed[ROB_Head] == 0)begin
                                if(isMemRetire==0 && (((ROB_Op[ROB_Head]==`select_mem_sw ||
ROB_Op[ROB_Head]==`select_mem_sh ||
ROB_Op[ROB_Head]==`select_mem_sb)&&(LDSTQ_hasAddr[0]&&LDSTQ_hasData[0]))
                                ||(!(ROB_Op[ROB_Head]==`select_mem_sw ||
ROB_Op[ROB_Head]==`select_mem_sh ||
ROB_Op[ROB_Head]==`select_mem_sb)&&LDSTQ_hasAddr[0]&&(loadDone==1))))begin
                                        if(!(ROB_Op[ROB_Head]==`select_mem_sw ||
ROB_Op[ROB_Head]==`select_mem_sh || ROB_Op[ROB_Head]==`select_mem_sb))begin
                                        //load
                                                if(LDSTQ_hasData[0] && LDSTQ_wroteValue[0])begin
                                                num_loads_written_before_retire =
num_loads_written_before_retire+1;

                                                end
                                                else if(LDSTQ_hasData[0])begin
                                                //got data through forwarding but didnt get to write
yet
                                                        pre_load = LDSTQ_Data[0];
                                                        load_reg = LDSTQ_Data_Reg[0];
                                                        pre_load_WE = 1;
                                                        loadDone = 0;
                                                        num_data_forwarded_not_written =
num_data_forwarded_not_written + 1;
                                                end
                                                else begin
                                                //doesnt have its data, need to issue cache access
                                                        isMemRetire = 1;
                                                        memOp = ROB_Op[ROB_Head];
                                                        MAR = LDSTQ_Addr[0];
                                                        load_reg = ROB_PhyReg[ROB_Head];
                                                        loadDone = 0;
                                                        num_rob_head_loads = num_rob_head_loads + 1;
                                                        num_d_accesses = num_d_accesses + 1;
                                                end
                                                ROB_free3 = MAP[ROB_LogReg[ROB_Head]];
                                                ROB_unready3 = 1<<MAP[ROB_LogReg[ROB_Head]];
                                                MAP[ROB_LogReg[ROB_Head]] = ROB_PhyReg[ROB_Head];
                                        end
                                        else begin
                                        //store
                                                SMDR = LDSTQ_Data[0];
                                                isMemRetire = 1;
                                                memOp = ROB_Op[ROB_Head];
                                                MAR = LDSTQ_Addr[0];
                                                ROB_free3 = 0;
                                                num_d_accesses = num_d_accesses + 1;
```

```
                                    end
                                    //shift LDSTQ forward
                                    for(i=0;i<31;i=i+1)begin
                                           LDSTQ_Addr_Reg[i] = LDSTQ_Addr_Reg[i+1];
                                           LDSTQ_Data_Reg[i] = LDSTQ_Data_Reg[i+1];
                                           LDSTQ_Addr[i] = LDSTQ_Addr[i+1];
                                           LDSTQ_Data[i] = LDSTQ_Data[i+1];
                                           LDSTQ_hasAddr[i] = LDSTQ_hasAddr[i+1];
                                           LDSTQ_Valid[i] = LDSTQ_Valid[i+1];
                                           LDSTQ_isStore[i] = LDSTQ_isStore[i+1];
                                           LDSTQ_isLoad[i] = LDSTQ_isLoad[i+1];
                                           LDSTQ_hasData[i] = LDSTQ_hasData[i+1];
                                           LDSTQ_ROB[i] = LDSTQ_ROB[i+1];
                                           LDSTQ_Size[i] = LDSTQ_Size[i+1];
                                           LDSTQ_wroteValue[i] = LDSTQ_wroteValue[i+1];
                                    end
                                    LDSTQ_Addr_Reg[31]=0;
                                    LDSTQ_Data_Reg[31]=0;
                                    LDSTQ_Addr[31]=0;
                                    LDSTQ_Data[31]=0;
                                    LDSTQ_hasAddr[31]=0;
                                    LDSTQ_Valid[31]=0;
                                    LDSTQ_isStore[31]=0;
                                    LDSTQ_isLoad[31]=0;
                                    LDSTQ_hasData[31]=0;
                                    LDSTQ_ROB[31]=0;
                                    LDSTQ_Size[31]=0;
                                    LDSTQ_wroteValue[31]=0;
                                    LDSTQ_nextFree = LDSTQ_nextFree - 1;
                                    data_snooper = data_snooper - 1;

                                    num_mem = num_mem + 1;
                                    num_instructions_retired = num_instructions_retired  +
1;numRetired = numRetired + 1;
                                    /**retireDone = 1;**/
                                    ROB_Ready[ROB_Head] = 0;ROB_Flushed[ROB_Head] = 0;
                                    ROB_LogReg[ROB_Head] =  0;ROB_PhyReg[ROB_Head] = 0;ROB_Head
= ROB_Head + 1;
                             end
                             else begin

                                    retireDone = 1;
                                    ROB_free3 = 0;
                             end
                      end
                      else begin
                             if(ROB_Flushed[ROB_Head]==0)num_instructions_retired =
num_instructions_retired  + 1;
                             if(ROB_isCF[ROB_Head][0])begin

       if(ROB_isCF[ROB_Head][2]&&ROB_LogReg[ROB_Head]==31)incorrect_pushes = incorrect_pushes +
1;
                                    if((ROB_Op[ROB_Head]==`select_qc_jalr ||
ROB_Op[ROB_Head]==`select_qc_jr)&&(ROB_debug_instr[ROB_Head][`rs]==31))incorrect_pops =
incorrect_pops + 1;
                             end
                             numRetired = numRetired + 1;
                             if(ROB_Flushed[ROB_Head]==0)ROB_free3 = MAP[ROB_LogReg[ROB_Head]];
                             else if(ROB_LogReg[ROB_Head]!=0)ROB_free3 = ROB_PhyReg[ROB_Head];
                             if(ROB_Flushed[ROB_Head]==0)ROB_unready3 =
1<<MAP[ROB_LogReg[ROB_Head]];
                             else if(ROB_LogReg[ROB_Head]!=0)ROB_unready3 =
1<<ROB_PhyReg[ROB_Head];
                             if(ROB_Flushed[ROB_Head]==0)MAP[ROB_LogReg[ROB_Head]] =
ROB_PhyReg[ROB_Head];
                             if(ROB_Flushed[ROB_Head]==0)begin

                                    end
                             ROB_Ready[ROB_Head] = 0;ROB_Flushed[ROB_Head] = 0;
                             ROB_LogReg[ROB_Head] =  0;ROB_PhyReg[ROB_Head] = 0;ROB_Head =
ROB_Head + 1;
```

```
                        end
                end
                else begin
                        ROB_free3 = 0;
                end

                ROB_Free = numRetired + ROB_Free;
                num_times_retire[numRetired]=num_times_retire[numRetired]+1;
                unresolved_store = 0;
                earlyaccess = 0;
                //Flush LDSTQ
                if(fullFlush)begin
                        for(i=0;i<32;i=i+1)begin
                                LDSTQ_Valid[i] = 0;
                        end
                        LDSTQ_nextFree = 0;
                end
                else begin
                //Issue Loads if nothing else is using the WB reg or d$
                        for(i=0;i<32;i=i+1)begin
                                if(loadDone == 1 && (isMemRetire == 0 || !isLoad &&
LDSTQ_hasData[i]) && LDSTQ_isLoad[i] && LDSTQ_Valid[i] && LDSTQ_hasAddr[i] &&
!LDSTQ_wroteValue[i])begin
                                        if(LDSTQ_hasData[i])begin
                                        //got data through forwarding but didnt get to write yet
                                                pre_load = LDSTQ_Data[i];
                                                load_reg = LDSTQ_Data_Reg[i];
                                                pre_load_WE = 1;
                                                loadDone = 0;
                                                LDSTQ_wroteValue[i] = 1;
                                                num_data_forwarded_written_ahead_of_time =
num_data_forwarded_written_ahead_of_time + 1;
                                        end
                                        else begin
                                                //dont issue the load until you know noone overlaps,
if they can forward the data, wait til they do
                                                unresolved_store = 0;
                                                for(j=0;j<i;j=j+1)begin

        if(LDSTQ_isStore[j]&&(!LDSTQ_hasAddr[j]||((LDSTQ_Addr[j]&32'hFFFFFFFC)==(LDSTQ_Addr[i]&32
'hFFFFFFFC))))begin
                                                                unresolved_store = 1;
                                                        end
                                                end
                                                //doesnt have its data, need to issue cache access
                                                if(unresolved_store == 0)begin
                                                        earlyaccess = 1;
                                                        isMemRetire = 1;
                                                        memOp = ROB_Op[LDSTQ_ROB[i]];
                                                        MAR = LDSTQ_Addr[i];
                                                        load_reg = LDSTQ_Data_Reg[i];
                                                        loadDone = 0;
                                                        LDSTQ_wroteValue[i] = 1;
                                                        LDSTQ_hasData[i] = 1;
                                                        num_early_load_accesses =
num_early_load_accesses + 1;
                                                        num_d_accesses = num_d_accesses + 1;
                                                end
                                        end
                                end
                        end
                end//ldstq

        end

end


always @(*)begin
        if(r_isCFA[2])begin
                link_WE = 1;
                link_reg = r_rdA;
                link_value = r_immA;
                link_ready = 1 << r_rdA;
        end
        else if(r_isCFB[2])begin
                link_WE = 1;
                link_reg = r_rdB;
                link_value = r_immB;
                link_ready = 1 << r_rdB;
        end
        else if(r_isCFC[2])begin
                link_WE = 1;
                link_reg = r_rdC;
                link_value = r_immC;
                link_ready = 1 << r_rdC;
        end
        else if(r_isCFD[2])begin
                link_WE = 1;
                link_reg = r_rdD;
                link_value = r_immD;
                link_ready = 1 << r_rdD;
        end
        else begin
                link_WE = 0;
                link_ready = 0;
                link_value = 0;
                link_reg = 0;
        end
end
wire [63:0] i_readyA,i_readyB,i_readyC,i_readyD;
ready_register_list ready_regs(
        .RESET(RESET),
        .CLK(CLK),
        .toBeReady0(i_readyA),
        .toBeReady1(i_readyB),
        .toBeReady2(link_ready),
        .toBeReady3(load_ready),
        .clearReady0(ROB_unready0),
        .clearReady1(ROB_unready1),
        .clearReady2(ROB_unready2),
        .clearReady3(ROB_unready3),
        .ReadyList(readyList),
        .fullFlush(fullFlush)
);

//Issue Queues
//output wires from issue queue/reg file
wire [7:0] i_opA,i_opB,i_opC,i_opD;
wire [5:0] i_rsA,i_rsB,i_rsC,i_rsD,i_rtA,i_rtB,i_rtC,i_rtD;
wire [5:0] i_rdA,i_rdB,i_rdC,i_rdD;
wire [31:0] iv_rsA,iv_rsB,iv_rsC,iv_rsD,iv_rtA,iv_rtB,iv_rtC,iv_rtD;
wire i_usesImmA,i_usesImmB,i_usesImmC,i_usesImmD;
wire [31:0] i_immA,i_immB,i_immC,i_immD;
wire [4:0] i_ROBA,i_ROBB,i_ROBC,i_ROBD;

wire ALUQueueFull,LDSTQueueFull,BRJRQueueFull;
wire [1:0] ALUNumIssued,LDSTNumIssued,BRJRNumIssued;
//registers piping to execution units
reg [7:0] ir_opA,ir_opB,ir_opC,ir_opD;
reg [5:0] ir_rsA,ir_rsB,ir_rsC,ir_rsD,ir_rtA,ir_rtB,ir_rtC,ir_rtD;
reg [5:0] ir_rdA,ir_rdB,ir_rdC,ir_rdD;
reg [31:0] irv_rsA,irv_rsB,irv_rsC,irv_rsD,irv_rtA,irv_rtB,irv_rtC,irv_rtD;
reg ir_usesImmA,ir_usesImmB,ir_usesImmC,ir_usesImmD;
reg [31:0] ir_immA,ir_immB,ir_immC,ir_immD;
reg [4:0] ir_ROBA,ir_ROBB,ir_ROBC,ir_ROBD;
reg [1:0] ir_ALUNumIssued,ir_LDSTNumIssued,ir_BRJRNumIssued;

//pipeline registers
always @(posedge CLK)begin
        ir_opA = i_opA;ir_opB = i_opB;ir_opC = i_opC;ir_opD = i_opD;
        ir_rsA = i_rsA;ir_rsB = i_rsB;ir_rsC = i_rsC;ir_rsD = i_rsD;
        ir_rtA = i_rtA;ir_rtB = i_rtB;ir_rtC = i_rtC;ir_rtD = i_rtD;
        ir_rdA = i_rdA;ir_rdB = i_rdB;ir_rdC = i_rdC;ir_rdD = i_rdD;
        irv_rsA = iv_rsA;irv_rsB = iv_rsB;irv_rsC = iv_rsC;irv_rsD = iv_rsD;
        irv_rtA = iv_rtA;irv_rtB = iv_rtB;irv_rtC = iv_rtC;irv_rtD = iv_rtD;
        ir_usesImmA = i_usesImmA;ir_usesImmB = i_usesImmB;ir_usesImmC = i_usesImmC;ir_usesImmD =
i_usesImmD;
        ir_immA = i_immA;ir_immB = i_immB;ir_immC = i_immC;ir_immD = i_immD;
        ir_ROBA = i_ROBA;ir_ROBB = i_ROBB;ir_ROBC = i_ROBC;ir_ROBD = i_ROBD;
        ir_ALUNumIssued = ALUNumIssued;ir_LDSTNumIssued = LDSTNumIssued;ir_BRJRNumIssued =
BRJRNumIssued;
end

issue_queue ALUqueue (
        .RESET(RESET),
        .CLK(CLK),
        .fullFlush(fullFlush),
        .maxIssue(2'b10),
        .queueNum(`ALUQueue),
        .readyFlags(readyList),
        .inOperation0(r_opA),.inImmediate0(r_immA),.inRT0(r_rtA),.inRS0(r_rsA),.inRD0(r_rdA),.inQ
ueue0(r_queueA),.inROB0(ROB_slot0),.inUImm0(r_usesImmA),
        .inOperation1(r_opB),.inImmediate1(r_immB),.inRT1(r_rtB),.inRS1(r_rsB),.inRD1(r_rdB),.inQ
ueue1(r_queueB),.inROB1(ROB_slot1),.inUImm1(r_usesImmB),
        .inOperation2(r_opC),.inImmediate2(r_immC),.inRT2(r_rtC),.inRS2(r_rsC),.inRD2(r_rdC),.inQ
ueue2(r_queueC),.inROB2(ROB_slot2),.inUImm2(r_usesImmC),
        .inOperation3(r_opD),.inImmediate3(r_immD),.inRT3(r_rtD),.inRS3(r_rsD),.inRD3(r_rdD),.inQ
ueue3(r_queueD),.inROB3(ROB_slot3),.inUImm3(r_usesImmD),
        .outOperation0(i_opA),.outImmediate0(i_immA),.outRT0(i_rtA),.outRS0(i_rsA),.outRD0(i_rdA)
,.outROB0(i_ROBA),.outUImm0(i_usesImmA),
        .outOperation1(i_opB),.outImmediate1(i_immB),.outRT1(i_rtB),.outRS1(i_rsB),.outRD1(i_rdB)
,.outROB1(i_ROBB),.outUImm1(i_usesImmB),
        .outReadyFlag0(i_readyA),.outReadyFlag1(i_readyB),
        .QueueTooFull(ALUQueueFull),
        .numIssued(ALUNumIssued)
);
issue_queue LDSTqueue(
        .RESET(RESET),
        .CLK(CLK),
        .fullFlush(fullFlush),
        .maxIssue(2'b01),
        .queueNum(`MEMQueue),
        .readyFlags(readyList),
        .inOperation0(r_opA),.inImmediate0(r_immA),.inRT0(6'b0),.inRS0(r_rsA),.inRD0(r_rdA),.inQu
eue0(r_queueA),.inROB0(ROB_slot0),.inUImm0(r_usesImmA),
        .inOperation1(r_opB),.inImmediate1(r_immB),.inRT1(6'b0),.inRS1(r_rsB),.inRD1(r_rdB),.inQu
eue1(r_queueB),.inROB1(ROB_slot1),.inUImm1(r_usesImmB),
        .inOperation2(r_opC),.inImmediate2(r_immC),.inRT2(6'b0),.inRS2(r_rsC),.inRD2(r_rdC),.inQu
eue2(r_queueC),.inROB2(ROB_slot2),.inUImm2(r_usesImmC),
        .inOperation3(r_opD),.inImmediate3(r_immD),.inRT3(6'b0),.inRS3(r_rsD),.inRD3(r_rdD),.inQu
eue3(r_queueD),.inROB3(ROB_slot3),.inUImm3(r_usesImmD),
        .outOperation0(i_opC),.outImmediate0(i_immC),.outRT0(i_rtC),.outRS0(i_rsC),.outRD0(i_rdC)
,.outROB0(i_ROBC),.outUImm0(i_usesImmC),
        .outOperation1(),.outImmediate1(),.outRT1(),.outRS1(),.outRD1(),.outROB1(),.outUImm1(),//
NC
        .outReadyFlag0(i_readyC),
        .outReadyFlag1(),//NC
        .QueueTooFull(LDSTQueueFull),
        .numIssued(LDSTNumIssued)
);
issue_queue BRJRqueue(
        .RESET(RESET),
        .CLK(CLK),
        .fullFlush(fullFlush),
        .maxIssue(2'b01),
        .queueNum(`BRQueue),
        .readyFlags(readyList),
        .inOperation0(r_opA),.inImmediate0(r_immA),.inRT0(r_rtA),.inRS0(r_rsA),.inRD0(r_rdA),.inQ
ueue0(r_queueA),.inROB0(ROB_slot0),.inUImm0(r_usesImmA),
        .inOperation1(r_opB),.inImmediate1(r_immB),.inRT1(r_rtB),.inRS1(r_rsB),.inRD1(r_rdB),.inQ
ueue1(r_queueB),.inROB1(ROB_slot1),.inUImm1(r_usesImmB),
        .inOperation2(r_opC),.inImmediate2(r_immC),.inRT2(r_rtC),.inRS2(r_rsC),.inRD2(r_rdC),.inQ
ueue2(r_queueC),.inROB2(ROB_slot2),.inUImm2(r_usesImmC),
        .inOperation3(r_opD),.inImmediate3(r_immD),.inRT3(r_rtD),.inRS3(r_rsD),.inRD3(r_rdD),.inQ
ueue3(r_queueD),.inROB3(ROB_slot3),.inUImm3(r_usesImmD),
        .outOperation0(i_opD),.outImmediate0(i_immD),.outRT0(i_rtD),.outRS0(i_rsD),.outRD0(i_rdD)
,.outROB0(i_ROBD),.outUImm0(i_usesImmD),
        .outOperation1(),.outImmediate1(),.outRT1(),.outRS1(),.outRD1(),.outROB1(),.outUImm1(),//
NC
        .outReadyFlag0(i_readyD),
        .outReadyFlag1(),//NC
        .QueueTooFull(BRJRQueueFull),
        .numIssued(BRJRNumIssued)
);

//Execution Units
//bypassing/forwarding logic to allow dependant instructions to issue back to back
wire [31:0] alu0_rs_v,alu0_rt_v,alu1_rs_v,alu1_rt_v,ldst_rs_v,brjr_rs_v,brjr_rt_v;

forwarding alu0_rs(
        .rfValue(irv_rsA),.rfAddr(ir_rsA),
        .wb0Value(exv_rdA),.wb0Addr(ex_rdA),//ALU0
        .wb1Value(exv_rdB),.wb1Addr(ex_rdB),//ALU1
        .wb2Value(cacheOut),.wb2Addr(load_reg),//Load?
        .wb3Value(link_value),.wb3Addr(link_reg),//Link
        .outValue(alu0_rs_v)
);
forwarding alu0_rt(
        .rfValue(irv_rtA),.rfAddr(ir_rtA),
        .wb0Value(exv_rdA),.wb0Addr(ex_rdA),//ALU0
        .wb1Value(exv_rdB),.wb1Addr(ex_rdB),//ALU1
        .wb2Value(cacheOut),.wb2Addr(load_reg),//Load?
        .wb3Value(link_value),.wb3Addr(link_reg),//Link
        .outValue(alu0_rt_v)
);
forwarding alu1_rs(
        .rfValue(irv_rsB),.rfAddr(ir_rsB),
        .wb0Value(exv_rdA),.wb0Addr(ex_rdA),//ALU0
        .wb1Value(exv_rdB),.wb1Addr(ex_rdB),//ALU1
        .wb2Value(cacheOut),.wb2Addr(load_reg),//Load?
        .wb3Value(link_value),.wb3Addr(link_reg),//Link
        .outValue(alu1_rs_v)
);
forwarding alu1_rt(
        .rfValue(irv_rtB),.rfAddr(ir_rtB),
        .wb0Value(exv_rdA),.wb0Addr(ex_rdA),//ALU0
        .wb1Value(exv_rdB),.wb1Addr(ex_rdB),//ALU1
        .wb2Value(cacheOut),.wb2Addr(load_reg),//Load?
        .wb3Value(link_value),.wb3Addr(link_reg),//Link
        .outValue(alu1_rt_v)
);
forwarding addr_rs(
        .rfValue(irv_rsC),.rfAddr(ir_rsC),
        .wb0Value(exv_rdA),.wb0Addr(ex_rdA),//ALU0
        .wb1Value(exv_rdB),.wb1Addr(ex_rdB),//ALU1
        .wb2Value(cacheOut),.wb2Addr(load_reg),//Load?
        .wb3Value(link_value),.wb3Addr(link_reg),//Link
        .outValue(ldst_rs_v)
);
forwarding br_rs(
        .rfValue(irv_rsD),.rfAddr(ir_rsD),
        .wb0Value(exv_rdA),.wb0Addr(ex_rdA),//ALU0
        .wb1Value(exv_rdB),.wb1Addr(ex_rdB),//ALU1
        .wb2Value(cacheOut),.wb2Addr(load_reg),//Load?
        .wb3Value(link_value),.wb3Addr(link_reg),//Link
        .outValue(brjr_rs_v)
);
forwarding br_rt(
        .rfValue(irv_rtD),.rfAddr(ir_rtD),
        .wb0Value(exv_rdA),.wb0Addr(ex_rdA),//ALU0
        .wb1Value(exv_rdB),.wb1Addr(ex_rdB),//ALU1
        .wb2Value(cacheOut),.wb2Addr(load_reg),//Load?
```

```
	.wb3Value(link_value),.wb3Addr(link_reg),//Link
	.outValue(brjr_rt_v)
);
forwarding store_value_fwd(
	.rfValue(pre_store_value),.rfAddr(store_reg),
	.wb0Value(exv_rdA),.wb0Addr(ex_rdA),//ALU0
	.wb1Value(exv_rdB),.wb1Addr(ex_rdB),//ALU1
	.wb2Value(cacheOut),.wb2Addr(load_reg),//Load?
	.wb3Value(link_value),.wb3Addr(link_reg),//Link
	.outValue(store_value)
);


always @(posedge CLK)begin
	ex_rdA = ir_rdA;
	ex_rdB = ir_rdB;
	exv_rdA = ALU0_out;
	exv_rdB = ALU1_out;

	if(ir_ALUNumIssued == 2)begin
		ALU0_WE = 1;
		ALU1_WE = 1;
	end
	else if(ir_ALUNumIssued == 1)begin
		ALU0_WE = 1;
		ALU1_WE = 0;
	end
	else begin
		ALU0_WE = 0;
		ALU1_WE = 0;
	end
	gotFlushed = fullFlush;
	LDST_Done = ir_LDSTNumIssued;
	BRJR_Done = ir_BRJRNumIssued;
	ex_BRJR_taken = BRJR_taken;
	ex_EffAddr = EffAddr;
	ex_BRJR_target = BRJR_target;
	ex_ROBA = ir_ROBA;
	ex_ROBB = ir_ROBB;
	ex_ROBC = ir_ROBC;
	ex_ROBD = ir_ROBD;
end

alu alu0(
	.RSOperand(alu0_rs_v),
	.RTOperand(alu0_rt_v),
	.Immediate(ir_immA),
	.UseImmediate(ir_usesImmA),
	.ALUOp(ir_opA),
	.ALUout(ALU0_out)
);
alu alu1(
	.RSOperand(alu1_rs_v),
	.RTOperand(alu1_rt_v),
	.Immediate(ir_immB),
	.UseImmediate(ir_usesImmB),
	.ALUOp(ir_opB),
	.ALUout(ALU1_out)
);
addr_calc ldst_addr(
	.RegVal(ldst_rs_v),
	.Immediate(ir_immC),
	.EffAddr(EffAddr)
);
branch_compare br_qc_jr(
	.RSOperand(brjr_rs_v),
	.RTOperand(brjr_rt_v),
	.ComparisonType(ir_opD),
	.Taken(BRJR_taken),
	.Target(BRJR_target)
);
```

```
reg [5:0] phy_to_log0,phy_to_log1,phy_to_log2,phy_to_log3;

always @(*)begin
	for(i=0;i<34;i=i+1)begin
		if(MAP[i] == ex_rdA) phy_to_log0 = i;
		if(MAP[i] == ex_rdB) phy_to_log1 = i;
		if(MAP[i] == link_reg) phy_to_log2 = i;
		if(MAP[i] == load_reg) phy_to_log3 = i;
	end
end

wire [31:0] pre_load_data;
assign pre_load_data = (pre_load_WE)?pre_load:cacheOut;

//Writeback / RegFile
physical_register_file reg_file(
	.RESET(RESET),
	.CLK(CLK),
	.ReadAddr0(i_rsA),.ReadValue0(iv_rsA),
	.ReadAddr1(i_rsB),.ReadValue1(iv_rsB),
	.ReadAddr2(i_rsC),.ReadValue2(iv_rsC),
	.ReadAddr3(i_rsD),.ReadValue3(iv_rsD),
	.ReadAddr4(i_rtA),.ReadValue4(iv_rtA),
	.ReadAddr5(i_rtB),.ReadValue5(iv_rtB),
	.ReadAddr6(store_reg),.ReadValue6(pre_store_value),
	.ReadAddr7(i_rtD),.ReadValue7(iv_rtD),
	.WriteValue0(exv_rdA),.WriteAddr0(ex_rdA),.WriteEnable0(ALU0_WE),
	.WriteValue1(exv_rdB),.WriteAddr1(ex_rdB),.WriteEnable1(ALU1_WE),
	.WriteValue2(link_value),.WriteAddr2(link_reg),.WriteEnable2(link_WE),
	.WriteValue3(pre_load_data),.WriteAddr3(load_reg),.WriteEnable3(isLoad||pre_load_WE)
);

initial begin
   //$shm_open();
   //$shm_probe(CPU, "AMC");
end

endmodule

******************************************
rename.v

`include "mips.h"

module
rename(RESET,CLK,fullFlush,numInstr,replacementMAP,numFree,renameStall,numRenamed,iStalled,

inRS0,inRT0,inRD0,outRS0,outRT0,outRD0,free0,

inRS1,inRT1,inRD1,outRS1,outRT1,outRD1,free1,

inRS2,inRT2,inRD2,outRS2,outRT2,outRD2,free2,

inRS3,inRT3,inRD3,outRS3,outRT3,outRD3,free3,

ab_swap,bc_swap,cd_swap,ab_dependant,bc_dependant,cd_dependant);
input RESET,CLK,fullFlush,iStalled;
input [2:0] numInstr;
input [203:0] replacementMAP;
input [7:0] numFree;
input ab_swap,bc_swap,cd_swap;
output renameStall;
reg [5:0] newMAP[33:0];
input [5:0]
inRS0,inRT0,inRD0,inRS1,inRT1,inRD1,inRS2,inRT2,inRD2,inRS3,inRT3,inRD3,free0,free1,free2,free3;
output reg [5:0]
outRS0,outRT0,outRD0,outRS1,outRT1,outRD1,outRS2,outRT2,outRD2,outRS3,outRT3,outRD3;
output reg [2:0] numRenamed;
output reg ab_dependant,bc_dependant,cd_dependant;
reg [5:0] MAP[33:0];
reg ab_l,bc_l,cd_l;
wire bothab,bothbc,bothcd;

assign bothab = ab_dependant&ab_l;
assign bothbc = bc_dependant&bc_l;
assign bothcd = cd_dependant&cd_l;

assign renameStall = numFree < 4;

always @(*)begin
        ab_dependant = 0;
        bc_dependant = 0;
        cd_dependant = 0;
        ab_l = 0;
        bc_l = 0;
        cd_l = 0;
        if(ab_swap)begin
                if(inRD0 && (inRD0 == inRS1 || inRD0 == inRT1 || inRD0 == inRD1))ab_dependant = 1;
                else ab_dependant = 0;
                if(inRD1 && (inRD1 == inRS0 || inRD1 == inRT0 || inRD1 == inRD0))ab_l = 1;
                else ab_l = 0;
        end
        if(bc_swap)begin
                if(inRD1 && (inRD1 == inRS2 || inRD1 == inRT2 || inRD1 == inRD2))bc_dependant = 1;
                else bc_dependant = 0;
                if(inRD2 && (inRD2 == inRS1 || inRD2 == inRT1 || inRD2 == inRD1))bc_l = 1;
                else bc_l = 0;
        end
        if(cd_swap)begin
                if(inRD2 && (inRD2 == inRS3 || inRD2 == inRT3 || inRD2 == inRD3))cd_dependant = 1;
                else cd_dependant = 0;
                if(inRD3 && (inRD3 == inRS2 || inRD3 == inRT2 || inRD3 == inRD2))cd_l = 1;
                else cd_l = 0;
        end
end

integer i;

always @(*)begin
        newMAP[0] = replacementMAP[5:0];
        newMAP[1] = replacementMAP[11:6];
        newMAP[2] = replacementMAP[17:12];
        newMAP[3] = replacementMAP[23:18];
        newMAP[4] = replacementMAP[29:24];
        newMAP[5] = replacementMAP[35:30];
        newMAP[6] = replacementMAP[41:36];
        newMAP[7] = replacementMAP[47:42];
        newMAP[8] = replacementMAP[53:48];
        newMAP[9] = replacementMAP[59:54];
        newMAP[10] = replacementMAP[65:60];
        newMAP[11] = replacementMAP[71:66];
        newMAP[12] = replacementMAP[77:72];
        newMAP[13] = replacementMAP[83:78];
        newMAP[14] = replacementMAP[89:84];
        newMAP[15] = replacementMAP[95:90];
        newMAP[16] = replacementMAP[101:96];
        newMAP[17] = replacementMAP[107:102];
        newMAP[18] = replacementMAP[113:108];
        newMAP[19] = replacementMAP[119:114];
        newMAP[20] = replacementMAP[125:120];
        newMAP[21] = replacementMAP[131:126];
        newMAP[22] = replacementMAP[137:132];
        newMAP[23] = replacementMAP[143:138];
        newMAP[24] = replacementMAP[149:144];
        newMAP[25] = replacementMAP[155:150];
        newMAP[26] = replacementMAP[161:156];
        newMAP[27] = replacementMAP[167:162];
        newMAP[28] = replacementMAP[173:168];
        newMAP[29] = replacementMAP[179:174];
        newMAP[30] = replacementMAP[185:180];
        newMAP[31] = replacementMAP[191:186];
        newMAP[32] = replacementMAP[197:192];
        newMAP[33] = replacementMAP[203:198];
end

always @(posedge CLK)begin
        if(RESET)begin
                for(i=0;i<34;i=i+1)begin
                        MAP[i] = i;

                end
                outRS0 = 0; outRT0 = 0; outRD0 = 0;
                outRS1 = 0; outRT1 = 0; outRD1 = 0;
                outRS2 = 0; outRT2 = 0; outRD2 = 0;
                outRS3 = 0; outRT3 = 0; outRD3 = 0;
                numRenamed = 0;
        end
        else begin
                if(fullFlush)begin
                        for(i=0;i<34;i=i+1)begin
                                MAP[i] = newMAP[i];

                        end
                        outRS0 = 0; outRT0 = 0; outRD0 = 0;
                        outRS1 = 0; outRT1 = 0; outRD1 = 0;
                        outRS2 = 0; outRT2 = 0; outRD2 = 0;
                        outRS3 = 0; outRT3 = 0; outRD3 = 0;
                        numRenamed = 0;
                end
                else begin
                        if(renameStall)begin
                                outRS0 = 0; outRT0 = 0; outRD0 = 0;
                                outRS1 = 0; outRT1 = 0; outRD1 = 0;
                                outRS2 = 0; outRT2 = 0; outRD2 = 0;
                                outRS3 = 0; outRT3 = 0; outRD3 = 0;
                                numRenamed = 0;
                        end
                        else begin
                                outRS0 = 0; outRT0 = 0; outRD0 = 0;
                                outRS1 = 0; outRT1 = 0; outRD1 = 0;
                                outRS2 = 0; outRT2 = 0; outRD2 = 0;
                                outRS3 = 0; outRT3 = 0; outRD3 = 0;
                                case(numInstr)
                                        3'd0:begin
                                              numRenamed = 0;
                                              end
                                        3'd1:begin
                                              numRenamed = 0;
                                              outRS0 = MAP[inRS0];
                                              outRT0 = MAP[inRT0];
                                                    if(inRD0)begin
                                                    case(numRenamed)
                                                     3'd0:begin
                                                                   outRD0 = free0;
                                                                   MAP[inRD0] = free0;
                                                             end
                                                     3'd1:begin
                                                             end
                                                     3'd2:begin
                                                          end
                                                     3'd3:begin
                                                             end
                                                     default:begin
                                                                    end
                                                     endcase
                                                     numRenamed = numRenamed + 1;
                                                    end
                                                    else begin
                                                     outRD0 = 0;
                                                    end
                                              end
                                        3'd2:begin
```

```
if(ab_swap)begin
        numRenamed = 0;
        outRS1 = MAP[inRS1];
        outRT1 = MAP[inRT1];
              if(inRD1)begin
              case(numRenamed)
               3'd0:begin
                            outRD1 = free0;
                            MAP[inRD1] = free0;
                       end
               3'd1:begin
                       end
               3'd2:begin
                    end
               3'd3:begin
                       end
               default:begin
                              end
               endcase
               numRenamed = numRenamed + 1;
              end
              else begin
               outRD1 = 0;
              end
        outRS0 = MAP[inRS0];
        outRT0 = MAP[inRT0];
              if(inRD0)begin
              case(numRenamed)
               3'd0:begin
                            outRD0 = free0;
                            MAP[inRD0] = free0;
                       end
               3'd1:begin
                            outRD0 = free1;
                            MAP[inRD0] = free1;
                       end
               3'd2:begin
                    end
               3'd3:begin
                       end
               default:begin
                              end
               endcase
               numRenamed = numRenamed + 1;
              end
              else begin
               outRD0 = 0;
              end
end
else begin
        numRenamed = 0;
        outRS0 = MAP[inRS0];
        outRT0 = MAP[inRT0];
              if(inRD0)begin
              case(numRenamed)
               3'd0:begin
                            outRD0 = free0;
                            MAP[inRD0] = free0;
                       end
               3'd1:begin
                       end
               3'd2:begin
                    end
               3'd3:begin
                       end
               default:begin
                              end
               endcase
               numRenamed = numRenamed + 1;
              end
              else begin
```

```
                                                             outRD0 = 0;
                                                            end
                                                      outRS1 = MAP[inRS1];
                                                      outRT1 = MAP[inRT1];
                                                            if(inRD1)begin
                                                            case(numRenamed)
                                                             3'd0:begin
                                                                           outRD1 = free0;
                                                                           MAP[inRD1] = free0;
                                                                      end
                                                             3'd1:begin
                                                                           outRD1 = free1;
                                                                           MAP[inRD1] = free1;
                                                                      end
                                                             3'd2:begin
                                                                  end
                                                             3'd3:begin
                                                                      end
                                                             default:begin
                                                                            end
                                                             endcase
                                                             numRenamed = numRenamed + 1;
                                                            end
                                                            else begin
                                                             outRD1 = 0;
                                                            end
                                                      end
                                             end
                                      3'd3:begin
                                              numRenamed = 0;
                                              outRS0 = MAP[inRS0];
                                              outRT0 = MAP[inRT0];
                                                     if(inRD0)begin
                                                     case(numRenamed)
                                                      3'd0:begin
                                                                    outRD0 = free0;
                                                                    MAP[inRD0] = free0;
                                                              end
                                                      3'd1:begin
                                                              end
                                                      3'd2:begin
                                                           end
                                                      3'd3:begin
                                                              end
                                                      default:begin
                                                                     end
                                                      endcase
                                                      numRenamed = numRenamed + 1;
                                                     end
                                                     else begin
                                                      outRD0 = 0;
                                                     end
                                                     if(bc_swap)begin
                                                             outRS2 = MAP[inRS2];
                                                             outRT2 = MAP[inRT2];
                                                                    if(inRD2)begin
                                                                    case(numRenamed)
                                                                     3'd0:begin
                                                                                   outRD2 =
free0;
                                                                                   MAP[inRD2] =
free0;
                                                                              end
                                                                     3'd1:begin
                                                                                   outRD2 =
free1;
                                                                                   MAP[inRD2] =
free1;
                                                                              end
                                                                     3'd2:begin
                                                                          end
```

```
                                                          3'd3:begin
                                                                  end
                                                          default:begin
                                                                        end
                                                          endcase
                                                          numRenamed = numRenamed + 1;
                                                         end
                                                         else begin
                                                          outRD2 = 0;
                                                         end
                                                  outRS1 = MAP[inRS1];
                                                  outRT1 = MAP[inRT1];
                                                         if(inRD1)begin
                                                         case(numRenamed)
                                                          3'd0:begin
                                                                       outRD1 =
free0;
                                                                       MAP[inRD1] =
free0;
                                                                  end
                                                          3'd1:begin
                                                                       outRD1 =
free1;
                                                                       MAP[inRD1] =
free1;
                                                                  end
                                                          3'd2:begin
                                                                       outRD1 =
free2;
                                                                       MAP[inRD1] =
free2;
                                                               end
                                                          3'd3:begin
                                                                  end
                                                          default:begin
                                                                        end
                                                          endcase
                                                          numRenamed = numRenamed + 1;
                                                         end
                                                         else begin
                                                          outRD1 = 0;
                                                         end
                                           end
                                           else begin
                                                  outRS1 = MAP[inRS1];
                                                  outRT1 = MAP[inRT1];
                                                         if(inRD1)begin
                                                         case(numRenamed)
                                                          3'd0:begin
                                                                       outRD1 =
free0;
                                                                       MAP[inRD1] =
free0;
                                                                  end
                                                          3'd1:begin
                                                                       outRD1 =
free1;
                                                                       MAP[inRD1] =
free1;
                                                                  end
                                                          3'd2:begin
                                                               end
                                                          3'd3:begin
                                                                  end
                                                          default:begin
                                                                        end
                                                          endcase
                                                          numRenamed = numRenamed + 1;
                                                         end
                                                         else begin
                                                          outRD1 = 0;
```

```
                                                                    end
                                                              outRS2 = MAP[inRS2];
                                                              outRT2 = MAP[inRT2];
                                                                    if(inRD2)begin
                                                                    case(numRenamed)
                                                                     3'd0:begin
                                                                                  outRD2 =
free0;
                                                                                  MAP[inRD2] =
free0;
                                                                             end
                                                                     3'd1:begin
                                                                                  outRD2 =
free1;
                                                                                  MAP[inRD2] =
free1;
                                                                             end
                                                                     3'd2:begin
                                                                                  outRD2 =
free2;
                                                                                  MAP[inRD2] =
free2;
                                                                          end
                                                                     3'd3:begin
                                                                             end
                                                                     default:begin
                                                                                   end
                                                                     endcase
                                                                     numRenamed = numRenamed + 1;
                                                                    end
                                                                    else begin
                                                                     outRD2 = 0;
                                                                    end
                                                       end
                                                 end
                                         3'd4:begin
                                                 numRenamed = 0;
                                                 outRS0 = MAP[inRS0];
                                                 outRT0 = MAP[inRT0];
                                                       if(inRD0)begin
                                                       case(numRenamed)
                                                        3'd0:begin
                                                                    outRD0 = free0;
                                                                    MAP[inRD0] = free0;
                                                               end
                                                        3'd1:begin
                                                               end
                                                        3'd2:begin
                                                            end
                                                        3'd3:begin
                                                               end
                                                        default:begin
                                                                     end
                                                        endcase
                                                        numRenamed = numRenamed + 1;
                                                       end
                                                       else begin
                                                        outRD0 = 0;
                                                       end
                                                 outRS1 = MAP[inRS1];
                                                 outRT1 = MAP[inRT1];
                                                       if(inRD1)begin
                                                       case(numRenamed)
                                                        3'd0:begin
                                                                    outRD1 = free0;
                                                                    MAP[inRD1] = free0;
                                                               end
                                                        3'd1:begin
                                                                    outRD1 = free1;
                                                                    MAP[inRD1] = free1;
                                                               end
```

```
                                                                    3'd2:begin
                                                                         end
                                                                    3'd3:begin
                                                                            end
                                                                    default:begin
                                                                                   end
                                                                    endcase
                                                                    numRenamed = numRenamed + 1;
                                                                   end
                                                                   else begin
                                                                    outRD1 = 0;
                                                                   end
                                                                   if(cd_swap)begin
                                                                            outRS3 = MAP[inRS3];
                                                                            outRT3 = MAP[inRT3];
                                                                                  if(inRD3)begin
                                                                                  case(numRenamed)
                                                                                   3'd0:begin
                                                                                                 outRD3 =
free0;
                                                                                                 MAP[inRD3] =
free0;
                                                                                            end
                                                                                   3'd1:begin
                                                                                                 outRD3 =
free1;
                                                                                                 MAP[inRD3] =
free1;
                                                                                            end
                                                                                   3'd2:begin
                                                                                                 outRD3 =
free2;
                                                                                                 MAP[inRD3] =
free2;
                                                                                        end
                                                                                   3'd3:begin
                                                                                            end
                                                                                   default:begin
                                                                                                  end
                                                                                   endcase
                                                                                   numRenamed = numRenamed + 1;
                                                                                  end
                                                                                  else begin
                                                                                   outRD3 = 0;
                                                                                  end
                                                                            outRS2 = MAP[inRS2];
                                                                            outRT2 = MAP[inRT2];
                                                                                  if(inRD2)begin
                                                                                  case(numRenamed)
                                                                                   3'd0:begin
                                                                                                 outRD2 =
free0;
                                                                                                 MAP[inRD2] =
free0;
                                                                                            end
                                                                                   3'd1:begin
                                                                                                 outRD2 =
free1;
                                                                                                 MAP[inRD2] =
free1;
                                                                                            end
                                                                                   3'd2:begin
                                                                                                 outRD2 =
free2;
                                                                                                 MAP[inRD2] =
free2;
                                                                                        end
                                                                                   3'd3:begin
                                                                                                 outRD2 =
free3;
```

```
                                                                                   MAP[inRD2] =
free3;

                                                                             end
                                                                     default:begin
                                                                                    end
                                                                     endcase
                                                                     numRenamed = numRenamed + 1;
                                                                    end
                                                                    else begin
                                                                     outRD2 = 0;
                                                                    end
                                                     end
                                                     else begin
                                                      outRS2 = MAP[inRS2];
                                                      outRT2 = MAP[inRT2];
                                                            if(inRD2)begin
                                                            case(numRenamed)
                                                             3'd0:begin
                                                                           outRD2 = free0;
                                                                           MAP[inRD2] = free0;
                                                                      end
                                                             3'd1:begin
                                                                           outRD2 = free1;
                                                                           MAP[inRD2] = free1;
                                                                      end
                                                             3'd2:begin
                                                                           outRD2 = free2;
                                                                           MAP[inRD2] = free2;

                                                                  end
                                                             3'd3:begin
                                                                      end
                                                             default:begin
                                                                            end
                                                             endcase
                                                             numRenamed = numRenamed + 1;
                                                            end
                                                            else begin
                                                             outRD2 = 0;
                                                            end
                                                      outRS3 = MAP[inRS3];
                                                      outRT3 = MAP[inRT3];
                                                            if(inRD3)begin
                                                            case(numRenamed)
                                                             3'd0:begin
                                                                           outRD3 = free0;
                                                                           MAP[inRD3] = free0;
                                                                      end
                                                             3'd1:begin
                                                                           outRD3 = free1;
                                                                           MAP[inRD3] = free1;
                                                                      end
                                                             3'd2:begin
                                                                           outRD3 = free2;
                                                                           MAP[inRD3] = free2;

                                                                  end
                                                             3'd3:begin
                                                                           outRD3 = free3;
                                                                           MAP[inRD3] = free3;

                                                                      end
                                                             default:begin
                                                                            end
                                                             endcase
                                                             numRenamed = numRenamed + 1;
                                                            end
                                                            else begin
                                                             outRD3 = 0;
                                                            end
```

```
                                                    end
                                               end
                                     default: begin
                                                     numRenamed = 0;
                                                     end
                                     endcase
                          end
                   end
            end
end

endmodule

******************************************
freelist.v

`include "mips.h"

module
freelist(RESET,CLK,numRequest,numFree,free0,free1,free2,free3,numAdd,nowFree0,nowFree1,nowFree2,n
owFree3);
input RESET,CLK;
input [2:0] numRequest;
output reg [7:0] numFree;
output [5:0] free0,free1,free2,free3;
input [2:0] numAdd;
input [5:0] nowFree0,nowFree1,nowFree2,nowFree3;
reg [31:0] total_free;
reg [5:0] freeRegisters[29:0];
reg [5:0] head,tail;//add at tail, remove from head
reg [2:0] nonZero;

assign free3 = freeRegisters[head];
assign free2 = freeRegisters[(head+30-1)%30];
assign free1 = freeRegisters[(head+30-2)%30];
assign free0 = freeRegisters[(head+30-3)%30];

integer i;

always @(negedge CLK)begin
        if(RESET)begin
                for(i=0;i<30;i=i+1)freeRegisters[i]=i+34;
                head = 3;
                tail = 0;
                numFree = 30;
                total_free = 0;
        end
        else begin

                nonZero = 0;
                if(nowFree0)begin
                        freeRegisters[tail] = nowFree0;
                        tail = tail + 1;
                        if(tail==30)tail=0;
                        nonZero = nonZero + 1;
                end
                if(nowFree1)begin
                        freeRegisters[tail] = nowFree1;
                        tail = tail + 1;
                        if(tail==30)tail=0;
                        nonZero = nonZero + 1;
                end
                if(nowFree2)begin
                        freeRegisters[tail] = nowFree2;
                        tail = tail + 1;
                        if(tail==30)tail=0;
                        nonZero = nonZero + 1;
                end
                if(nowFree3)begin
                        freeRegisters[tail] = nowFree3;
                        tail = tail + 1;
                        if(tail==30)tail=0;
                        nonZero = nonZero + 1;
                end

                numFree = numFree - numRequest + nonZero;
                total_free = total_free + numFree;
                head = head + numRequest;
                if(head>=30)head=head-30;
        end
end

endmodule

******************************************
return_address_stack.v

`include "mips.h"

module return_address_stack(RESET,CLK,push,pop,LinkPC,PredPC,incorrect_pushes,incorrect_pops);
input RESET;
input CLK;
input push;
input pop;
input [31:0] LinkPC;
output reg [31:0] PredPC;
reg [31:0] predStack[15:0];
reg [3:0] count;
reg [31:0] missed;
input [2:0] incorrect_pops,incorrect_pushes;
reg [2:0] add_pops,remove_pushes;
reg [31:0] total_occupancy,max_fullness;


integer i;

always @(posedge CLK)begin
        if(RESET)begin
                count = 0;
                missed = 0;
                for(i=0;i<16;i=i+1)begin
                        predStack[i] = 0;
                end
                PredPC = 0;
                total_occupancy = 0;
                max_fullness = 0;
        end
        else begin
                add_pops = incorrect_pops;
                remove_pushes = incorrect_pushes;
                for(i=0;i<4;i=i+1)begin
                        if(add_pops)begin
                                if(count != 15)begin
                                        count = count + 1;
                                end
                                else begin
                                        missed = missed + 1;
                                end
                                add_pops = add_pops - 1;
                        end
                end
                for(i=0;i<4;i=i+1)begin
                        if(remove_pushes)begin
                                if(missed != 0)begin
                                        missed = missed - 1;
                                end
                                else begin
                                        if(count != 0)begin
                                                count = count - 1;
                                        end
                                end
                                remove_pushes = remove_pushes - 1;
                        end
                end
                if(count!=0)PredPC = predStack[count-1];
                if(pop)begin
                        if(missed != 0)begin
                                missed = missed - 1;
                        end
                        else begin
                                if(count != 0)begin
                                        count = count - 1;
                                end
                        end
                end
                if(push)begin
                        if(count != 15)begin
                                predStack[count]=LinkPC;
                                count = count + 1;
                        end
                        else begin
                                missed = missed + 1;
                        end
                end
                total_occupancy = total_occupancy + count+missed;
                if(count+missed > max_fullness)max_fullness = count+missed;
        end
end

endmodule

******************************************
branch_predictor.v

`include "mips.h"

module
branch_predictor(RESET,CLK,PC,Prediction,UpdateEnable,UpdatePC,UpdateValue,PredictorSelect,Direct
ion);
input RESET;
input CLK;
input [31:0] PC;
output Prediction;
input UpdateEnable;
input [31:0] UpdatePC;
input UpdateValue;
input [2:0] PredictorSelect;
input Direction;

wire DPred;
wire GPred;
wire PPred;
wire TPred;
wire VPred;

reg [7:0] GHR;
reg [7:0] LHR [255:0];
reg [1:0] GSatCnt [255:0];
reg [1:0] PSatCnt [255:0];
reg [1:0] TSatCnt [255:0];

integer i;

assign DPred = (Direction)? 1'b1 : 1'b0;//If direction is backwards, predict taken, if direction
is forward, predict not taken
assign GPred = GSatCnt[GHR][1];//Indexed by GHR, top most bit of sat cnt is prediction
assign PPred = PSatCnt[PC[9:2]][1];//Indexed by lower bits of PC (not the two LSB, =0),top most
bit of sat cnt is prediction
assign TPred = TSatCnt[LHR[PC[9:2]]][1];//Index first table of history registers with pc, index
sat cnt table with that history
assign VPred = ((TPred&GPred)|(TPred&PPred)|(GPred&PPred));

//Select the Prediction to Use
```

```
assign  Prediction = (PredictorSelect == `select_pred_dpred)?DPred:1'bz,
               Prediction = (PredictorSelect == `select_pred_gpred)?GPred:1'bz,
               Prediction = (PredictorSelect == `select_pred_ppred)?PPred:1'bz,
        Prediction = (PredictorSelect == `select_pred_tpred)?TPred:1'bz,
               Prediction = (PredictorSelect == `select_pred_vpred)?VPred:1'bz;

always @(posedge CLK or RESET)begin
       if(RESET)begin
               //Clear table on Reset;Shouldn't need to but simulator will complain undefined
otherwise
               GHR = 0;
               for(i=0;i<256;i=i+1) LHR[i] = 0;
               for(i=0;i<256;i=i+1) GSatCnt[i] = 2'b01;
               for(i=0;i<256;i=i+1) PSatCnt[i] = 2'b01;
               for(i=0;i<256;i=i+1) TSatCnt[i] = 2'b01;
       end
       else if(UpdateEnable)begin
               //GPred
               case(GSatCnt[GHR])
               2'b00:begin
                              if(UpdateValue)GSatCnt[GHR] = 2'b01;
                              else GSatCnt[GHR] = 2'b00;
                         end
               2'b01:begin
                              if(UpdateValue)GSatCnt[GHR] = 2'b10;
                              else GSatCnt[GHR] = 2'b00;
                         end
               2'b10:begin
                              if(UpdateValue)GSatCnt[GHR] = 2'b11;
                              else GSatCnt[GHR] = 2'b01;
                         end
               2'b11:begin
                              if(UpdateValue)GSatCnt[GHR] = 2'b11;
                              else GSatCnt[GHR] = 2'b10;
                         end
               endcase
               GHR = (GHR << 1)|UpdateValue;

               //PPred
               case(PSatCnt[UpdatePC[9:2]])
               2'b00:begin
                              if(UpdateValue)PSatCnt[UpdatePC[9:2]] = 2'b01;
                              else PSatCnt[UpdatePC[9:2]] = 2'b00;
                         end
               2'b01:begin
                              if(UpdateValue)PSatCnt[UpdatePC[9:2]] = 2'b10;
                              else PSatCnt[UpdatePC[9:2]] = 2'b00;
                         end
               2'b10:begin
                              if(UpdateValue)PSatCnt[UpdatePC[9:2]] = 2'b11;
                              else PSatCnt[UpdatePC[9:2]] = 2'b01;
                         end
               2'b11:begin
                              if(UpdateValue)PSatCnt[UpdatePC[9:2]] = 2'b11;
                              else PSatCnt[UpdatePC[9:2]] = 2'b10;
                         end
               endcase

               //TPred
               case(TSatCnt[LHR[UpdatePC[9:2]]])
               2'b00:begin
                              if(UpdateValue)TSatCnt[LHR[UpdatePC[9:2]]] = 2'b01;
                              else TSatCnt[LHR[UpdatePC[9:2]]] = 2'b00;
                         end
               2'b01:begin
                              if(UpdateValue)TSatCnt[LHR[UpdatePC[9:2]]] = 2'b10;
                              else TSatCnt[LHR[UpdatePC[9:2]]] = 2'b00;
                         end
               2'b10:begin
                              if(UpdateValue)TSatCnt[LHR[UpdatePC[9:2]]] = 2'b11;
                              else TSatCnt[LHR[UpdatePC[9:2]]] = 2'b01;
```

```
                        end
                2'b11:begin
                                if(UpdateValue)TSatCnt[LHR[UpdatePC[9:2]]] = 2'b11;
                                else TSatCnt[LHR[UpdatePC[9:2]]] = 2'b10;
                        end
                endcase
                LHR[UpdatePC[9:2]] = (LHR[UpdatePC[9:2]]<<1)|UpdateValue;

        end
end



endmodule

****************************************
decode.v
`include "mips.h"

module
decode(instr,operation,operation2,queue,rs,rt,rd,rd2,usesImmediate,immediate,isSyscall,isMem,isMu
ltDiv,isCF,isALU,PC,target,fullFlush);

input [31:0] instr;
input [31:0] PC;
input fullFlush;
output reg [7:0] operation,operation2;
output reg [1:0] queue;
output reg [5:0] rs,rt,rd,rd2;
output reg usesImmediate;
output reg [31:0] immediate;
output reg isSyscall,isMem,isMultDiv,isALU;
output reg [2:0] isCF;
output reg [31:0] target;
wire [31:0] PC4;
assign PC4 = PC+4;

always @(*) begin
if(fullFlush)begin
        operation = 0;
        operation2 = 0;
        queue = 0;
        rs = 0;
        rt = 0;
        rd = 0;
        rd2 = 0;
        immediate = 0; usesImmediate = 0;
        isSyscall = 0;
        isMem = 0;
        isMultDiv = 0;
        isCF = 0;
        isALU = 0;
        target = 0;
end
else begin
    casex({instr[`op], instr[`function], instr[`rt]})
        {`SPECIAL, `ADD,     `dc5 }:begin
                                                             operation = `select_alu_add;
                                                             operation2 = 0;
                                                             queue = `ALUQueue;
                                                             rs = instr[`rs];
                                                             rt = instr[`rt];
                                                             rd = instr[`rd];
                                                             rd2 = 0;
                                                             immediate = 0; usesImmediate = 0;
                                                             isSyscall = 0;
                                                             isMem = 0;
                                                             isMultDiv = 0;
                                                             isCF = 0;
                                                             isALU = 1;
                                                            target = PC+4;
```

```
                                                end
{`SPECIAL, `ADDU,    `dc5 }:begin
                                                 operation = `select_alu_add;
                                                 operation2 = 0;
                                                 queue = `ALUQueue;
                                                 rs = instr[`rs];
                                                 rt = instr[`rt];
                                                 rd = instr[`rd];
                                                 rd2 = 0;
                                                 immediate = 0; usesImmediate = 0;
                                                 isSyscall = 0;
                                                 isMem = 0;
                                                 isMultDiv = 0;
                                                 isCF = 0;
                                                 isALU = 1;
                                                target = PC+4;
                                                end
{`SPECIAL, `SUB,     `dc5 }:begin
                                                operation = `select_alu_sub;
                                                 operation2 = 0;
                                                 queue = `ALUQueue;
                                                 rs = instr[`rs];
                                                 rt = instr[`rt];
                                                 rd = instr[`rd];
                                                 rd2 = 0;
                                                 immediate = 0; usesImmediate = 0;
                                                 isSyscall = 0;
                                                 isMem = 0;
                                                 isMultDiv = 0;
                                                 isCF = 0;
                                                 isALU = 1;
                                                target = PC+4;
                                                end
{`SPECIAL, `SUBU,    `dc5 }:begin
                                                operation = `select_alu_sub;
                                                 operation2 = 0;
                                                 queue = `ALUQueue;
                                                 rs = instr[`rs];
                                                 rt = instr[`rt];
                                                 rd = instr[`rd];
                                                 rd2 = 0;
                                                 immediate = 0; usesImmediate = 0;
                                                 isSyscall = 0;
                                                 isMem = 0;
                                                 isMultDiv = 0;
                                                 isCF = 0;
                                                 isALU = 1;
                                                target = PC+4;
                                                end
{`SPECIAL, `SLT,     `dc5 }:begin
                                                 operation = `select_alu_slt;
                                                 operation2 = 0;
                                                 queue = `ALUQueue;
                                                 rs = instr[`rs];
                                                 rt = instr[`rt];
                                                 rd = instr[`rd];
                                                 rd2 = 0;
                                                 immediate = 0; usesImmediate = 0;
                                                 isSyscall = 0;
                                                 isMem = 0;
                                                 isMultDiv = 0;
                                                 isCF = 0;
                                                 isALU = 1;
                                                target = PC+4;
                                                end
{`SPECIAL, `SLTU,    `dc5 }:begin
                                                 operation = `select_alu_sltu;
                                                 operation2 = 0;
                                                 queue = `ALUQueue;
                                                 rs = instr[`rs];
                                                 rt = instr[`rt];
```

```
                                                           rd = instr[`rd];
                                                           rd2 = 0;
                                                           immediate = 0; usesImmediate = 0;
                                                           isSyscall = 0;
                                                           isMem = 0;
                                                           isMultDiv = 0;
                                                           isCF = 0;
                                                           isALU = 1;
                                                          target = PC+4;
                                                          end
                  {`SPECIAL, `AND,     `dc5 }:begin
                                                           operation = `select_alu_and;
                                                           operation2 = 0;
                                                           queue = `ALUQueue;
                                                           rs = instr[`rs];
                                                           rt = instr[`rt];
                                                           rd = instr[`rd];
                                                           rd2 = 0;
                                                           immediate = 0; usesImmediate = 0;
                                                           isSyscall = 0;
                                                           isMem = 0;
                                                           isMultDiv = 0;
                                                           isCF = 0;
                                                           isALU = 1;
                                                          target = PC+4;
                                                          end
                  {`SPECIAL, `OR,      `dc5 }:begin
                                                           operation = `select_alu_or;
                                                           operation2 = 0;
                                                           queue = `ALUQueue;
                                                           rs = instr[`rs];
                                                           rt = instr[`rt];
                                                           rd = instr[`rd];
                                                           rd2 = 0;
                                                           immediate = 0; usesImmediate = 0;
                                                           isSyscall = 0;
                                                           isMem = 0;
                                                           isMultDiv = 0;
                                                           isCF = 0;
                                                           isALU = 1;
                                                          target = PC+4;
                                                          end
                  {`SPECIAL, `XOR,     `dc5 }:begin
                                                           operation = `select_alu_xor;
                                                           operation2 = 0;
                                                           queue = `ALUQueue;
                                                           rs = instr[`rs];
                                                           rt = instr[`rt];
                                                           rd = instr[`rd];
                                                           rd2 = 0;
                                                           immediate = 0; usesImmediate = 0;
                                                           isSyscall = 0;
                                                           isMem = 0;
                                                           isMultDiv = 0;
                                                           isCF = 0;
                                                           isALU = 1;
                                                          target = PC+4;
                                                          end
                  {`SPECIAL, `NOR,     `dc5 }:begin
                                                           operation = `select_alu_nor;
                                                           operation2 = 0;
                                                           queue = `ALUQueue;
                                                           rs = instr[`rs];
                                                           rt = instr[`rt];
                                                           rd = instr[`rd];
                                                           rd2 = 0;
                                                           immediate = 0; usesImmediate = 0;
                                                           isSyscall = 0;
                                                           isMem = 0;
                                                           isMultDiv = 0;
                                                           isCF = 0;
```

```
                                                         isALU = 1;
                                                        target = PC+4;
                                                        end
        {`SPECIAL, `SRL,     `dc5 }:begin
                                                         operation = `select_alu_srl;
                                                         operation2 = 0;
                                                         queue = `ALUQueue;
                                                         rs = 0;
                                                         rt = instr[`rt];
                                                         rd = instr[`rd];
                                                         rd2 = 0;
                                                         immediate = {16'b0,
    instr[`immediate]}; usesImmediate = 1;
                                                         isSyscall = 0;
                                                         isMem = 0;
                                                         isMultDiv = 0;
                                                         isCF = 0;
                                                         isALU = 1;
                                                        target = PC+4;
                                                        end
        {`SPECIAL, `SRA,     `dc5 }:begin
                                                         operation = `select_alu_sra;
                                                         operation2 = 0;
                                                         queue = `ALUQueue;
                                                         rs = 0;
                                                         rt = instr[`rt];
                                                         rd = instr[`rd];
                                                         rd2 = 0;
                                                         immediate = {16'b0,
    instr[`immediate]}; usesImmediate = 1;
                                                         isSyscall = 0;
                                                         isMem = 0;
                                                         isMultDiv = 0;
                                                         isCF = 0;
                                                         isALU = 1;
                                                        target = PC+4;
                                                        end
        {`SPECIAL, `SLL,     `dc5 }:begin
                                                         operation = `select_alu_sll;
                                                         operation2 = 0;
                                                         queue = `ALUQueue;
                                                         rs = 0;
                                                         rt = instr[`rt];
                                                         rd = instr[`rd];
                                                         rd2 = 0;
                                                         immediate = {16'b0,
    instr[`immediate]}; usesImmediate = 1;
                                                         isSyscall = 0;
                                                         isMem = 0;
                                                         isMultDiv = 0;
                                                         isCF = 0;
                                                         isALU = 1;
                                                        target = PC+4;
                                                        end
        {`SPECIAL, `SLLV,    `dc5 }:begin
                                                         operation = `select_alu_sll;
                                                         operation2 = 0;
                                                         queue = `ALUQueue;
                                                         rs = instr[`rs];
                                                         rt = instr[`rt];
                                                         rd = instr[`rd];
                                                         rd2 = 0;
                                                         immediate = 0; usesImmediate = 0;
                                                         isSyscall = 0;
                                                         isMem = 0;
                                                         isMultDiv = 0;
                                                         isCF = 0;
                                                         isALU = 1;
                                                        target = PC+4;
                                                        end
        {`SPECIAL, `SRLV,    `dc5 }:begin
```

```
                                        operation = `select_alu_srl;
                                        operation2 = 0;
                                        queue = `ALUQueue;
                                        rs = instr[`rs];
                                        rt = instr[`rt];
                                        rd = instr[`rd];
                                        rd2 = 0;
                                        immediate = 0; usesImmediate = 0;
                                        isSyscall = 0;
                                        isMem = 0;
                                        isMultDiv = 0;
                                        isCF = 0;
                                        isALU = 1;
                                       target = PC+4;
                                       end
{`SPECIAL, `SRAV,    `dc5 }:begin
                                        operation = `select_alu_sra;
                                        operation2 = 0;
                                        queue = `ALUQueue;
                                        rs = instr[`rs];
                                        rt = instr[`rt];
                                        rd = instr[`rd];
                                        rd2 = 0;
                                        immediate = 0; usesImmediate = 0;
                                        isSyscall = 0;
                                        isMem = 0;
                                        isMultDiv = 0;
                                        isCF = 0;
                                        isALU = 1;
                                       target = PC+4;
                                       end
{`SPECIAL, `JR,      `dc5 }:begin
                                        operation = `select_qc_jr;
                                        operation2 = 0;
                                        queue = `BRQueue;
                                        rs = instr[`rs];
                                        rt = 0;
                                        rd = 0;
                                        rd2 = 0;
                                        immediate = 0; usesImmediate = 0;
                                        isSyscall = 0;
                                        isMem = 0;
                                        isMultDiv = 0;
                                        isCF = 1;
                                        isALU = 0;
                                       target = 0;
                                       end
{`SPECIAL, `JALR,    `dc5 }:begin
                                        operation = `select_qc_jalr;
                                        operation2 = 0;
                                        queue = `BRQueue;
                                        rs = instr[`rs];
                                        rt = 0;
                                        rd = instr[`rd];
                                        rd2 = 0;
                                        immediate = PC+8; usesImmediate = 0;
                                        isSyscall = 0;
                                        isMem = 0;
                                        isMultDiv = 0;
                                        isCF = 3'b101;
                                        isALU = 0;
                                       target = 0;
                                       end
{`SPECIAL, `SYSCALL, `dc5 }:begin
                                        operation = 0;
                                        operation2 = 0;
                                        queue = 0;
                                        rs = 0;
                                        rt = 0;
                                        rd = 0;
                                        rd2 = 0;
```

```
                                                         immediate = 0; usesImmediate = 0;
                                                         isSyscall = 1;
                                                         isMem = 0;
                                                         isMultDiv = 0;
                                                         isCF = 0;
                                                         isALU = 0;
                                                         target = PC+4;
                                                        end
        {`SPECIAL, `BREAK,   `dc5 }:begin
                                                         operation = 0;
                                                         operation2 = 0;
                                                         queue = 0;
                                                         rs = 0;
                                                         rt = 0;
                                                         rd = 0;
                                                         rd2 = 0;
                                                         immediate = 0; usesImmediate = 0;
                                                         isSyscall = 0;
                                                         isMem = 0;
                                                         isMultDiv = 0;
                                                         isCF = 0;
                                                         isALU = 0;
                                                        target = PC+4;
                                                        end
        {`SPECIAL, `MULT,    `dc5 }:begin
                                                         operation = `select_alu_mult_h;
                                                         operation2 = `select_alu_mult_l;
                                                         queue = `ALUQueue;
                                                         rs = instr[`rs];
                                                         rt = instr[`rt];
                                                         rd = 32;
                                                         rd2 = 33;
                                                         immediate = 0; usesImmediate = 0;
                                                         isSyscall = 0;
                                                         isMem = 0;
                                                         isMultDiv = 1;
                                                         isCF = 0;
                                                         isALU = 1;
                                                        target = PC+4;
                                                        end
        {`SPECIAL, `MULTU,   `dc5 }:begin
                                                         operation = `select_alu_multu_h;
                                                         operation2 = `select_alu_multu_l;
                                                         queue = `ALUQueue;
                                                         rs = instr[`rs];
                                                         rt = instr[`rt];
                                                         rd = 32;
                                                         rd2 = 33;
                                                         immediate = 0; usesImmediate = 0;
                                                         isSyscall = 0;
                                                         isMem = 0;
                                                         isMultDiv = 1;
                                                         isCF = 0;
                                                         isALU = 1;
                                                        target = PC+4;
                                                        end
        {`SPECIAL, `DIV,     `dc5 }:begin
                                                         operation = `select_alu_div_h;
                                                         operation2 = `select_alu_div_l;
                                                         queue = `ALUQueue;
                                                         rs = instr[`rs];
                                                         rt = instr[`rt];
                                                         rd = 32;
                                                         rd2 = 33;
                                                         immediate = 0; usesImmediate = 0;
                                                         isSyscall = 0;
                                                         isMem = 0;
                                                         isMultDiv = 1;
                                                         isCF = 0;
                                                         isALU = 1;
                                                        target = PC+4;
                                                        end
        {`SPECIAL, `DIVU,    `dc5 }:begin
                                                         operation = `select_alu_divu_h;
                                                         operation2 = `select_alu_divu_l;
                                                         queue = `ALUQueue;
                                                         rs = instr[`rs];
                                                         rt = instr[`rt];
                                                         rd = 32;
                                                         rd2 = 33;
                                                         immediate = 0; usesImmediate = 0;
                                                         isSyscall = 0;
                                                         isMem = 0;
                                                         isMultDiv = 1;
                                                         isCF = 0;
                                                         isALU = 1;
                                                        target = PC+4;
                                                        end
        {`SPECIAL, `MFHI,    `dc5 }:begin
                                                         operation = `select_alu_mfhi;
                                                         operation2 = 0;
                                                         queue = `ALUQueue;
                                                         rs = 32;
                                                         rt = 0;
                                                         rd = instr[`rd];
                                                         rd2 = 0;
                                                         immediate = 0; usesImmediate = 0;
                                                         isSyscall = 0;
                                                         isMem = 0;
                                                         isMultDiv = 0;
                                                         isCF = 0;
                                                         isALU = 1;
                                                        target = PC+4;
                                                        end
        {`SPECIAL, `MFLO,    `dc5 }:begin
                                                         operation = `select_alu_mflo;
                                                         operation2 = 0;
                                                         queue = `ALUQueue;
                                                         rs = 33;
                                                         rt = 0;
                                                         rd = instr[`rd];
                                                         rd2 = 0;
                                                         immediate = 0; usesImmediate = 0;
                                                         isSyscall = 0;
                                                         isMem = 0;
                                                         isMultDiv = 0;
                                                         isCF = 0;
                                                         isALU = 1;
                                                        target = PC+4;
                                                        end
        {`REGIMM, `dc6, `BLTZ}:begin
                                                 operation = `select_qc_ltz;
                                                         operation2 = 0;
                                                         queue = `BRQueue;
                                                         rs = instr[`rs];
                                                         rt = 0;
                                                         rd = 0;
                                                         rd2 = 0;
                                                         immediate = 0; usesImmediate = 0;
                                                         isSyscall = 0;
                                                         isMem = 0;
                                                         isMultDiv = 0;
                                                         isCF = {1'b0,instr[15],1'b1};
                                                         isALU = 0;
                                                        target = PC4 + {{14{instr[15]}},
instr[`immediate], 2'b0};
                                                end
        {`REGIMM, `dc6, `BGEZ}:begin
                                                 operation = `select_qc_gez;
                                                         operation2 = 0;
                                                         queue = `BRQueue;
                                                         rs = instr[`rs];
```

```
                                            rt = 0;
                                            rd = 0;
                                            rd2 = 0;
                                            immediate = 0; usesImmediate = 0;
                                            isSyscall = 0;
                                            isMem = 0;
                                            isMultDiv = 0;
                                            isCF = {1'b0,instr[15],1'b1};
                                            isALU = 0;
                                            target = PC4 + {{14{instr[15]}},
instr[`immediate], 2'b0};
                                     end
       {`REGIMM, `dc6, `BGEZAL}: begin
                                            operation = `select_qc_gez;
                                            operation2 = 0;
                                            queue = `BRQueue;
                                            rs = instr[`rs];
                                            rt = 0;
                                            rd = instr[`rd];
                                            rd2 = 0;
                                            immediate = PC+8; usesImmediate = 0;
                                            isSyscall = 0;
                                            isMem = 0;
                                            isMultDiv = 0;
                                            isCF = {1'b1,instr[15],1'b1};
                                            isALU = 0;
                                           target = PC4 + {{14{instr[15]}},
instr[`immediate], 2'b0};
                                           end
       {`REGIMM, `dc6, `BLTZAL}: begin
                                            operation = `select_qc_ltz;
                                            operation2 = 0;
                                            queue = `BRQueue;
                                            rs = instr[`rs];
                                            rt = 0;
                                            rd = instr[`rd];
                                            rd2 = 0;
                                            immediate = PC+8; usesImmediate = 0;
                                            isSyscall = 0;
                                            isMem = 0;
                                            isMultDiv = 0;
                                            isCF = {1'b1,instr[15],1'b1};
                                            isALU = 0;
                                           target = PC4 + {{14{instr[15]}},
instr[`immediate], 2'b0};
                                           end
       {`ADDI,  `dc6, `dc5 }: begin
                                            operation = `select_alu_add;
                                            operation2 = 0;
                                            queue = `ALUQueue;
                                            rs = instr[`rs];
                                            rt = 0;
                                            rd = instr[`rt];
                                            rd2 = 0;
                                            immediate = {{16{instr[15]}},
instr[`immediate]}; usesImmediate = 1;
                                            isSyscall = 0;
                                            isMem = 0;
                                            isMultDiv = 0;
                                            isCF = 0;
                                            isALU = 1;
                                           target = PC+4;
                                           end
       {`ADDIU, `dc6, `dc5 }: begin
                                            operation = `select_alu_add;
                                            operation2 = 0;
                                            queue = `ALUQueue;
                                            rs = instr[`rs];
                                            rt = 0;
                                            rd = instr[`rt];
                                            rd2 = 0;
```

```
         immediate = {{16{instr[15]}},
instr[`immediate]}; usesImmediate = 1;
         isSyscall = 0;
         isMem = 0;
         isMultDiv = 0;
         isCF = 0;
         isALU = 1;
        target = PC+4;
        end
        {`SLTI,  `dc6, `dc5 }: begin
         operation = `select_alu_slt;
         operation2 = 0;
         queue = `ALUQueue;
         rs = instr[`rs];
         rt = 0;
         rd = instr[`rt];
         rd2 = 0;
         immediate = {{16{instr[15]}},
instr[`immediate]}; usesImmediate = 1;
         isSyscall = 0;
         isMem = 0;
         isMultDiv = 0;
         isCF = 0;
         isALU = 1;
        target = PC+4;
        end
        {`SLTIU, `dc6, `dc5 }: begin
         operation = `select_alu_sltu;
         operation2 = 0;
         queue = `ALUQueue;
         rs = instr[`rs];
         rt = 0;
         rd = instr[`rt];
         rd2 = 0;
         immediate = {{16{instr[15]}},
instr[`immediate]}; usesImmediate = 1;
         isSyscall = 0;
         isMem = 0;
         isMultDiv = 0;
         isCF = 0;
         isALU = 1;
        target = PC+4;
        end
        {`ANDI,  `dc6, `dc5 }: begin
         operation = `select_alu_and;
         operation2 = 0;
         queue = `ALUQueue;
         rs = instr[`rs];
         rt = 0;
         rd = instr[`rt];
         rd2 = 0;
         immediate = {16'b0,
instr[`immediate]}; usesImmediate = 1;
         isSyscall = 0;
         isMem = 0;
         isMultDiv = 0;
         isCF = 0;
         isALU = 1;
        target = PC+4;
        end
        {`ORI,   `dc6, `dc5 }: begin
         operation = `select_alu_or;
         operation2 = 0;
         queue = `ALUQueue;
         rs = instr[`rs];
         rt = 0;
         rd = instr[`rt];
         rd2 = 0;
         immediate = {16'b0,
instr[`immediate]}; usesImmediate = 1;
         isSyscall = 0;
```

```
                                                    isMem = 0;
                                                    isMultDiv = 0;
                                                    isCF = 0;
                                                    isALU = 1;
                                                   target = PC+4;
                                                   end
        {`XORI,  `dc6, `dc5 }: begin
                                                    operation = `select_alu_xor;
                                                    operation2 = 0;
                                                    queue = `ALUQueue;
                                                    rs = instr[`rs];
                                                    rt = 0;
                                                    rd = instr[`rt];
                                                    rd2 = 0;
                                                    immediate = {16'b0,
    instr[`immediate]}; usesImmediate = 1;
                                                    isSyscall = 0;
                                                    isMem = 0;
                                                    isMultDiv = 0;
                                                    isCF = 0;
                                                    isALU = 1;
                                                   target = PC+4;
                                                   end
        {`LUI,   `dc6, `dc5 }: begin
                                                    operation = `select_alu_or;
                                                    operation2 = 0;
                                                    queue = `ALUQueue;
                                                    rs = instr[`rs];
                                                    rt = 0;
                                                    rd = instr[`rt];
                                                    rd2 = 0;
                                                    immediate = {instr[`immediate],
    16'b0}; usesImmediate = 1;
                                                    isSyscall = 0;
                                                    isMem = 0;
                                                    isMultDiv = 0;
                                                    isCF = 0;
                                                    isALU = 1;
                                                   target = PC+4;
                                                   end
        {`LW,    `dc6, `dc5 }: begin
                                                    operation = `select_mem_lw;
                                                    operation2 = 0;
                                                    queue = `MEMQueue;
                                                    rs = instr[`rs];
                                                    rt = 0;
                                                    rd = instr[`rt];
                                                    rd2 = 0;
                                                    immediate = {{16{instr[15]}},
    instr[`immediate]}; usesImmediate = 1;
                                                    isSyscall = 0;
                                                    isMem = 1;
                                                    isMultDiv = 0;
                                                    isCF = 0;
                                                    isALU = 0;
                                                   target = PC+4;
                                                   end
        {`LHU,   `dc6, `dc5 }: begin
                                                    operation = `select_mem_lhu;
                                                    operation2 = 0;
                                                    queue = `MEMQueue;
                                                    rs = instr[`rs];
                                                    rt = 0;
                                                    rd = instr[`rt];
                                                    rd2 = 0;
                                                    immediate = {{16{instr[15]}},
    instr[`immediate]}; usesImmediate = 1;
                                                    isSyscall = 0;
                                                    isMem = 1;
                                                    isMultDiv = 0;
                                                    isCF = 0;
```

```
                                                              isALU = 0;
                                                             target = PC+4;
                                                             end
       {`LH,    `dc6, `dc5 }: begin
                                                              operation = `select_mem_lh;
                                                              operation2 = 0;
                                                              queue = `MEMQueue;
                                                              rs = instr[`rs];
                                                              rt = 0;
                                                              rd = instr[`rt];
                                                              rd2 = 0;
                                                              immediate = {{16{instr[15]}},
instr[`immediate]}; usesImmediate = 1;
                                                              isSyscall = 0;
                                                              isMem = 1;
                                                              isMultDiv = 0;
                                                              isCF = 0;
                                                              isALU = 0;
                                                             target = PC+4;
                                                             end
       {`LBU,   `dc6, `dc5 }: begin
                                                              operation = `select_mem_lbu;
                                                              operation2 = 0;
                                                              queue = `MEMQueue;
                                                              rs = instr[`rs];
                                                              rt = 0;
                                                              rd = instr[`rt];
                                                              rd2 = 0;
                                                              immediate = {{16{instr[15]}},
instr[`immediate]}; usesImmediate = 1;
                                                              isSyscall = 0;
                                                              isMem = 1;
                                                              isMultDiv = 0;
                                                              isCF = 0;
                                                              isALU = 0;
                                                             target = PC+4;
                                                             end
       {`LB,    `dc6, `dc5 }: begin
                                                              operation = `select_mem_lb;
                                                              operation2 = 0;
                                                              queue = `MEMQueue;
                                                              rs = instr[`rs];
                                                              rt = 0;
                                                              rd = instr[`rt];
                                                              rd2 = 0;
                                                              immediate = {{16{instr[15]}},
instr[`immediate]}; usesImmediate = 1;
                                                              isSyscall = 0;
                                                              isMem = 1;
                                                              isMultDiv = 0;
                                                              isCF = 0;
                                                              isALU = 0;
                                                             target = PC+4;
                                                             end
       {`SW,    `dc6, `dc5 }: begin
                                                              operation = `select_mem_sw;
                                                              operation2 = 0;
                                                              queue = `MEMQueue;
                                                              rs = instr[`rs];
                                                              rt = instr[`rt];
                                                              rd = 0;
                                                              rd2 = 0;
                                                              immediate = {{16{instr[15]}},
instr[`immediate]}; usesImmediate = 1;
                                                              isSyscall = 0;
                                                              isMem = 1;
                                                              isMultDiv = 0;
                                                              isCF = 0;
                                                              isALU = 0;
                                                             target = PC+4;
                                                             end
```

```
        {`SH,    `dc6, `dc5 }: begin
                                 operation = `select_mem_sh;
                                 operation2 = 0;
                                 queue = `MEMQueue;
                                 rs = instr[`rs];
                                 rt = instr[`rt];
                                 rd = 0;
                                 rd2 = 0;
                                 immediate = {{16{instr[15]}},
instr[`immediate]}; usesImmediate = 1;
                                 isSyscall = 0;
                                 isMem = 1;
                                 isMultDiv = 0;
                                 isCF = 0;
                                 isALU = 0;
                                target = PC+4;
                                end
        {`SB,    `dc6, `dc5 }: begin
                                 operation = `select_mem_sb;
                                 operation2 = 0;
                                 queue = `MEMQueue;
                                 rs = instr[`rs];
                                 rt = instr[`rt];
                                 rd = 0;
                                 rd2 = 0;
                                 immediate = {{16{instr[15]}},
instr[`immediate]}; usesImmediate = 1;
                                 isSyscall = 0;
                                 isMem = 1;
                                 isMultDiv = 0;
                                 isCF = 0;
                                 isALU = 0;
                                target = PC+4;
                                end
        {`J,     `dc6, `dc5 }: begin
                                 operation = `select_qc_j;
                                 operation2 = 0;
                                 queue = 0;
                                 rs = 0;
                                 rt = 0;
                                 rd = 0;
                                 rd2 = 0;
                                 immediate = 0; usesImmediate = 0;
                                 isSyscall = 0;
                                 isMem = 0;
                                 isMultDiv = 0;
                                 isCF = 1;
                                 isALU = 0;
                                target =
{PC4[31:28],instr[`target],2'b0};
                                end
        {`JAL,   `dc6, `dc5 }: begin
                                 operation = `select_qc_jal;
                                 operation2 = 0;
                                 queue = 0;
                                 rs = 0;
                                 rt = 0;
                                 rd = 31;
                                 rd2 = 0;
                                 immediate = PC+8; usesImmediate = 0;
                                 isSyscall = 0;
                                 isMem = 0;
                                 isMultDiv = 0;
                                 isCF = 3'b101;
                                 isALU = 0;
                                target =
{PC4[31:28],instr[`target],2'b0};
                                end
        {`BNE,   `dc6, `dc5 }: begin
                                 operation = `select_qc_ne;
                                 operation2 = 0;
```

```
                                                             queue = `BRQueue;
                                                             rs = instr[`rs];
                                                             rt = instr[`rt];
                                                             rd = 0;
                                                             rd2 = 0;
                                                             immediate = {{14{instr[15]}},
instr[`immediate], 2'b0}; usesImmediate = 0;
                                                             isSyscall = 0;
                                                             isMem = 0;
                                                             isMultDiv = 0;
                                                             isCF = {1'b0,instr[15],1'b1};
                                                             isALU = 0;
                                                            target = PC4 + {{14{instr[15]}},
instr[`immediate], 2'b0};
                                                            end
        {`BEQ,   `dc6, `dc5 }: begin
                                                             operation = `select_qc_eq;
                                                             operation2 = 0;
                                                             queue = `BRQueue;
                                                             rs = instr[`rs];
                                                             rt = instr[`rt];
                                                             rd = 0;
                                                             rd2 = 0;
                                                             immediate = {{14{instr[15]}},
instr[`immediate], 2'b0}; usesImmediate = 0;
                                                             isSyscall = 0;
                                                             isMem = 0;
                                                             isMultDiv = 0;
                                                             isCF = {1'b0,instr[15],1'b1};
                                                             isALU = 0;
                                                            target = PC4 + {{14{instr[15]}},
instr[`immediate], 2'b0};
                                                            end
        {`BLEZ,  `dc6, `dc5 }: begin
                                                             operation = `select_qc_lez;
                                                             operation2 = 0;
                                                             queue = `BRQueue;
                                                             rs = instr[`rs];
                                                             rt = 0;
                                                             rd = 0;
                                                             rd2 = 0;
                                                             immediate = {{14{instr[15]}},
instr[`immediate], 2'b0}; usesImmediate = 0;
                                                             isSyscall = 0;
                                                             isMem = 0;
                                                             isMultDiv = 0;
                                                             isCF = {1'b0,instr[15],1'b1};
                                                             isALU = 0;
                                                            target = PC4 + {{14{instr[15]}},
instr[`immediate], 2'b0};
                                                            end
        {`BGTZ,  `dc6, `dc5 }: begin
                                                             operation = `select_qc_gtz;
                                                             operation2 = 0;
                                                             queue = `BRQueue;
                                                             rs = instr[`rs];
                                                             rt = 0;
                                                             rd = 0;
                                                             rd2 = 0;
                                                             immediate = {{14{instr[15]}},
instr[`immediate], 2'b0}; usesImmediate = 0;
                                                             isSyscall = 0;
                                                             isMem = 0;
                                                             isMultDiv = 0;
                                                             isCF = {1'b0,instr[15],1'b1};
                                                             isALU = 0;
                                                            target = PC4 + {{14{instr[15]}},
instr[`immediate], 2'b0};
                                                            end
        // These are passed thru as NOPs so that real
        // program may run.
        {`LWC1,  `dc6, `dc5 }: begin
                                                             operation = 0;
                                                             operation2 = 0;
                                                             queue = 0;
                                                             rs = 0;
                                                             rt = 0;
                                                             rd = 0;
                                                             rd2 = 0;
                                                             immediate = 0; usesImmediate = 0;
                                                             isSyscall = 0;
                                                             isMem = 0;
                                                             isMultDiv = 0;
                                                             isCF = 0;
                                                             isALU = 0;
                                                            target = PC+4;
                                                            end
        {`SWC1,  `dc6, `dc5 }: begin
                                                             operation = 0;
                                                             operation2 = 0;
                                                             queue = 0;
                                                             rs = 0;
                                                             rt = 0;
                                                             rd = 0;
                                                             rd2 = 0;
                                                             immediate = 0; usesImmediate = 0;
                                                             isSyscall = 0;
                                                             isMem = 0;
                                                             isMultDiv = 0;
                                                             isCF = 0;
                                                             isALU = 0;
                                                            target = PC+4;
                                                            end
        {`COP1,  `dc6, `dc5 }: begin
                                                             operation = 0;
                                                             operation2 = 0;
                                                             queue = 0;
                                                             rs = 0;
                                                             rt = 0;
                                                             rd = 0;
                                                             rd2 = 0;
                                                             immediate = 0; usesImmediate = 0;
                                                             isSyscall = 0;
                                                             isMem = 0;
                                                             isMultDiv = 0;
                                                             isCF = 0;
                                                             isALU = 0;
                                                            target = PC+4;
                                                            end

        default:
          begin
                                                             operation = 0;
                                                             operation2 = 0;
                                                             queue = 0;
                                                             rs = 0;
                                                             rt = 0;
                                                             rd = 0;
                                                             rd2 = 0;
                                                             immediate = 0; usesImmediate = 0;
                                                             isSyscall = 0;
                                                             isMem = 0;
                                                             isMultDiv = 0;
                                                             isCF = 0;
                                                             isALU = 0;
                                                            target = PC+4;
          end
     endcase

    end
 end
```

```
endmodule

************************************
branch_compare.v

`include "mips.h"

module branch_compare(RSOperand, RTOperand, ComparisonType, Taken, Target);
input   [31:0]  RSOperand;
input   [31:0]  RTOperand;
input   [7:0]   ComparisonType;
output reg Taken;
output reg [31:0] Target;

always @(*) begin
   case (ComparisonType)
      `select_qc_ne: begin
                                       Taken = (RSOperand != RTOperand);
                                       Target = 0;
                                       end
      `select_qc_eq: begin
                                       Taken = (RSOperand == RTOperand);
                                       Target = 0;
                                       end
      `select_qc_lez:begin
                                       Taken = (RSOperand[31] == 1) | (RSOperand == 0);
                                       Target = 0;
                                       end
      `select_qc_gtz:begin
                                   Taken = (RSOperand[31] == 0) & (RSOperand != 0);
                                       Target = 0;
                                       end
      `select_qc_gez:begin
                                       Taken = (RSOperand[31] == 0);
                                       Target = 0;
                                       end
      `select_qc_ltz:begin
                                       Taken = (RSOperand[31] == 1);
                                       Target = 0;
                                       end
          `select_qc_jr: begin
                                       Target = RSOperand;
                                       Taken = 1;
                                   end
          `select_qc_jalr:begin
                                       Target = RSOperand;
                                       Taken = 1;
                                       end
      default:          begin
                                       Target = 0;
                                       Taken = 0;
                                       end
   endcase
end

endmodule

************************************
physical_register_file.v
`include "mips.h"

module physical_register_file(RESET,CLK,ReadAddr0,ReadValue0,ReadAddr1,ReadValue1,

       ReadAddr2,ReadValue2,ReadAddr3,ReadValue3,

       ReadAddr4,ReadValue4,ReadAddr5,ReadValue5,

       ReadAddr6,ReadValue6,ReadAddr7,ReadValue7,

       WriteValue0,WriteAddr0,WriteEnable0,
        WriteValue1,WriteAddr1,WriteEnable1,

        WriteValue2,WriteAddr2,WriteEnable2,

        WriteValue3,WriteAddr3,WriteEnable3);

//Read0,1 ALU0
//Read2,3 ALU1
//Read4,5 LDST
//Read6,7 Br/Jr
//Write0 ALU0, two cycle write for mult/div
//Write1 ALU1, two cycle write for mult/div
//Write2 LD
//Write3 Link, could combine with load

input RESET,CLK;
input [5:0] ReadAddr0;
output reg [31:0] ReadValue0;
input [5:0] ReadAddr1;
output reg [31:0] ReadValue1;
input [5:0] ReadAddr2;
output reg [31:0] ReadValue2;
input [5:0] ReadAddr3;
output reg [31:0] ReadValue3;
input [5:0] ReadAddr4;
output reg [31:0] ReadValue4;
input [5:0] ReadAddr5;
output reg [31:0] ReadValue5;
input [5:0] ReadAddr6;
output reg [31:0] ReadValue6;
input [5:0] ReadAddr7;
output reg [31:0] ReadValue7;
input WriteEnable0;
input [5:0] WriteAddr0;
input [31:0] WriteValue0;
input WriteEnable1;
input [5:0] WriteAddr1;
input [31:0] WriteValue1;
input WriteEnable2;
input [5:0] WriteAddr2;
input [31:0] WriteValue2;
input WriteEnable3;
input [5:0] WriteAddr3;
input [31:0] WriteValue3;

reg     [31:0]  RegFile [63:0];

always @(RegFile[ReadAddr0])begin
        ReadValue0 = RegFile[ReadAddr0];
end
always @(RegFile[ReadAddr1])begin
        ReadValue1 = RegFile[ReadAddr1];
end
always @(RegFile[ReadAddr2])begin
        ReadValue2 = RegFile[ReadAddr2];
end
always @(RegFile[ReadAddr3])begin
        ReadValue3 = RegFile[ReadAddr3];
end
always @(RegFile[ReadAddr4])begin
        ReadValue4 = RegFile[ReadAddr4];
end
always @(RegFile[ReadAddr5])begin
        ReadValue5 = RegFile[ReadAddr5];
end
always @(RegFile[ReadAddr6])begin
        ReadValue6 = RegFile[ReadAddr6];
end
always @(RegFile[ReadAddr7])begin
        ReadValue7 = RegFile[ReadAddr7];
```

```
end

integer i;

always @(negedge CLK)begin
	if(RESET)begin
		for(i=0;i<64;i=i+1)RegFile[i]=0;
	end
	else begin
		if(WriteEnable0 && WriteAddr0!=0)RegFile[WriteAddr0]=WriteValue0;
		if(WriteEnable1 && WriteAddr1!=0)RegFile[WriteAddr1]=WriteValue1;
		if(WriteEnable2 && WriteAddr2!=0)RegFile[WriteAddr2]=WriteValue2;
		if(WriteEnable3 && WriteAddr3!=0)RegFile[WriteAddr3]=WriteValue3;
	end
end

endmodule

module
ready_register_list(RESET,CLK,toBeReady0,toBeReady1,toBeReady2,toBeReady3,clearReady0,clearReady1
,clearReady2,clearReady3,ReadyList,fullFlush);
input RESET;
input CLK;
input [63:0] toBeReady0;
input [63:0] toBeReady1;
input [63:0] toBeReady2;
input [63:0] toBeReady3;
input [63:0] clearReady0;
input [63:0] clearReady1;
input [63:0] clearReady2;
input [63:0] clearReady3;
input fullFlush;
output reg [63:0] ReadyList;

always @(posedge CLK)begin
	if(RESET)begin
		ReadyList = 0;
	end
	else begin
		if(!fullFlush)ReadyList = ReadyList | toBeReady0 | toBeReady1 | toBeReady2;
		ReadyList = ReadyList | toBeReady3;//add new ready registers
		ReadyList = ReadyList & ~clearReady0 & ~clearReady1 & ~clearReady2 &
~clearReady3;//clear retired registers (or flush)
		ReadyList = ReadyList | 64'h0000000000000001;
	end
end

endmodule

*******************************
memory.v

//---------------------------------------------------------------------------
//
//  Copyright (c) 1999 Cornell University
//  Computer Systems Laboratory
//  Cornell University, Ithaca, NY 14853
//  All Rights Reserved
//
//  Permission to use, copy, modify, and distribute this software
//  and its documentation for any purpose and without fee is hereby
//  granted, provided that the above copyright notice appear in all
//  copies. Cornell University makes no representations
//  about the suitability of this software for any purpose. It is
//  provided "as is" without express or implied warranty. Export of this
//  software outside of the United States of America may require an
//  export license.
//
//  $Id: mem.v,v 1.8 2000/10/14 19:21:49 heinrich Exp $
//
//---------------------------------------------------------------------------
```

```
`include "mips.h"
`include "cache.h"

module mem (CLK, memOperation, isMem, RESET, MAR, Valid, SMDR, IaddrA, IaddrB, IaddrC, IaddrD,
Bus, Read, Write, Addr, cacheOut, IinA, IinB, IinC, IinD, Istall, Dstall, isLoad);

   input              CLK;
   input [7:0]  memOperation;
   input              isMem;
   input              RESET;
   input [31:0] MAR;
   input              Valid;
   input [31:0] SMDR;
   input [31:0] IaddrA;
   input [31:0] IaddrB;
   input [31:0] IaddrC;
   input [31:0] IaddrD;

   inout [31:0] Bus;
   reg [31:0]  BusReg;

   output             Read;
   reg                Read;
   output             Write;
   reg                Write;

   output reg [31:0] Addr;
   output [31:0] cacheOut;
   output [31:0]      IinA;
   output [31:0]      IinB;
   output [31:0]      IinC;
   output [31:0]      IinD;
   reg [31:0] preCacheOut;
   reg [`I_TAG_WIDTH-1:0] iTagA;
   reg [`I_TAG_WIDTH-1:0] iTagB;
   reg [`I_TAG_WIDTH-1:0] iTagC;
   reg [`I_TAG_WIDTH-1:0] iTagD;
   reg [`D_TAG_WIDTH-1:0] dTag;
   reg [1:0] iStateA;
   reg [1:0] iStateB;
   reg [1:0] iStateC;
   reg [1:0] iStateD;
   reg [1:0] dState;
   reg rDstall;
   reg rIstallA,rIstallB,rIstallC,rIstallD;
   reg deferred_rIstallA,deferred_rIstallB,deferred_rIstallC,deferred_rIstallD;
   reg retry;
   output reg isLoad;
   reg isStore;
   reg [`D_WO_WIDTH-1:0] offset;
   reg writeDone;
   reg readDone;
   reg [1:0] Dsize;
   reg [31:0] iCacheOutA;
   reg [31:0] iCacheOutB;
   reg [31:0] iCacheOutC;
   reg [31:0] iCacheOutD;
   reg [4:0] dcount;
   reg [4:0] icount;
   reg [31:0] num_icache_accesses;
   reg hit;
   wire [31:0] magic_number;
   assign magic_number = 32'h0043BFF4;
   wire google;
   assign google =
(IaddrA[31:5]==magic_number[31:5])|(IaddrB[31:5]==magic_number[31:5])|(IaddrC[31:5]==magic_number
[31:5])|(IaddrD[31:5]==magic_number[31:5]);

   output Dstall;
   output Istall;
   // These wires are for performing sub-word stores
   wire [4:0] sa;
   wire [31:0] mask;
   wire [31:0] value;

   // This is a good place to keep track of the cache stats
   always @(posedge CLK) begin
      // This information is for STATs only
      if (RESET) begin
                CPU.numLoads =`TICK 32'b0;
                CPU.numStores =`TICK 32'b0;
      end

      if (isLoad & ~Dstall) begin
               CPU.numLoads =`TICK CPU.numLoads + 1;
      end
      if (isStore & ~Dstall) begin
               CPU.numStores =`TICK CPU.numStores + 1;
      end
   end

   // This always block handles simple decoding of isLoad or isStore
   always @(*) begin
         if ((isMem)&&(memOperation==`select_mem_lw || memOperation==`select_mem_lh ||
memOperation==`select_mem_lhu || memOperation==`select_mem_lbu ||
memOperation==`select_mem_lb))begin
               isLoad = 1;
      end
      else begin
               isLoad = 0;
      end

         if ((isMem)&&(memOperation==`select_mem_sw || memOperation==`select_mem_sh ||
memOperation==`select_mem_sb))begin
               isStore = 1;
      end
      else begin
        isStore = 0;
      end

      // Set the data size correctly
      if (memOperation==`select_mem_sh) begin
                Dsize = `SIZE_HALF;
      end
      else if (memOperation==`select_mem_sb) begin
                Dsize = `SIZE_BYTE;
      end
      else begin
                Dsize = `SIZE_WORD;
      end
   end

   // Main I$ control signals
    assign Istall =
rIstallA|rIstallB|rIstallC|rIstallD|deferred_rIstallA|deferred_rIstallB|deferred_rIstallC|deferre
d_rIstallD ;
    assign IinA = iCacheOutA;
    assign IinB = iCacheOutB;
        assign IinC = iCacheOutC;
        assign IinD = iCacheOutD;

   // Main D$ control signals
   assign Dstall = rDstall;

   wire [31:0] preLoadDataLB, preLoadDataLH;

   assign      preLoadDataLB = (preCacheOut >> (( ~MAR & 32'h3) << 3)) & 32'hff;
   assign      preLoadDataLH = (preCacheOut >> (((~MAR >> 1 ) & 32'h1)  << 4)) & 32'hffff;
   assign
              cacheOut = (memOperation == `select_mem_lw)  ?  preCacheOut : 32'bz,
              cacheOut = (memOperation == `select_mem_lhu) ? (preCacheOut >> ((( ~MAR >> 1 ) &
32'h1) << 4)) & 32'hffff : 32'bz,
              cacheOut = (memOperation == `select_mem_lbu) ? (preCacheOut >> ((~MAR & 32'h3) <<
3)) & 32'hff : 32'bz,
              cacheOut = (memOperation == `select_mem_lb)  ? {{24{preLoadDataLB[7]}},
preLoadDataLB[7:0]} : 32'bz,
              cacheOut = (memOperation == `select_mem_lh)  ?
{{16{preLoadDataLH[15]}},preLoadDataLH[15:0]}:32'bz;



   // This always block handles the D$ cache access, we trigger on I3 because unknown MAR's
seemed to be causing problems
   // With the isStore||isLoad block we won't do more than triggering on MAR other than the if
statement
   always @(*) begin
      if(isStore || isLoad)begin
        // This should set cacheOut appropriately
        // Read tag and state
       dTag = $dcache_tag_read(MAR[`D_INDEX],0);
       dState = $dcache_state_read(MAR[`D_INDEX],0);

               //DCache Hit, tags match, and is valid
       if((dState[`D_VALID]==1'b1) && (dTag == MAR[`D_TAG]))begin
                      hit =1'b1;
                      if(isLoad)begin
                              preCacheOut =$dcache_data_read(MAR[`D_INDEX],0,MAR[`D_WO]);
                      end
           end
             //DCache Miss
           else begin
                      hit =1'b0;
                      rDstall =1'b1;

                      //Dirty and Valid, writeback to memory then read
                      if(dState == 2'b11)begin
                        dcount =`TICK 4'b0000;
                      end
                      else begin
                      //Not dirty so just read block from memory
                        dcount =`TICK 4'b1000;
                      end
           end
         end
   end

   //Retry Logic - Will always be a hit
   //Handle I$ miss before D$ miss if both missed
   //Separated from other triggered cache always blocks for easier debugging/reading
   always @(posedge retry)begin
               if(rIstallA||rIstallB||rIstallC||rIstallD)begin
                      iTagA = $icache_tag_read(IaddrA[`I_INDEX],0);
                      iStateA = $icache_state_read(IaddrA[`I_INDEX],0);
                      iTagB = $icache_tag_read(IaddrB[`I_INDEX],0);
                      iStateB = $icache_state_read(IaddrB[`I_INDEX],0);
                      iTagC = $icache_tag_read(IaddrC[`I_INDEX],0);
                      iStateC = $icache_state_read(IaddrC[`I_INDEX],0);
                      iTagD = $icache_tag_read(IaddrD[`I_INDEX],0);
                      iStateD = $icache_state_read(IaddrD[`I_INDEX],0);
                      //ICache Hit
                      if((iStateA[`I_VALID]==1'b1) && (iTagA == IaddrA[`I_TAG])) begin
                              iCacheOutA = $icache_data_read(IaddrA[`I_INDEX],0,IaddrA[`I_WO]);
                      end
                      else $finish;
                      if((iStateB[`I_VALID]==1'b1) && (iTagB == IaddrB[`I_TAG])) begin
                              iCacheOutB = $icache_data_read(IaddrB[`I_INDEX],0,IaddrB[`I_WO]);
                      end
                      else $finish;
                      if((iStateC[`I_VALID]==1'b1) && (iTagC == IaddrC[`I_TAG])) begin
                              iCacheOutC = $icache_data_read(IaddrC[`I_INDEX],0,IaddrC[`I_WO]);
                      end
```

```
                        else $finish;
                        if((iStateD[`I_VALID]==1'b1) && (iTagD == IaddrD[`I_TAG])) begin
                                iCacheOutD = $icache_data_read(IaddrD[`I_INDEX],0,IaddrD[`I_WO]);
                        end
                        else $finish;
                end
                else if(rDstall)begin
                        dTag = $dcache_tag_read(MAR[`D_INDEX],0);
                        dState = $dcache_state_read(MAR[`D_INDEX],0);
                        //DCache Hit
                                if((dState[`D_VALID]==1'b1) && (dTag == MAR[`D_TAG]))begin
                                        hit =1'b1;
                                        if(isLoad)begin
                                                preCacheOut
=$dcache_data_read(MAR[`D_INDEX],0,MAR[`D_WO]);
                                        end
                                end
                                if(deferred_rIstallA)begin
                                        rIstallA = deferred_rIstallA;
                                        deferred_rIstallA = 0;
                                end
                                if(deferred_rIstallB)begin
                                        rIstallB = deferred_rIstallB;
                                        deferred_rIstallB = 0;
                                end
                                if(deferred_rIstallC)begin
                                        rIstallC = deferred_rIstallC;
                                        deferred_rIstallC = 0;
                                end
                                if(deferred_rIstallD)begin
                                        rIstallD = deferred_rIstallD;
                                        deferred_rIstallD = 0;
                                end

                end
                retry =`TICK 0;
   end

   // This always block handles the I$ access
   always @(IaddrA or IaddrB or IaddrC or IaddrD) begin
        // This should set iCacheOut appropriately
                iTagA = $icache_tag_read(IaddrA[`I_INDEX],0);
                iStateA = $icache_state_read(IaddrA[`I_INDEX],0);
                iTagB = $icache_tag_read(IaddrB[`I_INDEX],0);
                iStateB = $icache_state_read(IaddrB[`I_INDEX],0);
                iTagC = $icache_tag_read(IaddrC[`I_INDEX],0);
                iStateC = $icache_state_read(IaddrC[`I_INDEX],0);
                iTagD = $icache_tag_read(IaddrD[`I_INDEX],0);
                iStateD = $icache_state_read(IaddrD[`I_INDEX],0);
                num_icache_accesses = num_icache_accesses + 1;
                //ICache Hit
                if((iStateA[`I_VALID]==1'b1) && (iTagA == IaddrA[`I_TAG])
                 &&(iStateB[`I_VALID]==1'b1) && (iTagB == IaddrB[`I_TAG])
                 &&(iStateC[`I_VALID]==1'b1) && (iTagC == IaddrC[`I_TAG])
                 &&(iStateD[`I_VALID]==1'b1) && (iTagD == IaddrD[`I_TAG])) begin
                        iCacheOutA = $icache_data_read(IaddrA[`I_INDEX],0,IaddrA[`I_WO]);
                        iCacheOutB = $icache_data_read(IaddrB[`I_INDEX],0,IaddrB[`I_WO]);
                        iCacheOutC = $icache_data_read(IaddrC[`I_INDEX],0,IaddrC[`I_WO]);
                        iCacheOutD = $icache_data_read(IaddrD[`I_INDEX],0,IaddrD[`I_WO]);
                end
                //ICache Miss, we don't have dirty here because you only read I$
                else begin
                        iCacheOutA =32'b0;
                        iCacheOutB =32'b0;
                        iCacheOutC =32'b0;
                        iCacheOutD =32'b0;
                        if((iStateA[`I_VALID]!=1'b1) || (iTagA != IaddrA[`I_TAG]))begin
                                if(rDstall)deferred_rIstallA = 1;
                                else rIstallA =1'b1;
                        end
                        if(((iStateB[`I_VALID]!=1'b1) || (iTagB != IaddrB[`I_TAG]))
                                &&(IaddrA[`I_INDEX]!=IaddrB[`I_INDEX])) begin
                                if(rDstall)deferred_rIstallB = 1;
                                else rIstallB =1'b1;
                        end
                        if(((iStateC[`I_VALID]!=1'b1) || (iTagC != IaddrC[`I_TAG]))
                                &&(IaddrA[`I_INDEX]!=IaddrC[`I_INDEX])
                                &&(IaddrB[`I_INDEX]!=IaddrC[`I_INDEX])) begin
                                if(rDstall)deferred_rIstallC = 1;
                                else rIstallC =1'b1;
                        end
                        if(((iStateD[`I_VALID]!=1'b1) || (iTagD != IaddrD[`I_TAG]))
                                &&(IaddrA[`I_INDEX]!=IaddrD[`I_INDEX])
                                &&(IaddrB[`I_INDEX]!=IaddrD[`I_INDEX])
                                &&(IaddrC[`I_INDEX]!=IaddrD[`I_INDEX])) begin
                                if(rDstall)deferred_rIstallD = 1;
                                else rIstallD =1'b1;
                        end
                        icount =0;
                end
   end

   //Handle All Miss State Machines Here, POSEDGE triggered
   always @(posedge CLK) begin
       //I$ miss
       if(rIstallA)begin
                //Set read and addr
                if(icount==0)begin
                        Read =1'b1;
                        Addr =IaddrA;
                end
                //Wait for valid and read in words for 8 cycles
                if(Valid && (icount < 8))begin
                        $icache_tag_write(IaddrA[`I_INDEX],0,IaddrA[`I_TAG]);
                        $icache_state_write(IaddrA[`I_INDEX],0,2'b01);
                        $icache_data_write(IaddrA[`I_INDEX],0,icount[2:0],Bus);
                end
                //After 8 Cycles retry cache read
                if(icount==8)begin
                        Read =1'b0;
                        if(rIstallB==0 && rIstallC==0 && rIstallD==0)retry =`TICK 1'b1;
                        icount =`TICK icount+1;
                        rIstallA =1'b0;
                        CPU.numIMisses =CPU.numIMisses + 1;
                        icount =`TICK 0;
                end
                //After 9 cycles we unstall and go on
                else if(icount == 9) begin

                end
                //Advance counter on valid signal
                else begin
                        if(Valid)begin
                                icount =`TICK icount+1;
                        end
                end
       end
                //I$ miss
       else if(rIstallB)begin
                //Set read and addr
                if(icount==0)begin
                        Read =1'b1;
                        Addr =IaddrB;
                end
                //Wait for valid and read in words for 8 cycles
                if(Valid && (icount < 8))begin
                        $icache_tag_write(IaddrB[`I_INDEX],0,IaddrB[`I_TAG]);
                        $icache_state_write(IaddrB[`I_INDEX],0,2'b01);
                        $icache_data_write(IaddrB[`I_INDEX],0,icount[2:0],Bus);
                end
                //After 8 Cycles retry cache read
                if(icount==8)begin
```

```
                Read =1'b0;
                if(rIstallC==0 && rIstallD==0)retry =`TICK 1'b1;
                icount =`TICK icount+1;
                rIstallB =1'b0;
                CPU.numIMisses =CPU.numIMisses + 1;
                icount =`TICK 0;
        end
        //After 9 cycles we unstall and go on
        else if(icount == 9) begin

        end
        //Advance counter on valid signal
        else begin
                if(Valid)begin
                        icount =`TICK icount+1;
                end
        end
end
else if(rIstallC)begin
        //Set read and addr
        if(icount==0)begin
                Read =1'b1;
                Addr =IaddrC;
        end
        //Wait for valid and read in words for 8 cycles
        if(Valid && (icount < 8))begin
                $icache_tag_write(IaddrC[`I_INDEX],0,IaddrC[`I_TAG]);
                $icache_state_write(IaddrC[`I_INDEX],0,2'b01);
                $icache_data_write(IaddrC[`I_INDEX],0,icount[2:0],Bus);
        end
        //After 8 Cycles retry cache read
        if(icount==8)begin
                Read =1'b0;
                if(rIstallD==0)retry =`TICK 1'b1;
                icount =`TICK icount+1;
                rIstallC =1'b0;
                CPU.numIMisses =CPU.numIMisses + 1;
                icount =`TICK 0;
        end
        //After 9 cycles we unstall and go on
        else if(icount == 9) begin

        end
        //Advance counter on valid signal
        else begin
                if(Valid)begin
                        icount =`TICK icount+1;
                end
        end
end
else if(rIstallD)begin
        //Set read and addr
        if(icount==0)begin
                Read =1'b1;
                Addr =IaddrD;
        end
        //Wait for valid and read in words for 8 cycles
        if(Valid && (icount < 8))begin
                $icache_tag_write(IaddrD[`I_INDEX],0,IaddrD[`I_TAG]);
                $icache_state_write(IaddrD[`I_INDEX],0,2'b01);
                $icache_data_write(IaddrD[`I_INDEX],0,icount[2:0],Bus);
        end
        //After 8 Cycles retry cache read
        if(icount==8)begin
                Read =1'b0;
                retry =`TICK 1'b1;
                icount =`TICK icount+1;
                rIstallD =1'b0;
                CPU.numIMisses =CPU.numIMisses + 1;
                icount =`TICK 0;
        end
```

```
            //After 9 cycles we unstall and go on
            else if(icount == 9) begin

            end
            //Advance counter on valid signal
            else begin
                    if(Valid)begin
                            icount =`TICK icount+1;
                    end
            end
     end
     //D$ miss
     else if(rDstall) begin
       //After writeback is done (if needed) read in the block, setting read and address
       if(dcount==8)begin
            Read =1'b1;
            Addr =MAR;
            Write =`TICK 1'b0;
       end
       //Writeback block from cache to memory
       if(dcount<8) begin
            Write =1'b1;
            Addr ={dTag,MAR[`D_INDEX],dcount[2:0],2'b00};
            BusReg =$dcache_data_read(MAR[`D_INDEX],0,dcount[2:0]);
            dcount =`TICK dcount+1;
       end
       //Read block from memory to cache
       else if(Valid && (dcount < 16)) begin
            $dcache_tag_write(MAR[`D_INDEX],0,MAR[`D_TAG]);
            $dcache_state_write(MAR[`D_INDEX],0,2'b01);
            $dcache_data_write(MAR[`D_INDEX],0,dcount[2:0],Bus);
             dcount =`TICK dcount+1;
       end
       //Reread data after its updated
       else if(dcount == 16) begin
            Read =1'b0;
            retry =`TICK 1'b1;
            dcount =`TICK dcount+1;
            rDstall =`TICK 1'b0;
            CPU.numDMisses =CPU.numDMisses + 1;
            dcount =`TICK 0;
       end
       //Unstall and continue pipe
       else if(dcount == 17) begin

        end

     end
end

//Shift Amount
assign sa = (Dsize == `SIZE_BYTE) ? (( ~MAR & 32'h3) << 3) : 5'bz,
       sa = (Dsize == `SIZE_HALF) ? (((~MAR >> 1) & 32'h1) << 4) : 5'bz,
       sa = (Dsize == `SIZE_WORD) ? 0 : 5'bz;
//Data Mask
assign
       mask = (Dsize == `SIZE_BYTE) ? (32'hff << sa) : 32'bz,
       mask = (Dsize == `SIZE_HALF) ? (32'hffff << sa) : 32'bz,
       mask = (Dsize == `SIZE_WORD) ? 0 : 32'bz;

// Assume that stores write the data array on the negedge of the clock
always @(negedge CLK) begin
   // This code is complete and it works!  Note how it uses the
   // cache PLI calls for both reads and writes
   if (hit & isStore) begin
      // Need to write the data array and the Dirty bit
      // Handle sub-word stores here
      if (Dsize == `SIZE_WORD) begin
         $dcache_data_write(MAR[`D_INDEX], 0, MAR[`D_WO], SMDR);
      end
      else if (Dsize == `SIZE_HALF) begin
```

```
           $dcache_data_write(MAR[`D_INDEX], 0, MAR[`D_WO],
($dcache_data_read(MAR[`D_INDEX],0,MAR[`D_WO]) & ~mask)|((SMDR & 32'hffff) << sa));
        end
        else if (Dsize == `SIZE_BYTE) begin
           $dcache_data_write(MAR[`D_INDEX], 0, MAR[`D_WO],
($dcache_data_read(MAR[`D_INDEX],0,MAR[`D_WO]) & ~mask)|((SMDR & 32'hff) << sa));
        end
        // Write Dirty and Valid bits
        $dcache_state_write(MAR[`D_INDEX], 0, 2'b11);
      end
   end

   // This always block handles reset
   // We moved miss state machines out of this block for easier reading/debugging
   always @(posedge CLK) begin
      if (RESET) begin
              offset =`TICK 0;
              retry =`TICK 1'b0;
              writeDone =`TICK 1'b0;
              readDone =`TICK 1'b0;
              CPU.numDMisses =`TICK 32'b0;
              CPU.numIMisses =`TICK 32'b0;
              Read = 1'b0;
              Write = 1'b0;
              //Addr = 32'b0;
              rDstall = 0;
              icount = 0;
              dcount = 0;
              rIstallA = 1;
              rIstallB = 0;
              rIstallC = 0;
              rIstallD = 0;
              deferred_rIstallA = 0;
              deferred_rIstallB = 0;
              deferred_rIstallC = 0;
              deferred_rIstallD = 0;
              isStore = 0;
              isLoad = 0;
              preCacheOut = 0;
              num_icache_accesses = 0;
      end
   end // always @ (posedge CLK)

   //Initialize some signals
   initial begin
       Read = 1'b0;
       Write = 1'b0;
       Addr = 32'b0;
       rDstall = 0;
       icount = 0;
       dcount = 0;
       rIstallA = 0;
       rIstallB = 0;
       rIstallC = 0;
       rIstallD = 0;
       isStore = 0;
       isLoad = 0;
   end

   // Only drive Bus during write back
   assign Bus = (Write) ? BusReg : 32'bz;


//----------------------------------------------------------------------

endmodule

***************************************
issue_queue.v
`include "mips.h"
```

```
module issue_queue(RESET,CLK,fullFlush,maxIssue,queueNum,readyFlags,

        inOperation0,inImmediate0,inRT0,inRS0,inRD0,inQueue0,inROB0,inUImm0,

        inOperation1,inImmediate1,inRT1,inRS1,inRD1,inQueue1,inROB1,inUImm1,

        inOperation2,inImmediate2,inRT2,inRS2,inRD2,inQueue2,inROB2,inUImm2,

        inOperation3,inImmediate3,inRT3,inRS3,inRD3,inQueue3,inROB3,inUImm3,

        outOperation0,outImmediate0,outRT0,outRS0,outRD0,outROB0,outUImm0,

        outOperation1,outImmediate1,outRT1,outRS1,outRD1,outROB1,outUImm1,
                                      outReadyFlag0,outReadyFlag1,QueueTooFull,numIssued);

input RESET,CLK,fullFlush;
input [1:0] maxIssue,queueNum;
input [63:0] readyFlags;
output reg [63:0] outReadyFlag0;
output reg [63:0] outReadyFlag1;
input [7:0] inOperation0,inOperation1,inOperation2,inOperation3;
input [31:0] inImmediate0,inImmediate1,inImmediate2,inImmediate3;
input [4:0] inROB0,inROB1,inROB2,inROB3;
input [5:0] inRT0,inRT1,inRT2,inRT3,inRS0,inRS1,inRS2,inRS3,inRD0,inRD1,inRD2,inRD3;
input [1:0] inQueue0,inQueue1,inQueue2,inQueue3;
input inUImm0,inUImm1,inUImm2,inUImm3;
output reg outUImm0,outUImm1;
output reg[7:0] outOperation0,outOperation1;
output reg[31:0] outImmediate0,outImmediate1;
output reg[5:0] outRT0,outRT1,outRS0,outRS1,outRD0,outRD1;
output reg[4:0] outROB0,outROB1;
output reg QueueTooFull;

reg [31:0] max_fullness;
reg [31:0] total_occupancy;

reg [7:0] Operation [31:0];
reg [31:0] Immediate [31:0];
reg Valid [31:0];
reg [5:0] RT [31:0];
reg [5:0] RS [31:0];
reg [5:0] RD [31:0];
reg [4:0] ROB [31:0];
reg usesImm[31:0];
output reg [1:0] numIssued;
reg Inserted;
reg [5:0] FreeSlots;
//reg [31:0] wtfRS,wtfRT,wtfAND;
wire wtfRS,wtfRT,wtfAND;
integer i;

reg [31:0] entryReady;
always @(*)begin
for(i=0;i<32;i=i+1)entryReady[i] = Valid[i] & ((readyFlags>>RS[i])&1'b1) &
((readyFlags>>RT[i])&1'b1);
end

assign wtfRS = ((readyFlags>>RS[0])&1'b1);
assign wtfRT = ((readyFlags>>RT[0])&1'b1);
assign wtfAND = wtfRS & wtfRT;

always @(posedge CLK)begin
               //compact issue queue
        for(i=0;i<31;i=i+1)begin
               if((Valid[i]==0) && (Valid[i+1]==1))begin
                      //$display("compacting %d to %d",i+1,i);
                      Valid[i]=1;
                      Operation[i] = Operation[i+1];
                      Immediate[i] = Immediate[i+1];
                      RT[i] = RT[i+1];
                      RS[i] = RS[i+1];
```

```
                        RD[i] = RD[i+1];
                        ROB[i] = ROB[i+1];
                        usesImm[i] = usesImm[i+1];
                        Valid[i+1]=0;
                end
        end
end

always @(negedge CLK)begin
        if(RESET || fullFlush)begin
                for(i=0;i<32;i=i+1)Valid[i]=0;
                FreeSlots = 32;
                numIssued = 0;
                outReadyFlag0 = 0;
                outReadyFlag1 = 0;
                outOperation0 = 0;
                outImmediate0 = 0;
                outRT0 = 0;
                outRS0 = 0;
                outRD0 = 0;
                outROB0 = 0;
                outUImm0 = 0;
                outReadyFlag0 = 0;
                outOperation1 = 0;
                outImmediate1 = 0;
                outRT1 = 0;
                outRS1 = 0;
                outRD1 = 0;
                outUImm1 = 0;
                outROB1 = 0;
                outReadyFlag1 = 0;
                total_occupancy = 0;
                max_fullness = 0;
        end
        else begin


                //insert instructions
                Inserted = 0;
                for(i=0;i<32;i=i+1)begin
                        if((Valid[i]==0) && (Inserted==0) && (inOperation0!=0) &&
(inQueue0==queueNum))begin
                                Valid[i]=1;
                                Inserted=1;
                                Operation[i] = inOperation0;
                                Immediate[i] = inImmediate0;
                                RT[i] = inRT0;
                                RS[i] = inRS0;
                                RD[i] = inRD0;
                                ROB[i] = inROB0;
                                usesImm[i] = inUImm0;
                                FreeSlots = FreeSlots - 1;
                        end
                end

                Inserted = 0;
                for(i=0;i<32;i=i+1)begin
                        if((Valid[i]==0) && (Inserted==0) && (inOperation1!=0) &&
(inQueue1==queueNum))begin
                                Valid[i]=1;
                                Inserted=1;
                                Operation[i] = inOperation1;
                                Immediate[i] = inImmediate1;
                                RT[i] = inRT1;
                                RS[i] = inRS1;
                                RD[i] = inRD1;
                                ROB[i] = inROB1;
                                usesImm[i] = inUImm1;
                                FreeSlots = FreeSlots - 1;
                        end
                end
```

```
		Inserted = 0;
		for(i=0;i<32;i=i+1)begin
			if((Valid[i]==0) && (Inserted==0) && (inOperation2!=0) &&
(inQueue2==queueNum))begin
				Valid[i]=1;
				Inserted=1;
				Operation[i] = inOperation2;
				Immediate[i] = inImmediate2;
				RT[i] = inRT2;
				RS[i] = inRS2;
				RD[i] = inRD2;
				ROB[i] = inROB2;
				usesImm[i] = inUImm2;
				FreeSlots = FreeSlots - 1;
			end
		end

		Inserted = 0;
		for(i=0;i<32;i=i+1)begin
			if((Valid[i]==0) && (Inserted==0) && (inOperation3!=0) &&
(inQueue3==queueNum))begin
				Valid[i]=1;
				Inserted=1;
				Operation[i] = inOperation3;
				Immediate[i] = inImmediate3;
				RT[i] = inRT3;
				RS[i] = inRS3;
				RD[i] = inRD3;
				ROB[i] = inROB3;
				usesImm[i] = inUImm3;
				FreeSlots = FreeSlots - 1;
			end
		end

		total_occupancy = total_occupancy + (32-FreeSlots);
		if((32-FreeSlots)>max_fullness)max_fullness = (32-FreeSlots);

		//find next ones to issue
		numIssued = 0;
		outReadyFlag0 = 0;
		outReadyFlag1 = 0;
		outOperation0 = 0;
		outImmediate0 = 0;
		outRT0 = 0;
		outRS0 = 0;
		outRD0 = 0;
		outROB0 = 0;
		outUImm0 = 0;
		outReadyFlag0 = 0;
		outOperation1 = 0;
		outImmediate1 = 0;
		outRT1 = 0;
		outRS1 = 0;
		outRD1 = 0;
		outUImm1 = 0;
		outROB1 = 0;
		outReadyFlag1 = 0;
		for(i=0;i<32;i=i+1)begin
			if(entryReady[i] && numIssued < maxIssue)begin
				if(numIssued == 0)begin
					outOperation0 = Operation[i];
					outImmediate0 = Immediate[i];
					outRT0 = RT[i];
					outRS0 = RS[i];
					outRD0 = RD[i];
					outROB0 = ROB[i];
					outUImm0 = usesImm[i];
					outReadyFlag0 = (1'b1 << RD[i]);
					numIssued = 1;
					FreeSlots = FreeSlots + 1;
```

```
                                        Valid[i] = 0;
                                end
                                else begin
                                        outOperation1 = Operation[i];
                                        outImmediate1 = Immediate[i];
                                        outRT1 = RT[i];
                                        outRS1 = RS[i];
                                        outRD1 = RD[i];
                                        outUImm1 = usesImm[i];
                                        outROB1 = ROB[i];
                                        outReadyFlag1 = (1'b1 << RD[i]);
                                        numIssued = 2;

                                        FreeSlots = FreeSlots + 1;
                                        Valid[i] = 0;
                                end
                        end
                end

        end
        QueueTooFull = (FreeSlots < 4);
end

endmodule

**********************************
addr_calc.v
`include "mips.h"

module addr_calc(RegVal, Immediate, EffAddr);
input [31:0] RegVal;
input [31:0] Immediate;
output reg [31:0] EffAddr;

always @(*) begin
        EffAddr = RegVal + Immediate;
end

endmodule

***********************************
alu.v
`include "mips.h"

module alu(RSOperand, RTOperand, Immediate, UseImmediate, ALUOp, ALUout);
input [31:0] RSOperand;
input [31:0] RTOperand;
input [31:0] Immediate;
input UseImmediate;
input [7:0]    ALUOp;
output reg [31:0] ALUout;
reg    [31:0] Temp;
reg [4:0] ShiftAmount;
wire [31:0]
signed_div,signed_div_t,unsigned_div,signed_rem,signed_rem_t,unsigned_rem,signed_RS,signed_RT;
wire [32:0] unsigned_RS,unsigned_RT;
wire [63:0] signed_mult,signed_mult_t;
wire [63:0] unsigned_mult;
wire signofRS,signofRT;

assign signofRS = RSOperand[31];
assign signofRT = RTOperand[31];

assign  unsigned_RS = {1'b0,RSOperand};
assign  unsigned_RT = {1'b0,RTOperand};
assign  signed_RS = (signofRS)?(~RSOperand)+1:RSOperand;
assign  signed_RT = (signofRT)?(~RTOperand)+1:RTOperand;

assign signed_mult_t = signed_RS * signed_RT;
assign unsigned_mult = unsigned_RS * unsigned_RT;
assign unsigned_div = {1'b0,RSOperand}/{1'b0,RTOperand};
assign signed_div_t = {signed_RS}/{signed_RT};
assign unsigned_rem = {1'b0,RSOperand}%{1'b0,RTOperand};
assign signed_rem_t = {signed_RS}%{signed_RT};
assign signed_mult = (signofRS!=signofRT)?(~signed_mult_t) + 1:signed_mult_t;
assign signed_div = (signofRS!=signofRT)?(~signed_div_t) + 1:signed_div_t;
assign signed_rem = (signofRS)?(~signed_rem_t) + 1:signed_rem_t;

always @(*) begin
   Temp = (UseImmediate) ? Immediate : RTOperand;
   ShiftAmount = (UseImmediate) ? Immediate[10:6] : RSOperand[4:0];

   case (ALUOp)
      `select_alu_add:  ALUout = RSOperand + Temp;
      `select_alu_and: ALUout = RSOperand & Temp;
      `select_alu_xor: ALUout = RSOperand ^ Temp;
      `select_alu_or:   ALUout = RSOperand | Temp;
      `select_alu_nor: ALUout = ~(RSOperand | Temp);
      `select_alu_sub: ALUout = RSOperand - Temp;
      `select_alu_sltu:       ALUout = ({1'b0,RSOperand} < {1'b0,Temp}) ? 1 : 0;
      `select_alu_slt: ALUout = (RSOperand[31] != Temp[31]) ? RSOperand[31] : ((RSOperand < Temp)
? 1 : 0);
      `select_alu_sra:  ALUout = {{32{RTOperand[31]}},RTOperand} >> ShiftAmount;
      `select_alu_srl:  ALUout = RTOperand >> ShiftAmount;
      `select_alu_sll:  ALUout = RTOperand << ShiftAmount;
         `select_alu_mult_h: ALUout = signed_mult[63:32];
         `select_alu_mult_l: ALUout = signed_mult[31:0];
         `select_alu_multu_h: ALUout = unsigned_mult[63:32];
         `select_alu_multu_l: ALUout = unsigned_mult[31:0];
         `select_alu_div_h: ALUout = signed_rem;
      `select_alu_div_l: ALUout = signed_div;
         `select_alu_divu_h: ALUout = unsigned_rem;
         `select_alu_divu_l: ALUout = unsigned_div;
         `select_alu_mfhi: ALUout = RSOperand;
         `select_alu_mflo: ALUout = RSOperand;
     default:          ALUout = 0;
   endcase

  end

endmodule

**************************************
forwarding.v
`include "mips.h"

module
forwarding(rfValue,rfAddr,wb0Value,wb0Addr,wb1Value,wb1Addr,wb2Value,wb2Addr,wb3Value,wb3Addr,out
Value);
input [5:0] rfAddr,wb0Addr,wb1Addr,wb2Addr,wb3Addr;
input [31:0] rfValue,wb0Value,wb1Value,wb2Value,wb3Value;
output reg [31:0] outValue;

always @(*)begin
        if(rfAddr == 0)begin
                outValue = 0;
        end
        else begin
                if(rfAddr == wb0Addr)begin
                        outValue = wb0Value;
                end
                else if(rfAddr == wb1Addr)begin
                        outValue = wb1Value;
                end
                else if(rfAddr == wb2Addr)begin
                        outValue = wb2Value;
                end
                else if(rfAddr == wb3Addr)begin
                        outValue = wb3Value;
                end
                else begin
                        outValue = rfValue;
```

```
               end
       end
end

endmodule

*********************************
mips.v
//----------------------------------------------------------------------------
//
//  Copyright (c) 1999 Cornell University
//  Computer Systems Laboratory
//  Cornell University, Ithaca, NY 14853
//  All Rights Reserved
//
//  Permission to use, copy, modify, and distribute this software
//  and its documentation for any purpose and without fee is hereby
//  granted, provided that the above copyright notice appear in all
//  copies. Cornell University makes no representations
//  about the suitability of this software for any purpose. It is
//  provided "as is" without express or implied warranty. Export of this
//  software outside of the United States of America may require an
//  export license.
//
//  $Id: mips.v,v 1.6 1999/10/19 03:39:05 heinrich Exp $
//
//----------------------------------------------------------------------------

`include "mips.h"
`include "cache.h"

`define MEM_DELAY      #162

module TOP;

   wire [31:0]        Bus;
   reg [31:0]  BusReg;

   reg         RESET, CLK;

   wire [31:0]        Addr;                  // Address Bus between cpu & memory
   wire        Read;                  // Bus Read
   wire        Write;                 // Bus Write
   reg         Valid;                 // Valid signal for cache fill

   reg         startRead;
   reg [`D_WO_WIDTH-1:0] currentWord;
   reg                  scheduled;

   // Create a 50% duty cycle clock
   initial begin
      CLK = 1;
      forever begin
         `PHASE
           CLK = 0;
         `PHASE
           CLK = 1;
      end
   end



   // This holds reset long enough to clear all the machine state
   initial begin
      RESET = 1;
      #86 RESET = 0;
   end

   // This is the R10000
   cpu  CPU
     (
   .CLK     (CLK),
   .RESET (RESET),
   .Bus     (Bus),
   .Addr    (Addr),
   .Write   (Write),
   .Read    (Read),
   .Valid   (Valid)
   );


// This block is a good place to put $monitor or $dump statements
initial
  begin
            $seed_mm;

            // I$ and D$ creation.
            // The args are:
            // associativity, linesize, number of lines

            $create_icache(1, 32, 256);
            $create_dcache(1, 32, 256);

  end

// Memory system
// The interface here is simple:
//
// Writes:
//
// At the posedge of the clock, if the Write signal is asserted,
// store the word on the Bus to memory at the address on the Addr bus.
//
// Reads:
//
// "Reads" are any kind of cache fill from the processor (note these
// could be triggered by a load miss or a store miss, it doesn't matter)
// At the posedge of the clock, if the Read signal is sensed the scheduled
// signal is set, which indicates that a memory request will be served.
// At the same time, the startRead signal is scheduled to be asserted
// MEM_DELAY ticks later. When startRead is high and Read signal is still set,
// the memory system places the word at Addr (which is still being driven
// by the cpu) onto the Bus. For length MAX_OFFSET+1 cycles, the memory system
// increments the local Addr by 4 and drives the next word on the Bus.
// The Valid line is asserted as long as the memory system is producing
// Valid data. After the last word in the cache line is driven,
// the Valid line and scheduled signal are de-asserted. At that time
// (when the read of all values is complete), the Read signal must also be
// set low.

always @(BusReg)begin
     //$display("bbb %b",BusReg);
end

always @(posedge CLK) begin
   if (RESET) begin
             Valid       <= `TICK 1'b0;
             currentWord <= `TICK 0;
             startRead   <= `TICK 1'b0;
             scheduled   <= `TICK 1'b0;
   end

   if (Read & !scheduled) begin
             startRead <= `MEM_DELAY 1'b1;
             scheduled <= `TICK 1'b1;
   end

   if (startRead) begin
             if(Read) begin
         // Note that this implies a line size of 4 words.  It is also
         // what dictates that the I$ and the D$ have the same line size.
         // You can change the line size in the cache creation statements
```

```
            // above, but you must also change the following line, and
            // the defines in cache.h
            BusReg <= `TICK $load_mm({Addr[31:5], currentWord[2:0], 2'b0});
            Valid  <= `TICK 1;
            if (currentWord != `MAX_OFFSET) begin
                      currentWord <= `TICK currentWord + 1;
            end
            else begin
                      startRead <= `TICK 1'b0;
            end
                end
                else begin
            Valid       <= `TICK 1'b0;
            scheduled   <= `TICK 1'b0;
            startRead   <= `TICK 1'b0;
            currentWord <= `TICK 0;
                end
      end
      else begin
                Valid <= `TICK 1'b0;
                if(currentWord != 0) begin
            scheduled   <= `TICK 1'b0;
            currentWord <= `TICK 0;
                end
      end

      // Handle writes to the memory system.  Partial word writes are now
      // handled in the cache!
      if (Write) begin
                //$display("Time: %d",$time);
                $store_mm(Addr,Bus);
      end
   end // always @ (posedge CLK)
   // Drive Bus with BusReg unless a write is in progress
   assign Bus =(~Write) ? BusReg : 32'bz;

   //-----------------------------------------------------------------------

endmodule

****************************************
mips.h
//-*-mode:verilog-*-------------------------------------------------------
//
//  Copyright (c) 1999 Cornell University
//  Computer Systems Laboratory
//  Cornell University, Ithaca, NY 14853
//  All Rights Reserved
//
//  Permission to use, copy, modify, and distribute this software
//  and its documentation for any purpose and without fee is hereby
//  granted, provided that the above copyright notice appear in all
//  copies. Cornell University makes no representations
//  about the suitability of this software for any purpose. It is
//  provided "as is" without express or implied warranty. Export of this
//  software outside of the United States of America may require an
//  export license.
//
//  $Id: mips.h,v 1.2 1999/10/18 17:58:51 heinrich Exp $
//
//-----------------------------------------------------------------------

// Clock parameters
`define cycle                 10
`define phase                 5
`define TICK                  #2
`define CYCLE                 #`cycle
`define PHASE                 #`phase

/*-------------------------------------------------------------------------------------------------
---
```

```
Instruction Fields
-------------------------------------------------------------------------------------------------
-*/

`define op                      31:26   // 6-bit operation code
`define rs                      25:21   // 5-bit source register specifier
`define rt                      20:16   // 5-bit source/dest register spec or sub opcode
`define immediate               15:0    // 16-bit immediate, branch or address disp
`define target                  25:0    // 26-bit jump target address
`define rd                      15:11   // 5-bit destination register specifier
`define sa                      10:6    // 5-bit shift amount
`define function                5:0     // 6-bit function field
`define sub                     25:21   // 5-bit sub-operation code
`define coprocessor             20:0    // 21-bit coprocessor-specific field


/*-----------------------------------------------------------------------------------------------
--
Symbolic Register Names for Hardware
-------------------------------------------------------------------------------------------------
-*/

`define r0                      5'b00000
`define r1                      5'b00001
`define r2                      5'b00010
`define r3                      5'b00011
`define r4                      5'b00100
`define r5                      5'b00101
`define r6                      5'b00110
`define r7                      5'b00111
`define r8                      5'b01000
`define r9                      5'b01001
`define r10                     5'b01010
`define r11                     5'b01011
`define r12                     5'b01100
`define r13                     5'b01101
`define r14                     5'b01110
`define r15                     5'b01111
`define r16                     5'b10000
`define r17                     5'b10001
`define r18                     5'b10010
`define r19                     5'b10011
`define r20                     5'b10100
`define r21                     5'b10101
`define r22                     5'b10110
`define r23                     5'b10111
`define r24                     5'b11000
`define r25                     5'b11001
`define r26                     5'b11010
`define r27                     5'b11011
`define r28                     5'b11100
`define r29                     5'b11101
`define r30                     5'b11110
`define r31                     5'b11111

/*-----------------------------------------------------------------------------------------------
----
Symbolic Register Names for Assembler and Compiler
-------------------------------------------------------------------------------------------------
--*/

`define zero                    5'b00000       // Read only zero value
`define at                      5'b00001       // Assembler temporary
`define v0                      5'b00010       // Integer function value
`define v1                      5'b00011
`define a0                      5'b00100       // Parameters
`define a1                      5'b00101
`define a2                      5'b00110
`define a3                      5'b00111
`define t0                      5'b01000       // not preserved by subroutines
`define t1                      5'b01001
```

```
`define t2                      5'b01010
`define t3                      5'b01011
`define t4                      5'b01100
`define t5                      5'b01101
`define t6                      5'b01110
`define t7                      5'b01111
`define s0                      5'b10000       // preserved by subroutines
`define s1                      5'b10001
`define s2                      5'b10010
`define s3                      5'b10011
`define s4                      5'b10100
`define s5                      5'b10101
`define s6                      5'b10110
`define s7                      5'b10111
`define t8                      5'b11000       // preserved by subroutines
`define t9                      5'b11001
`define k0                      5'b11010       // Kernel
`define k1                      5'b11011
`define gp                      5'b11100       // Global pointer
`define sp                      5'b11101       // Stack pointer
`define s8                      5'b11110       // preserved by subroutines
`define ra                      5'b11111       // Link register

/*-----------------------------------------------------------------------------------------------
---
Opcode Assignments for `op Operations
-----------------------------------------------------------------------------------------------
--*/

`define SPECIAL                 6'b000000
`define REGIMM                  6'b000001
`define J                       6'b000010
`define JAL                     6'b000011
`define BEQ                     6'b000100
`define BNE                     6'b000101
`define BLEZ                    6'b000110
`define BGTZ                    6'b000111

`define ADDI                    6'b001000
`define ADDIU                   6'b001001
`define SLTI                    6'b001010
`define SLTIU                   6'b001011
`define ANDI                    6'b001100
`define ORI                     6'b001101
`define XORI                    6'b001110
`define LUI                     6'b001111

`define COP0                    6'b010000
`define COP1                    6'b010001
`define COP2                    6'b010010
`define COP3                    6'b010011
`define BEQL                    6'b010100
`define BNEL                    6'b010101
`define BLEZL                   6'b010110
`define BGTZL                   6'b010111

`define LB                      6'b100000
`define LH                      6'b100001
`define LWL                     6'b100010
`define LW                      6'b100011
`define LBU                     6'b100100
`define LHU                     6'b100101
`define LWR                     6'b100110

`define SB                      6'b101000
`define SH                      6'b101001
`define SWL                     6'b101010
`define SW                      6'b101011

`define SWR                     6'b101110
`define CACHE                   6'b101111
```

```
`define LL                     6'b110000
`define LWC1                   6'b110001
`define LWC2                   6'b110010
`define LWC3                   6'b110011

`define LDC1                   6'b110101
`define LDC2                   6'b110110
`define LDC3                   6'b110111

`define SC                     6'b111000
`define SWC1                   6'b111001
`define SWC2                   6'b111010
`define SWC3                   6'b111011

`define SDC1                   6'b111101
`define SDC2                   6'b111110
`define SDC3                   6'b111111


/*------------------------------------------------------------------------------
Opcode Assignments for `SPECIAL function Operations
------------------------------------------------------------------------------*/

`define SLL                    6'b000000
`define SRL                    6'b000010
`define SRA                    6'b000011
`define SLLV                   6'b000100
`define SRLV                   6'b000110
`define SRAV                   6'b000111

`define JR                     6'b001000
`define JALR                   6'b001001

`define SYSCALL                6'b001100
`define BREAK                  6'b001101

`define MFHI                   6'b010000
`define MTHI                   6'b010001
`define MFLO                   6'b010010
`define MTLO                   6'b010011

`define MULT                   6'b011000
`define MULTU                  6'b011001
`define DIV                    6'b011010
`define DIVU                   6'b011011

`define ADD                    6'b100000
`define ADDU                   6'b100001
`define SUB                    6'b100010
`define SUBU                   6'b100011
`define AND                    6'b100100
`define OR                     6'b100101
`define XOR                    6'b100110
`define NOR                    6'b100111

`define SLT                    6'b101010
`define SLTU                   6'b101011

`define TGE                    6'b110000
`define TGEU                   6'b110001
`define TLT                    6'b110010
`define TLTU                   6'b110011
`define TEQ                    6'b110100

`define TNE                    6'b110110

/*-----------------------------------------------------------------------------
Opcode Assignments for `REGIMM rt Operations
------------------------------------------------------------------------------*/
```

```
`define BLTZ                    5'b00000
`define BGEZ                    5'b00001
`define BLTZL                   5'b00010
`define BGEZL                   5'b00011

`define TGEI                    5'b01000
`define TGEIU                   5'b01001
`define TLTI                    5'b01010
`define TLTIU                   5'b01011
`define TEQI                    5'b01100

`define TNEI                    5'b01110

`define BLTZAL                  5'b10000
`define BGEZAL                  5'b10001
`define BLTZALL                 5'b10010
`define BGEZALL                 5'b10011

/*----------------------------------------------------------------------------
Opcode Assignments for `COPz rs Operations
-----------------------------------------------------------------------------*/

`define MF                      5'b00000
`define CF                      5'b00010
`define MT                      5'b00100
`define CT                      5'b00110

`define BC                      5'b01000

/*-----------------------------------------------------------------------------------------------
----
Opcode Assignments for `COPz rt Operations
-------------------------------------------------------------------------------------------------
--*/

`define BCF                     5'b00000
`define BCT                     5'b00001
`define BCFL                    5'b00010
`define BCTL                    5'b00011

/*-----------------------------------------------------------------------------------------------
----
Encoded data path controls
-------------------------------------------------------------------------------------------------
--*/
`define select_pc_pc    5'b00000
`define select_pc_inc           5'b11110       // Next PC value select
`define select_pc_add           5'b11101
`define select_pc_jump          5'b11011
`define select_pc_vector        5'b10111
`define select_pc_register      5'b01111

`define select_bp_reg           3'b000
`define select_bp_stage3A       3'b001
`define select_bp_stage4A       3'b010
`define select_bp_stage3B       3'b101
`define select_bp_stage4B       3'b110

`define instr_sel_new   2'b00
`define instr_sel_olda  2'b01
`define instr_sel_oldb  2'b10
`define instr_sel_nop   2'b11

`define instr_sel_swizzle   2'b00
`define instr_sel_self      2'b01

`define pc_cf       2'b11
`define plus_8  2'b10
`define plus_4  2'b01
`define no_change 2'b00
```

```
// Decoded ALU operation select (ALUsel) signals
`define select_alu_add        8'b00000001
`define select_alu_and        8'b00000010
`define select_alu_xor        8'b00000100
`define select_alu_or         8'b00001000
`define select_alu_nor        8'b00010000
`define select_alu_sub        8'b00100001
`define select_alu_sltu               8'b01000000
`define select_alu_slt        8'b01100000
`define select_alu_shift      8'b10000000
`define select_alu_sra        8'b10000001
`define select_alu_srl        8'b10000010
`define select_alu_sll        8'b10000100
`define select_alu_mult_h     8'b10001000
`define select_alu_mult_l     8'b10010000
`define select_alu_multu_h    8'b10100000
`define select_alu_multu_l    8'b11000000
`define select_alu_div_h      8'b11000001
`define select_alu_div_l      8'b11000010
`define select_alu_divu_h     8'b11000100
`define select_alu_divu_l     8'b11001000
`define select_alu_mfhi               8'b11010000
`define select_alu_mflo               8'b11100000

`define select_mem_lw         8'b00000001
`define select_mem_lhu        8'b00000010
`define select_mem_lh         8'b00000100
`define select_mem_lbu        8'b00001000
`define select_mem_lb         8'b00010000
`define select_mem_sw         8'b00100000
`define select_mem_sh         8'b01000000
`define select_mem_sb         8'b10000000

`define ALUQueue                      2'b01
`define MEMQueue                      2'b10
`define BRQueue                       2'b11

// Decoded quick compare condition select (QCsel) signals
`define select_qc_ne          8'b00000001
`define select_qc_eq          8'b00000010
`define select_qc_lez         8'b00000100
`define select_qc_gtz         8'b00001000
`define select_qc_gez         8'b00010000
`define select_qc_ltz         8'b00100000
`define select_qc_jr          8'b01000000
`define select_qc_jalr        8'b10000000
`define select_qc_j                   8'b10000001
`define select_qc_jal         8'b10000010


// Select data to be written back to register file
`define select_wb_alu         3'b110
`define select_wb_load        3'b101
`define select_wb_link        3'b011

`define select_pred_dpred 3'b000
`define select_pred_gpred 3'b001
`define select_pred_ppred 3'b010
`define select_pred_tpred 3'b011
`define select_pred_vpred 3'b100


/*----------------------------------------------------------------------------
Miscellaneous
----------------------------------------------------------------------------*/

// Various width don't care and invalid values
`define dc32                  32'bxxxxxxxx
`define invalid               32'bxxxxxxxx
`define undefined             32'bxxxxxxxx
`define dc8                   8'bxxxxxxxx
```

```
`define dc6                   6'bxxxxxx
`define dc5                   5'bxxxxx
`define dc3                   3'bxxx
`define dc2                   2'bxx
`define dc                    1'bx
`define null                  1'bx
`define true                  1'b1
`define false                 1'b0

// State encodings for multi-cycle MIPS
`define IF_STATE      3'h0
`define ID_STATE      3'h1
`define EX_STATE      3'h2
`define MEM_STATE     3'h3
`define WB_STATE      3'h4

// Dsize encoding for supporting sub-word writes
`define SIZE_BYTE     2'h0
`define SIZE_HALF     2'h1
`define SIZE_WORD     2'h2

*********************************************
cache.h
// Tag fields
`define D_BO_WIDTH    2


//-----------------------------------------------------------------------
//
// Data Cache
//
//-----------------------------------------------------------------------
// If you change the number of words in a $ line you must change
// *all* of the following defines, as well as the memory system
// in mips.v
`define D_WO_WIDTH    3
`define MAX_OFFSET    7

`define D_INDEX_WIDTH 9
`define D_TAG_WIDTH   18

`define D_BO          1:0
`define D_WO          4:2
`define D_INDEX       13:5
`define D_TAG         31:14

// State fields
`define D_VALID               0
`define D_DIRTY               1


//-----------------------------------------------------------------------
//
// Instruction Cache
//
//-----------------------------------------------------------------------
// If you change the number of words in a $ line you must change
// *all* of the following defines, as well as the memory system
// in mips.v
`define I_WO_WIDTH    3

`define I_INDEX_WIDTH 9
`define I_TAG_WIDTH   18

`define I_BO          1:0
`define I_WO          4:2
`define I_INDEX       13:5
`define I_TAG         31:14

// State fields
`define I_VALID               0
```