AN ABSTRACT OF THE THESIS OF

Wanpeng Zhang for the degree of Master of Science in Electrical and Computer Engineering presented on November 22, 1999.

Tiltle: Power Estimation of Superscalar Microprocessor using VHDL Model

Abstract approved: _____

Shih-Lien Lu

Power optimization becomes more and more important due to the design cost and reliability. Sometimes high power consumption means expensive package cost and low reliability. The first step in optimizing power consumption is determining where power is consumed within a processor. While system-level code tracing and bit transition calculation are not enough to estimate the power distribution, transistor-level HSPICE simulation to model a microprocessor is too complex and time-consuming.

In our research, a VHDL model with enhanced signal tracing function will be developed based on the existing VHDL behavior model. The power consumption of superscalar microprocessor in terms of switching activity and capacitance will be carefully studied. Two factors served as the basis for study: accessibility and importance for power calculations. A brief examination of the datapath suggests that the register file, the instruction cache and data cache are some of the major contributors to power usage. It was therefore decided to track the

input and output bit transitions to these modules. These transitions are counted along with the number of accesses to each of the modules.

In order to gather all of this data, the original VHDL model simulator has been enhanced. As instructions pass through the CPU, additional code is required to track and record the necessary information. For each individual instruction in the ISA, various information is recorded based on the elements in the processor that the instruction affects. For instance, if the simulator is about to execute a load instruction, the instruction uses the programmer counter, the instruction bus, data bus, the address bus, the ALU (adder) and the register file. The information being recorded for each of these elements must be updated to reflect the execution of that particular load instruction.

Also, the inside circuit of each module, i.e. register file, instruction cache and data cache and the 6-transistor memory cell layout considering the 0.25μm CMOS technology will be studied in order to extract the capacitance. We do not need very accurate, absolute power estimation, therefore, we will try to keep the model simple.

Power Estimation of Superscalar Microprocessor using VHDL Model

by

Wanpeng Zhang

A THESIS

Submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented November 22, 1999
Commencement June 2000

Master of Science thesis of Wanpeng Zhang presented on November 22, 1999

APPROVED:

Major Professor, representing Electrical and Computer Engineering

Head of Department of Electrical and Computer Engineering

Redacted for privacy

Dean of Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Redacted for privacy

Wanpeng Zhang, Author

ACKNOWLEDGMENTS

I would like to express my sincere appreciation and gratitude to Prof. Shih-Lien Lu for his help, guidance and patience during my studies at Oregon state university. I am grateful to Prof. Ben Lee, Prof. Len Forbes and Prof. Garbor Temes for their advice and help. I would like to thank Professor Charles Brunner for his taking time out of his busy schedule to be present at my thesis defense.

I thank the faculty and office staff at the Electrical and Computer Engineering Department, especially Ms. Ferne Simendinger who does a wonderful job of keeping everything running smoothly.

I am thankful to Jose Silva for his friendship and support.

Finally, I will always be grateful to my parents for their support during my graduate studies and for many other things too numerous to mention here.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

POWER ESTIMATION OF SUPERSCALAR MICROPROCESSOR USING VHDL MODEL

Chapter 1
Introduction

1. Background

Previously, speed and area were the main forces driving the integrated circuits design. This is particularly true in the area of computer architecture and processor design, where area limitations primarily determine the allowable cache size. The design focus has traditionally been on boosting performance with higher frequency operation and additional circuitry for expanded functionality or throughput. Although temporarily, this increase in power dissipation can be resolved easily with simple package techniques and heat sinks. But the hardware complexity and consumer demands may make power a limiting factor in determining performance. There is a clear financial advantage to reducing the power considering the cost accompanied with the expensive packaging. In addition to cost, another important issue is reliability. High power system tends to run hot, and high temperature tends to exacerbate several silicon failure mechanisms. Every 10o C increase in operating temperature roughly doubles a component's failure rate. Thus, power consumption is yet another criterion for system design.

Current microprocessor design has a tendency towards wider issue and increasingly complex out-of-order execution. This leads to growth of the on-chip hardware, size of the register file, I-cache (Instruction Cache), D-cache (Data Cache) and consequently, an increase in dissipated power. These portions of a

micro-architecture have been described and analyzed [1] where complexity grows with increasing instruction-level parallelism. Among them are: register renaming logic, wakeup logic, selection logic, data bypass logic, register files, caches and instruction fetch logic.

Microprocessor designs must optimize trade-offs to meet the dual goals of high performance and low power system operation. The power consumption of some processors is shown in Table1.1 (from http://bwrc.eecs.berkeley.edu/CIC/).

	Pentium (75MHz)	P6 (150MHz)	PII (300MHz)	PIII (450MHz)
Power	16.00W	23.2W	43.0W	
	MIPS R4200 (80MHz)	MIPS R10000 (195MHz)	PowerPC604	PowerPC620
Power	1.5W	~30W*	14.00W	30.0W*
	Sparc Micro2 (110MHz)	Sparc Super 2 (90MHz)	UltraSparc (200MHz)	
Power	9W*	16W*	30.00W*	

* Peak power. The typical value is not available.

Table1.1. Power of Some Microprocessors

There have been a lot of power optimization techniques [2][3] at all level of the design hierarchy, i.e. system, algorithm (behavior), architecture, logic/circuits, devices and technology levels. The presented techniques and approaches ultimately all come down to a fundamental set of concepts: Lowering the supply voltage, the voltage swing, the physical capacitance, the switching activity or a combination of the above.

The first step in optimizing power consumption is determining where power is consumed within a processor. In the next section, the basic concepts and method will be reviewed.

2. Motivation and Methodology

In the early design stage, a precise estimation of the power dissipation is not always necessary. A fast architecture-level estimation is more useful for making early design decisions. As the design work progresses, a great deal of accuracy in the power estimation is increasingly required. Because of the long turn around time for gate-level simulation, a combination of both will be more efficient.

Dynamic power estimation tools simulate the design with a set of input vectors and estimate the power consumed. Since output switching transitions are quantified determininistically, dynamic power estimation is accurate. However, it requires prohibitively long simulation time for larger designs and larger input vector sets, especially for microprocessor. Although there are some methods to reduce the time complexity, such as generating a compact, representative vector set with similar switching behavior as the original larger vector set based on the fractal concept [6]. But further study reveals it is almost impossible for a microprocessor due to the large volume of address/data, also it is also unnecessary to generate a huge vector to estimate the power considering the design style of the microprocessor. For example, each time for a bit line write, there will be a bit transition on the bit line no matter what the data is written into the cache due to the bit line precharging and equalizing. Thus we can trace the write signal instead of the whole data to be written into the cache, which will save us a lot of time.

In our research, we attempt to deal with the simulation problem by analyzing the effect of actual program execution on bit transition activity. In order to do this, two things are required: the processor model to get the activity factor and the estimation of the capacitance.

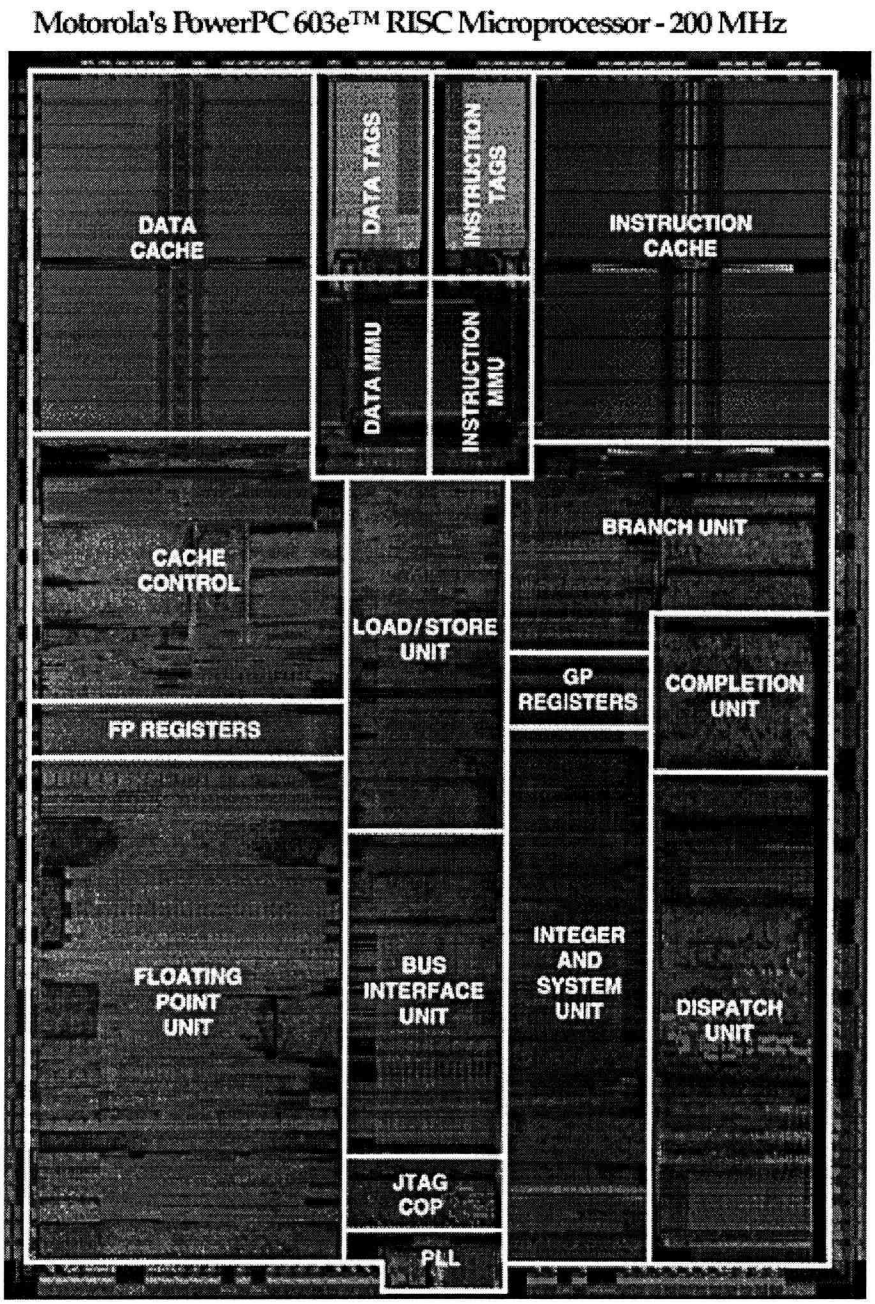

Figure1.1 Layout of the PowerPC603 Implementation

A VHDL (Very High Speed IC Hardware Description Language) model with enhanced signal tracing function has been developed based on the existing VHDL behavior model. The layout of the PowerPC603 is shown in Figure1.1.

From the layout, we can see that I-cache and D-cache (including cache control circuits) not only possess about 1/3 of the chip area, but also consume about 1/3 of power of the whole chip [7]. Based on the previous research [7][8], the register file, the instruction cache and data cache are some of the major contributors to power usage. It is therefore decided to track the input and output bit transitions to these modules. These transitions are counted along with the number of accesses to each of the modules. Also, the inside circuit structure of each module and even the layout will be studied in order to get the capacitance.

In summary, power estimation is very important in the early stage of microprocessor design. In the next chapter, a brief review of the literature will be presented. In chapter 3, the power consumption of the RF, I-cache and D-cache with respect to the data activity and capacitance will be carefully studied. In chapter 4 and 5, the simulation result and future work will be given out.

Chapter 2
REVIEW OF LITERATURE

2.1 Superscalar Microprocessor [9] [10]

Pipelining is a key implementation technique to increase IPC (Instruction Per Cycle) whereby multiple instructions are overlapped in execution. The basic five stages are shown in Figure2.1. Superscalar processors are uniprocessor organization capable of increasing machine performance by executing scalar instructions in parallel. The term superscalar emphasizes multiple, concurrent operations on scalar quantities. It is conceptually simple, but there is much more to achieving performance than widening a processor's pipeline. Three fundamental limitations, i.e. true data dependencies, procedural dependencies and resource conflicts, can limit the performance of a superscalar processor.

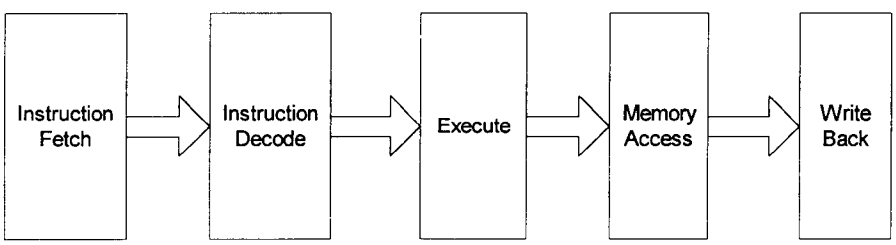

Figure 2.1 Five-Stage Pipeline Sketch

2.2 Sources of Dissipation in Digital Integrated Circuits [11]

To set the scene for the rest of this research, it is judicious at this point to briefly discuss the mechanisms for power consumption in CMOS circuits. Considering the CMOS inverter, the power consumed when this inverter is in use can be decomposed into two basic classes: static and dynamic. Each of these components will now be analyzed individually.

2.2.1 Static Power

Ideally, CMOS circuits dissipate no static power since in the steady state there is no direct path from V_{dd} to ground. Of course, this scenario can never be realized in practice since the MOS transistor is not a perfect switch. Thus there will always be leakage currents and substrate injection currents, which will give rise to the static component of CMOS power dissipation. For a submicron NMOS device with an effective W/L = 10/0.5, the substrate injection current is on the order of 1-100μA for V_{dd}=5V [12]. Since the substrate current reaches its maximum for gate voltage near $0.4V_{dd}$ and since gate voltage are only transiently in this range as device switch, the actual power contribution of the substrate injection current is several orders of magnitude below other contributors. Likewise, reverse-bias junction leakage currents associated with parasitic diodes in the CMOS device structure are on the order of nanoamps and will have little effect on overall power consumption.

Another form of static power dissipation occurs for so-called ratioed logic [13]. The static power consumed by these logic families can be considerable because for a pseudo-NMOS inverter, see Figure2.2, the PMOS pull-up is always on and there is a direct path from V_{dd} to ground and static current flow. For this

reason, logic families such as this with static power consumption should be avoided for low-power design. With that in mind, the static component of power consumption in low-power CMOS circuits should be negligible, and the focus shifts primarily to dynamic power consumption.

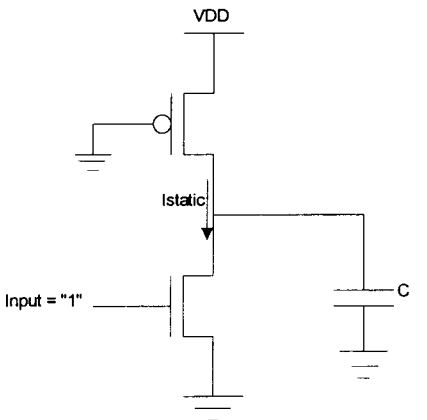

Figure2.2 Static Power Dissipation in Pseudo-NMOS Inverter

2.2.2 Dynamic Power

The dynamic component of power dissipation arises from the transient switching behavior of the CMOS device. At some point during the switching transient, both the NMOS and PMOS devices in Figure2.3 are turned on and there will be a current flow from V_{dd} to ground. A detailed analysis of this phenomenon by Veendrick reveals that with careful design of the transition edges this component can be kept below 10~15% of the total power [14].

Instead, dynamic dissipation due to capacitance charging consumes most of the power used by CMOS circuits. This component of dynamic power dissipation is

the result of charging and discharging parasitic capacitance in the circuit. Considering the behavior of the circuit over one full cycle of operation with the input voltage going from V_{dd} to ground and back to V_{dd} again. As the input switches from high to low, the NMOS pull-down network is cut off and PMOS pull-up is activated charging load capacitance C up to V_{dd}. This charging process draws an energy equal to CV_{dd}^2 from the power supply. Half of this is dissipated immediately in PMOS transistors, while the other half is stored on the load capacitance. Then, when the input returns to V_{dd}, the process is reversed and the capacitance is discharged, its energy being dissipated in the NMOS network. In summary, every time a capacitive node switches from ground to V_{dd} (and back to ground), an energy of CV_{dd}^2 is consumed.

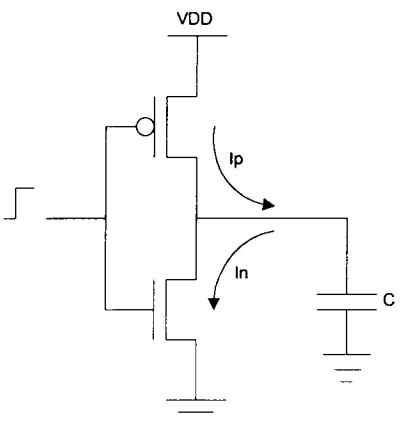

Figure2.3 CMOS Inverter for Power Analysis

This leads to the conclusion that CMOS power consumption depends on the switching activity of the signals involved. In this context, we can define activity, α, as the expected number of zero to one transitions per data cycle and average data-rate, *f*, as the clock frequency of the system, then we can get the following formulation for average CMOS power consumption,

$$P_{dyn} = \alpha C V_{dd}^2 f$$

In CMOS circuits, this component of power dissipation is by far the most important, typically accounting for at least 90% of the total power dissipation [14].

With its quadratic relationship to power, voltage reduction offers the most direct and dramatic means of minimizing energy consumption. There are, however, limiting factor such as minimum performance considering circuit delay and noise and compatibility requirements that limit voltage scaling. Unfortunately, these factors will likely lead designers to fix the voltage within a system.

Minimizing capacitance offers another technique for minimizing power consumption. This physical capacitance stems from two primary sources: devices and interconnect. For past technologies, device capacitance dominated over interconnect parasitic capacitance. As technologies continue to scale down, however, this no longer holds true and we must consider the contribution of interconnect to the overall physical capacitance [13].

In addition to voltage and physical capacitance, switching activity also influences dynamic power consumption. There are two components to switching activity. The first is the data rate f, which reflects how often on average new data arrives at each node. This data might or might not be different from the previous data value. In synchronous system f might correspond to the clock frequency. The second is the data activity α, which corresponds to the expected number of energy consuming transitions that will be triggered by the arrival of each new piece of data.

2.3 Power Estimation Techniques

Circuit simulation based techniques [15][16] simulate the circuit with a representative set of input vectors. They are accurate and capable of handling various device models, different circuit design styles, single and multi-phase

clocking methodologies, tri-state devices, etc. However, they suffer from memory and execution time constraints and are not suitable for large, cell-based designs. In general, it is difficult to generate a compact stimulus vector set to calculate accurate activity factors at the circuit nodes. The size of such a vector set is dependent on the application and the system environment [17].

A Monte Carlo simulation approach for power estimation which alleviates this problem has been proposed in [18]. This approach consists of applying randomly generated input patterns at the circuit inputs and monitoring the power dissipation per time interval T using a simulator. Based on the assumption that the power consumed by the circuit over any period T has a normal distribution, and for a desired percentage error in the power estimate and a given confidence level, the number of required power samples is estimated. The designer can use an existing simulator (circuit-level, gate-level or behavioral) in the inner loop of the Monte-Carlo program, thus trading accuracy for higher efficiency. The convergence time for this approach is fast when estimating the total power consumption of the circuit. However, when signal probability (or power consumption) values on individual lines of the circuit are required, the convergence rate is very slow. The method does not handle spatial correlations at the circuit inputs. The normality assumption does not hold in many circuits (for example, small circuits or circuits with enable lines that control activity in a significant part of the logic), and thus the approach may converge to wrong power estimate as a result of underestimating the number of required power samples.

Switch-level simulation techniques are in general much faster circuit-level simulation techniques, but are not as accurate or versatile. Standard switch-level simulators (such as IRSIM [19]) can be easily modified to report the switched capacitance (and thus dynamic power dissipation) during a simulation run.

PowerMill [20] is a transistor-level power simulator and analyzer which applies an event-driven timing simulation algorithm (based on simplified able-driven device models, circuit partitioning and single-step nonlinear iteration) to increase the speed by two or three orders of magnitude over SPICE. PowerMill

gives detailed power information (instantaneous, average and arms current values) as well as the total power consumption (due to steady-state transitions, hazards and glitches, transient short circuit currents and leakage currents). It also tracks the current density and voltage drop in the power net and identifies reliability problems caused by EM failures, ground bounce and excessive voltage drops.

Entice-Aspen [21] is a power analysis system which raises the level of abstraction for power estimation from the transistor level to the gate level. Aspen computes the circuit activity information using the Entice power characterization data as follows. A stimulus file is to be supplied to Entice where power and timing delay vectors are specified. The set of power vectors observes all possible events in which power can be dissipated by the cell. With the relevant parameters set according to the user's specs, a SPICE circuit simulation is invoked to accurately obtain the power dissipation of each cell and computes the total power consumption as the sum of the power dissipation for all cells in the power vector path.

Chapter 3
Power Estimation

In this chapter, we will carefully study the methodology to make the signal statistics and capacitance estimation on register file, instruction cache and data cache. From the following equation, we can see that the estimation of power consumption requires both the physical aspects of the design (i.e. technology parameters, circuit structure, delay model) as well as the signal statistics (i.e. data activity and correlations).

$$P_{dyn} = \alpha C V_{dd}^2 f$$

3.1 Activity

Unfortunately, while the equation for power calculation is simple enough, estimating α is often an extremely difficult problem because it depends not only on the switching activities of the circuit inputs and the logic function computed by the circuit, but also on the spatial and temporal correlations among the circuit inputs. Most circuit designers use a first-order, "white-noise" approximation that assumes the gate will switch on average every other clock cycle. Thus, α is usually taken as 25%, that is, the switching activity is 25% of the clock frequency. While this may be a reliable method for certain types of logic blocks, it can be wildly inaccurate when used to estimate power consumption in a microprocessor. When bit lines are correlated, the switching frequency of those lines can be much lower than the 25% estimate.

The problem of accurately assessing relative power usage ultimately requires the use of data from actual dynamic program execution. Such data is necessary to track the amount of bit activity in the various modules of a microprocessor. Actual programs may exhibit behavior that radically deviates from the simple "white-noise" assumptions. A good example of this would be the program counter. Not only are the bit transitions radically different from the "white-noise" assumption on every bit, but a more advanced estimate based on the sequential nature of the program counter depends on the percentage of instructions that are branches. If the program never branched, then bit two of the program counter would flip every instruction, or 100% of the time, bit three 50% of the time, bit four 25%, and so on. The presence of branches changes the situation. Yet the degree of change depends upon the actual branch frequency.

In this paper, we attempt to deal with this estimation problem by analyzing the effect of actual program execution on bit transition activity. In order to do this, two things are required: the estimation of capacitance values and the traces of actual program execution to estimate the frequency these capacitance are switched.

While many microprocessor manufacturers actually simulate their processors at the switch level on actual programs to verify their correctness, this is extremely computationally expensive, and usually not an option. For our research, such a simulation would be nearly impossible due to time and resource constraints and it would probably not be that much more effective than approach actually taken. Two factors served as the basis for selection: accessibility and importance for power calculations. It turns out that the signals that are important for power estimation are also easy to capture during simulation. Rather than simulate the entire processor, we will study some of the principle logic modules, such as the register file, the instruction cache and data cache.

3.1.1 Simulation Tool

In order to perform the subsequent power analysis on the superscalar microprocessor architecture, we must choose a simulation tool to monitor the execution of the program.

Trace-driven simulator uses a predetermined instruction sequence, an instruction trace, to evaluate superscalar performance. Trace-driven simulation is efficient, because the simulator is concerned only with the processor features that affect performance. But the simulator doesn't execute instructions or model the detailed operation of the functional units. To determine performance, the simulator simply models the functional units as elements which delay the operand values needed by some instructions and which also prevent the simultaneous execution of some instructions. It is not suitable in power estimation because the input/output of some functional units must be recorded to calculate the switching activity.

The advantage of execution-driven processor simulators, such as SimpleScalar, is high flexibility, portability, extensibility and performance. Simplescalar (release 2.0) can do a extremely fast functional simulation or a detailed simulation based on out-of-order issue superscalar processor that supports non-blocking caches and speculative execution. But it simulates a processor more like a virtual machine without taking the inside structure of each module into consideration.

Therefore, a VHDL model with almost identical configuration as PowerPC603 from the TU Darmstadt, Germany is introduced here (http://www.rs.e-technik.tu-darmstadt.de/TUD/res/dlxdocu/SuperscalarDLX.html), shown in Figure3.1. The PowerPC603 is chosen because it is a very typical suprscalar architecture with few extraneous quirks. The highlights of some modules such as register file, instruction cache and data cache are discussed. The main deviation between our research and the actual PowerPC603 is the absence of floating point unit in our model.

Below is the feature of this model,

- Two instructions can be fetched per clock cycle, in big-endian format
- The instruction set is a superset of the DLX RISC instruction set.
- Four execution units branch resolve unit, arithmetic logic unit, multiply divide unit and load store unit with reservation station
- Write-Buffer with write-through
- 2KByte Instruction Cache and 2KByte Data Cache (Direct mapped , block size 64 bits)
- 4 entry Instruction-Address-Translation-Buffer, page size: 128 byte
- 4 entry Data-Address-Translation-Buffer, page size: 128 byte

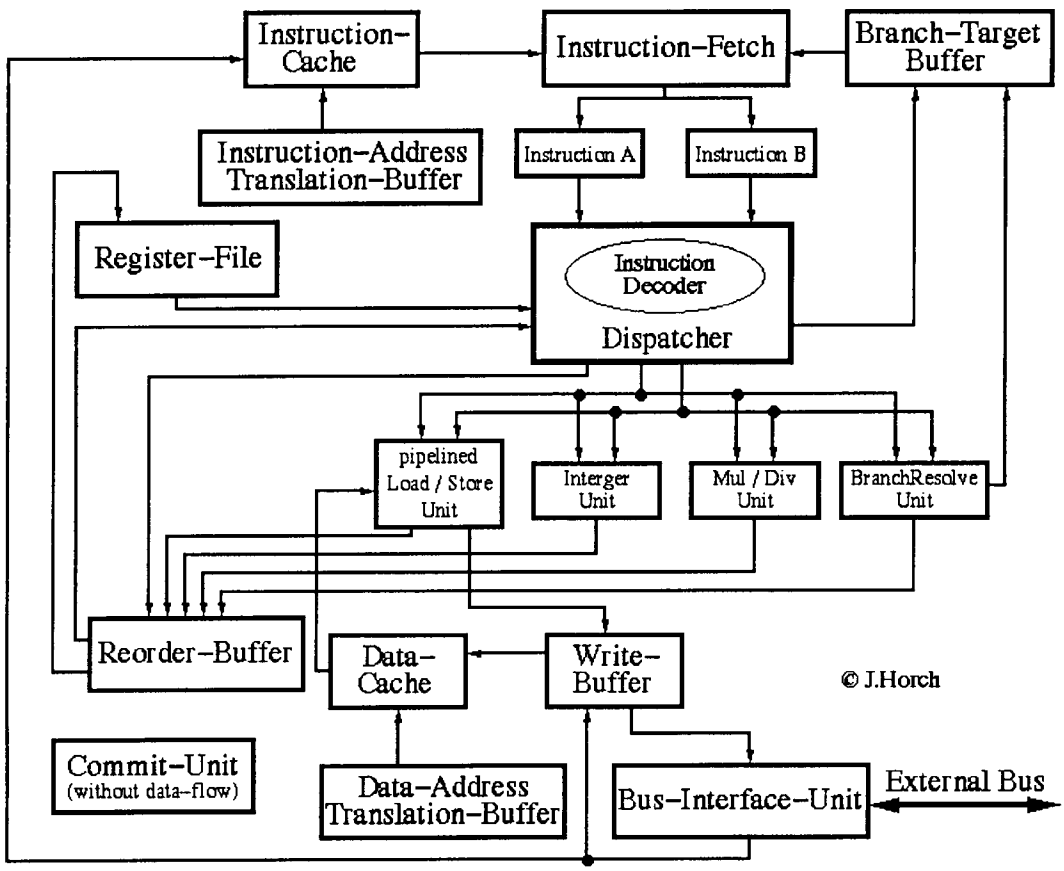

Figure3.1 PowerPC603 Superscalar Structure

Integer Datapath. The three main units, from a power perspective, are the register file, ALU and shifter. The register file contains two read and one write

port. It is designed so that all writes occur on the falling edge of the clock, such that register writeback occurs in the first half of the clock cycle, and register reads can occur on the second half. The register file brings in a 5-bit register specification for each port, and the decoding to 32 select lines is performed in the control slice appended to the bottom of the LSB bit slice. The ALU implements all the add, sub and logic instructions, as well as all the set-less-than instructions. The module is comprised of an adder, a comparator, and a custom front-end cell containing the logic blocks and MUX. The shifter module is a combination of a logarithmic up shifter and a down shifter. This module implements all the shift instructions and the load-upper-immediate instruction, which is essentially a 16-bit, hard-coded left shift.

Instruction Cache and Data Cache. The write-through policy was selected to ensure cache coherency with minimal extra hardware. With no virtual memory, all the indexing occurs with physical tags. The 64-byte line size was selected primarily for the instruction cache, which tends to make mostly sequential accesses. In a future implementation, the line-size for the data cache most likely would be reduced to minimize unnecessary loads into the data cache.

The 2KByte size was selected due to global area constraints. If the technology is upgraded, the caches can be easily increased in size, with minimal hardware changes. Also, the power increase does not scale linearly with cache size (a bulk of the power goes to fixed overhead), and is roughly constant, to first order. The extra memory doubles the number of blocks (256 64-bit words) in the memory. Since only one block is activated at a time, the larger memory consumes the same amount of power per access. So, if the technology allows it, the memory can be increased without increasing the average power consumed per cycle.

3.1.2 Register File

The processor's 32 registers are stored in a structure called a *register file*. A register file is a collection of registers in which any register can be read or written by specifying the number of the register in the file. The register file contains the register state of the machine. The interface of the register file is shown in Figure3.2. During the instruction decode stage, for each data word to be read from the registers, we need an input to the register file that specifies the register number to be read (i.e. IF_InstrRegA and IF_InstrRegB) and an output from the register file to carry the register value (i.e. RF_DataA1, A2, B1, B2). During the write back stage, when the write control signal (i.e. CU_RegisterWriteEnableA, B) is asserted, we need two inputs: one to specify the register number to be written (i.e. RB_RegisterDestinationCommitA, B) and one to supply the data to be written into the register (i.e. RB_DataCommitA, B).

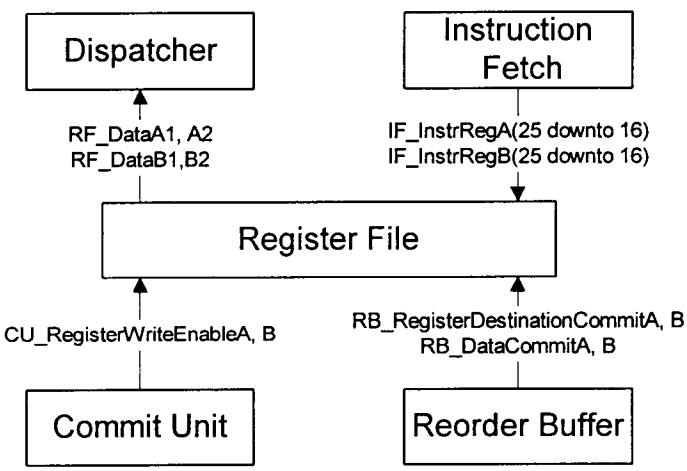

Figure 3.2 Interface of the Register File

Since writes to the register file are edge-triggered, we can legally read and write the same register within a clock cycle: the read will get the value written in an earlier clock cycle, while the value written will be available to a read in the subsequent clock cycle. The inputs carrying the register number are all 5-bit wide and data are 32-bit wide. In our design, there are two read ports and one write port for the register file. Two instructions can be fetched at the same clock cycle. Tracing the inputs and outputs of the register file, we can get the information about the activity factor of the register file.

3.1.3 Instruction Cache

Instruction cache is difficult to characterize due to the varied modes of operation it can be in, from simply fetching a word for the controller, to simultaneously loading in lines and fetching words. For the scope of this work, the caches are analyzed within the realm of normal operation, which is a cache hit. Previous work has shown a 2KB cache yields under 15% miss rates, which puts an upper bound on the error of this assumption [henn90]. Here the cache miss is ignored for power estimation. That is, we don't need to take care of the interface between instruction cache and memory.

The interface of the instruction cache is shown in Figure3.3. The two main routes in this module with high switching activity are the 32-bit wide program counter (PC) bus (i.e. IF_InstrCounterReg), whose data is either from the sequentially following instruction address (PC+4) or the branch target address and the instruction bus (i.e. IC_InstrA, B). Two instructions can be fetched simultaneously.

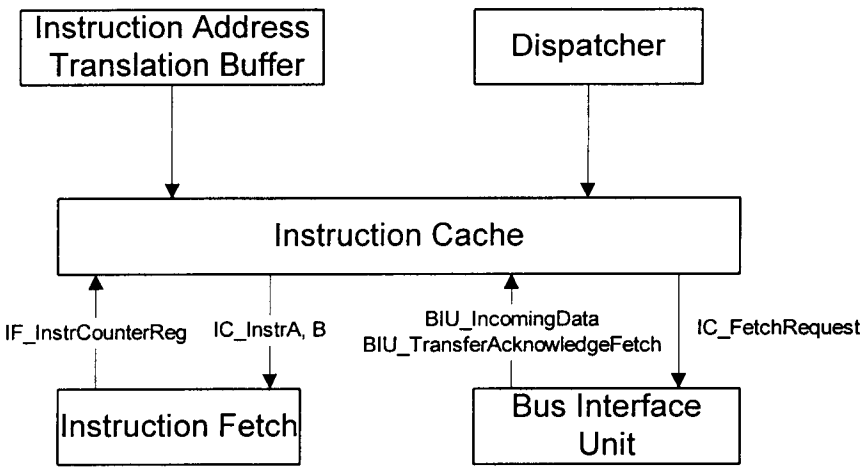

Figure 3.3 Interface of the Instruction Cache

3.1.4 Data Cache

Data cache is similar to the instruction cache except that there are two operations, read and write for a data cache. Due to the low miss rate when the data cache is reasonably big enough, here we ignore the data interface between the main memory and data cache. The interface of the instruction cache is shown in Figure3.4. For a cache read, when the cache read is asserted, the load/store unit (LSU) will generate an effective address (i.e. ISU_EA_AddrReg) to load the data (i.e. DC_DataOut). During the write back stage, while the cache write control signal (i.e. CU_CommitStore) is asserted, the ALU will calculate the effective address (i.e. WB_EntranceaddrReg) to store the data (i.e. WB_DataOut).

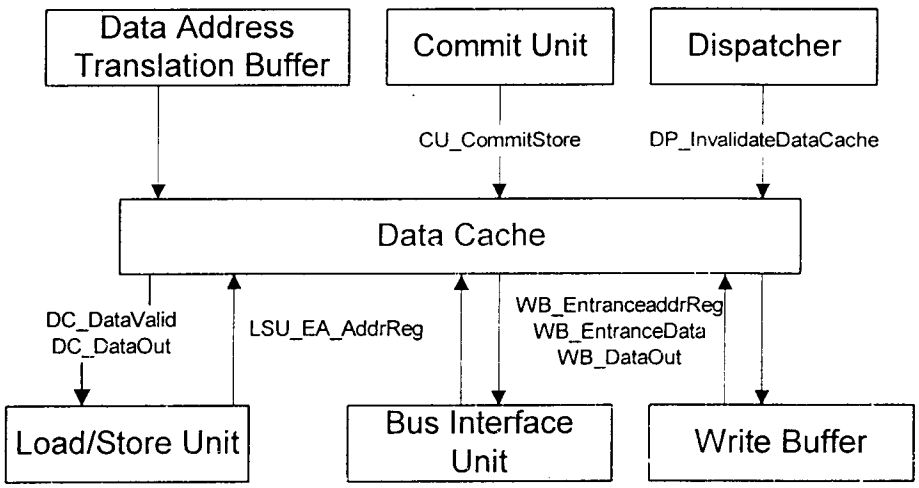

Figure 3.4 Interface of the D-cache

3.2 Capacitance

3.2.1 Basic Capacitance [23]

Gate Capacitance. The gate capacitance of a transistor consists two parts: the capacitance of the gate, and the capacitance of the polysilicon line going into the gate. Generally, the capacitance of the polysilicon line is much smaller than the gate capacitance.

$$C_{gate} = W \times L_{eff} \times C'_{gate} + L_{poly} \times L_{eff} \times C'_{polywire}$$

where W is the width of the transistor, L_{eff} is the effective length of the transistor, C'_{gate} and $C_{polywir}$ are the unit area capacitance of the gate and polysilicon line respectively.

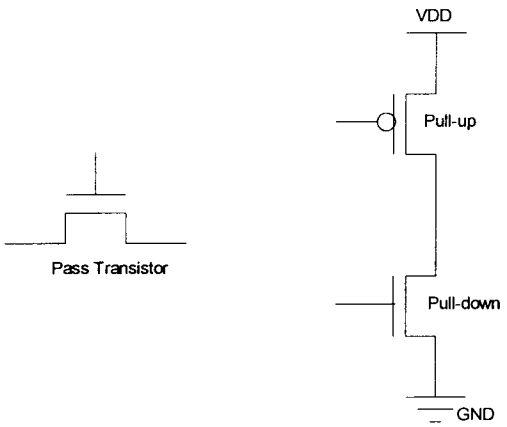

Figure3.5 Pass Transistor and Pull-up/down Transistor

The value of C_{gate} depends on whether the transistor is being used as a pass transistor, or as a pull-up or pull-down transistor in a static gate. As shown in Figure3.5, for a pull-up or pull-down transistor, the drain/source is connected to V_{dd}/GND. But for pass transistor, the drain and the source are supposed to have the same potential. The capacitance will be different due to the voltage difference. Thus, two equations for the gate capacitance are required,

$$C_{gate}(W, L_{poly}) = W \times L_{eff} \times C'_{gate} + L_{poly} \times L_{eff} \times C_{polywire}$$

$$C_{gate}(W, L_{poly})_{pass} = W \times L_{eff} \times C'_{gate,pass} + L_{poly} \times L_{eff} \times C_{polywire}$$

C'_{gate}, $C'_{gate,pass}$, $C_{polywire}$ and L_{eff} are technology parameters shown in the next chapter. A different L_{poly} was used in the model for each transistor. This L_{poly} was chosen on typical poly wire lengths for the structure in which it is used.

Drain Capacitance. Figure3.6 shows the typical transistor layout. The drain capacitance is composed of both an area and perimeter component. Using the geometry in Figure3.7, the drain capacitance for a single transistor can be obtained.

$$C_{drain}(W) = 3 \times W \times L_{eff} \times C_{diffarea} + (6L_{eff} + W) \times C_{diffside} + W \times C_{diffgate}$$

where $C_{diffarea}$, $C_{diffside}$ and $C_{diffgate}$ are process dependent parameters (for PMOS and NMOS, the value is different).

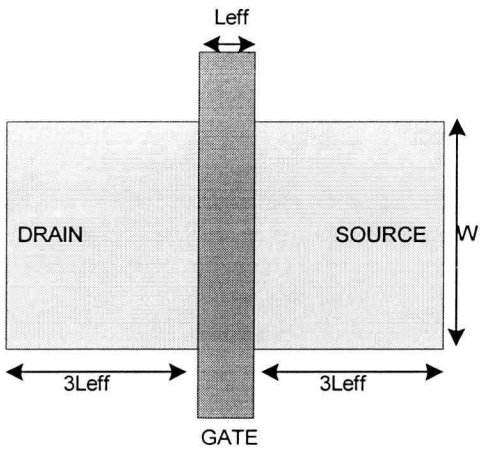

Figure3.6 Transistor geometry

Other Capacitance. Other parasitic capacitance such as metal wiring are modeled using the values for $C_{bitmetal}$ and $C_{wordmetal}$ given in chapter 4. These values are fixed values per unit length in terms of the RAM cell length and width. These values include an expected value for the area capacitance estimation of the bit lines and word lines themselves, they are also used to model the capacitance of the pre-decode lines, data bus, address and other signals in the memory. Although the capacitance per unit length would probably less for many of these buses than for the bit lines and word lines, the same value is used for simplicity of modeling.

3.2.2 Register File [24]

The overall RF access power (read or write) is the sum of power dissipated in the decode logic, memory array, sense amplifiers(in case of read access), power dissipated for driving signals that control the operation of the sense amplifiers, precharge circuitry and write drivers (see Figure3.7).

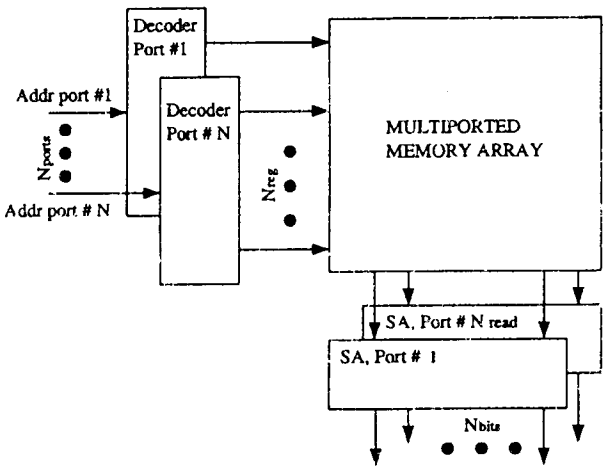

Figure3.7 Generic Register File with N_{read} Read Ports and N_{write} Write Ports

The power dissipated by the decoder logic is typically less than 10% of the total power. Moreover, it does not significantly depend on the register file organization. Therefore we will exclude this portion of the dissipation power from our estimation. The memory array portion, on the other hand, needs to be studied in detail, because it represents the major portion of the RF power dissipation, and because none of the existing energy/power models can be applied to these highly multiported memory configurations.

The conventional multiported memory cell for RF, shown in Figure3.7, typically uses two bit lines per write port and one bit line per read port, as well as one word line per every port to control the connection of the cell to the bit lines of the corresponding port [25]. Thus, there are $N_{read} + N_{write}$ word lines for every row

in the array, and $N_{read} + 2N_{write}$ bit lines. Multiple word lines can go high at the same time in case of simultaneous access through several ports to the same cell. Therefore the cell must be capable of driving significant current which is proportional to the number of read ports. To protect the data stored in the cell during such multiple read accesses, an additional buffer is typically inserted between the cell flip flop and the read pass transistor, further referred to as a decoupling buffer. Because of this decoupling buffer, the read bit line cannot serve as a write bit line, and thus the total of $N_{read} + 2N_{write}$ bit lines are needed.

3.2.2.1 Read access

Based on the previous research, the bit line energy is the dominant component to the power of the read cycle for large register files. For small register files, on the other hand, the sense amplifier energy dominates.

We will consider three terms in the read access power: the word line power, $P_{wl,read}$, bit line power, $P_{bl,read}$ and sense amplifier power P_{SA}.

Word Line Power. $C_{wl,read}$ is the read word line capacitance which is the sum of the line capacitance and the gate capacitance of pass transistors,

$$C_{wl,read} = C_{line} + C_{pass,r}$$

In single ended sensing schemes the word line is connected to one pass transistor in every cell, therefore, $C_{pass,r} = C_{gate,pass} N_{bits}$. For the line capacitance we have: $C_{line} = C_{wordmetal} N_{bits}$, where $C_{wordmetal}$ is the word metal line capacitance per bit. Thus, we have for the word line power consumption of one read access:

$$P_{wl,read} = C_{wl,read} V_{dd}^2 f$$

Bit Line Power. The bit line capacitance is the sum of the metal line capacitance and the diffusion capacitance of pass transistors connected to the bit line,

$$C_{bl,read} = C_{line} + C_{pass,r}$$

where $C_{pass,r} = C_{drain} W_{pass,r} N_{reg}$, C_{drain} is the drain capacitance of the pass transistor (including the capacitance of a contact), $W_{pass,r}$ is the width of the cell pass transistor and N_{reg} is the number of registers. $C_{line} = C_{bitmetal} N_{reg}$, where $C_{bitmetal}$ is the bit metal line capacitance per unit length.

We assume that bit line loads are entirely cut off during reading, so that the selected memory cell is the only current source that drives the bit line during reading phase. On the other hand, during the precharge phase, the precharge transistors are the only current source that drives bit lines. Under these assumptions, the power dissipated for driving bit lines in a read access is

$$P_{bl,read} = \alpha_{bitline} N_{bits} C_{bl,read} V_{dd} V_{bl} f$$

where empirically, $V_{bl} = 1.3 V_{sense} = 1.3 * (2 * 300mV + 0.1 V_{dd})$, V_{sense} is the voltage sufficient for reliable sensing and $\alpha_{bitline}$ is the bit line transition activity, i.e. the percentage of the zero on each read access because the bit line is precharged to high and will be discharged only if zero is read from the cell.

Sensing Circuitry Power. The energy dissipated by sensing circuitry greatly depends on the type of the sense amplifier (SA) used. The scheme that is normally used is basically an inverter with the input connected to the bit line. The power dissipated in one SA during access can be estimated as ,

$$P_{SA,inv} = \frac{1}{32} V_{dd} I_{dsat}$$

where $I_{dsat} = 0.5mA$, I_{dsat} is the drain saturation current per unit and it is independent of the feature size, which has indeed been observed in practice [26]

3.2.2.2 Write access

The write operation power in a register file has a higher importance than in memories because in RISC processors the ratio between the number of RF reads and writes is typically reported as about 2:1, compared to the typical 3:1 ratio between the number of loads and stores.

For the write access power we consider two terms: the word line power, $P_{wl,write}$ and the bit line power, $P_{bl,write}$.

Word Line Power. For the word line power, every word line is connected to the gates of two write pass transistors in every cell for differential writes. These write pass transistors do not need to be any bigger than the minimum transistor size, Thus,

$$C_{wl,write} = C_{line} + C_{pass,w}$$

$$P_{wl,write} = C_{wl,write} V_{dd}^2 f$$

where $C_{pass,w} = 2C_{gate,pass} N_{bits}$ and $C_{line} = C_{wordmetal} N_{bits}$, the same as read access.

Bit Line Power. Each time for a bit line write, there will be a bit transition on the bit line no matter what the data is written into the register due to the bit line precharging and equalizing. The write power can be reduced if after every write operation we equalize the write bit lines through an equalizing transistor, shown in Figure3.8. In this case, in the limit, we save half of the energy stored in the bit line that was at V_{dd} during the write operation, so that for each write operation,

$$P_{bl,write} = \frac{1}{2} N_{bits} C_{bl,write} V_{dd}^2 f$$

The write bit line capacitance is the same as the read bit line capacitance, except that the write pass transistors have the minimum size independent of the size of the register file,

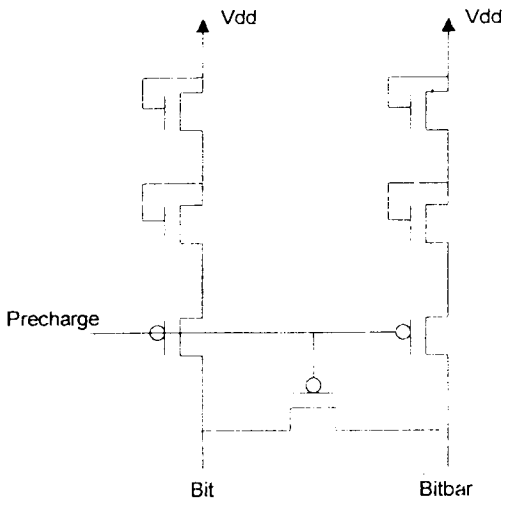

Figure 3.8 Precharging and Equilibration Transistors

3.2.3 I-cache and D-cache [23]

Before describing the model, the internal structure of an SRAM cache will be briefly reviewed. Figure3.9 shows the assumed organization. The decoder first decodes the address and selects the appropriate row by driving one word line in the data array and one word line in the tag array. Each array contains as many word lines as there are rows in the array, but only one word line in each array can go high at a time. Each memory cell along the selected row is associated with a pair of bit lines; each bit line is initially precharged high. When a word line goes high, each memory cell in that row pulls down one of its two bit lines; the value stored in the memory cell determines which bit line goes low.

Each sense amplifier monitors a pair of bit lines and detects when one changes. By detecting which line goes low, the sense amplifier can determine what is in the memory cell. It is possible for one sense amplifier to be shared among several pairs of bit lines. In this case, a multiplexer is inserted before the sense amps; the select lines of the multiplexor are driven by the decoder. The number of bit lines that share a sense amplifier depends on the layout parameters described in the next section.

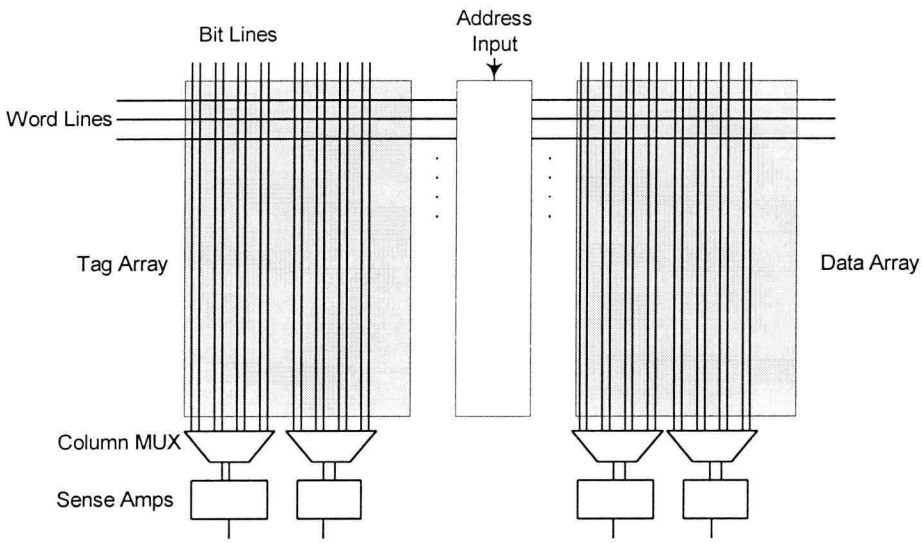

Figure 3.9 Cache Structure

The information read from the tag array is compared to tag bits of the address. In an A-way set-associative cache, A comparators are required. The results of the A comparisons are used to drive a valid (hit/miss) output as well as to drive the output multiplexors. These output multiplexors select the proper data from the data array, and drive the selected data out of the cache.

Generally, the power dissipated by decode logic is much less than 10%, it can be neglected. The bit line power is dominant component for cache. We will consider two terms in the I-cache: the word line power, $P_{wl,I-cache}$ and bit line power, $P_{bl,I-cache}$.

Word Line Power. The word line power consists of two parts, the data word line and the tag word line. Figure3.10 shows the word line driver driving a word line. $C_{wl,tag}$ and $C_{wl,data}$ are the tag word line and data word line capacitance respectively, which is the sum of the line capacitance, the drain capacitance and the gate capacitance of pass transistors,

$$C_{wl,tag} = 2B_{tag}C_{gata,pass}(W, W_{cell} - 2W) + C_{drainp}(W_{tagwordp}1) + C_{drainn}(W_{tagwordn}1) + B_{tag}C_{wordmental}$$

$$C_{wl,data} = 2B_{data}C_{gata,pass}(W, W_{cell} - 2W) + C_{drainp}(W_{datawordp}1) + C_{drainn}(W_{datawordn}1) + B_{data}C_{wordment}$$

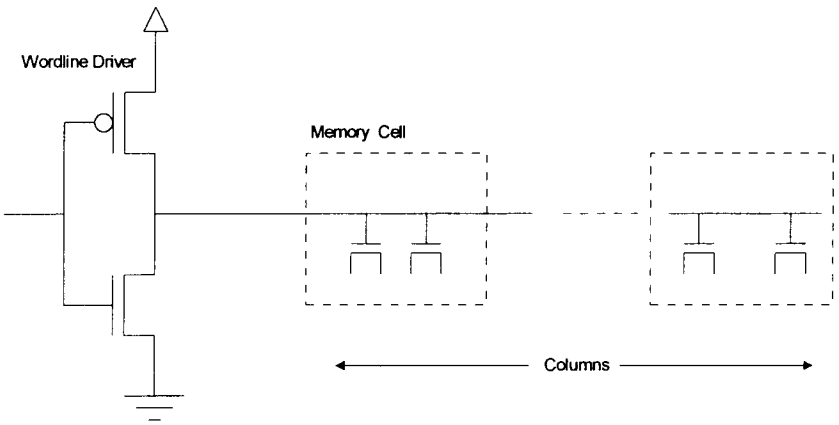

Figure3.10 Word Line Architecture

where $B_{data} = \frac{8BAN_{spd}}{N_{dwl}}$, B is the block size in bytes, A is associativity, N_{spd} indicates how many sets are mapped to a single word line and N_{dwl} indicates how many times the array has been split with vertical cut lines.

$B_{tag} = B_{addr} - \log_2 C + \log_2 A + 2$, B_{addr} is address width in bits, C is the cache size in bytes and +2 is because of the valid and dirty bits. $C_{wordmetal}$ is the word metal line capacitance per unit length, and W_{cell} is the cell width. Thus, we have for the word line power consumption of one access:

$$P_{wl,1-cache} = (C_{wl,tag} + C_{wl,data})V_{dd}^2 f$$

Bit Line Power. Each column in the array has two bit lines: bit and bitbar. A typical SRAM cell is shown in Figure3.11. After one of the word lines goes high, each memory cell in the selected row begins to pull down one of its two bit lines; which bit line drops depends on the value stored in that memory cell.

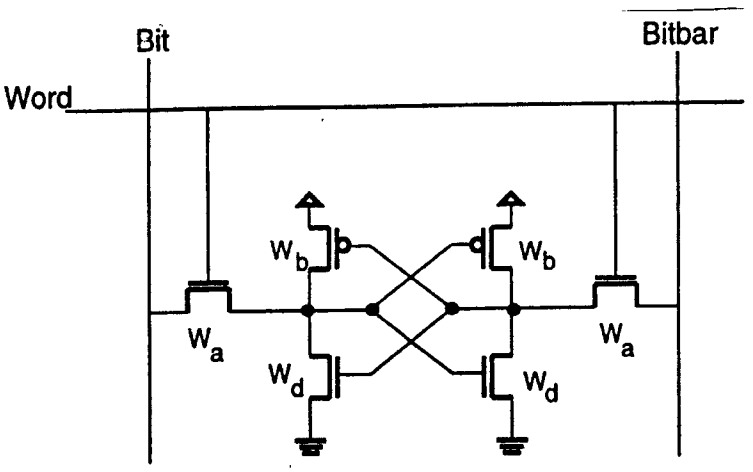

Figure3.11 One Memory Cell

The bit line capacitance has two parts, $C_{line,cache}$ and $C_{mux,cache}$. The capacitance $C_{line,cache}$ includes the capacitance of the memory cells sharing the bit line, the metal line capacitance, the drain capacitance of the precharge circuit, and the drain capacitance of the column multiplexor pass transistor. The capacitance $C_{mux,cache}$ is the capacitance seen by the output of the conducting column multiplexor pass transistor, shown in Figure3.12. It includes the drain capacitance of all pass transistors connected to this sense amplifier and the input capacitance of the sense amplifier:

$$C_{bl,cache} = C_{line,cache} + C_{mux,cache}$$

$$C_{line,cache} = B_{rows}(\frac{1}{2}C_{drain,n}(W,1) + C_{bitmetal}) + 2C_{drain,p}(W_{bitpreequ},1) + C_{drain,n}(W_{bitmuxn},1)$$

$$C_{mux,cache} = N_{spd}N_{dwl}C_{drain,n}(W_{bitmuxn},1) + 2C_{gate}(W_{senseQ\,1to4},10)$$

where $B_{rows} = \frac{C}{BAN_{dbl}N_{spd}}$, is the rows of the memory array. If $N_{dbl}N_{spd} = 1$, then there is no column multiplexors, the bit line capacitance will be

$$C_{bl,cache} = C_{line,cache} + C_{mux,cache}$$

$$C_{line,cache} = B_{rows}(\frac{1}{2}C_{drain,n}(W,1) + C_{bitmental}) + 2C_{drain,p}(W_{bitpreequ},1)$$

$$C_{mux,cache} = 2C_{gate}(W_{senseQ1to4},10)$$

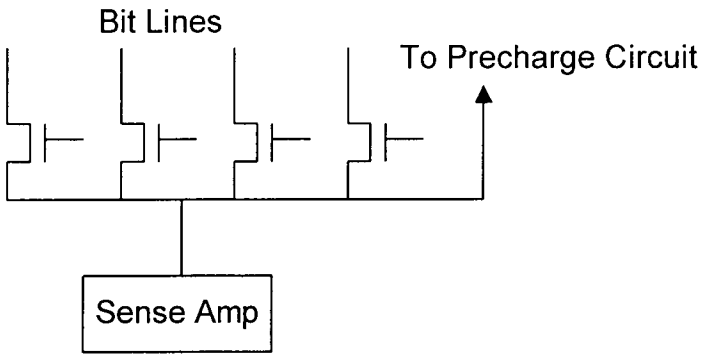

Figure3.12 Column Select Multiplexor

We assume that bit line loads are entirely cut off during reading, so that the selected memory cell is the only current source that drives the bit line during reading phase. On the other hand, during the precharge phase, the precharge transistors are the only current source that drives bit lines. Under these assumptions, the power dissipated for driving bit lines in a read access is

$$P_{bl,1-cache} = \alpha_{bitline}N_{bits}C_{bl,cache}V_{dd}V_{bl}f$$

Here the estimation method is the same as the register file.

The main difference between the caches is that the I-cache is only written during a line load, while the D-cache can be written during both a line load, and during a store instruction. This creates a different bus structure for the external data

bus (MDBUS) leading into the cache. This also creates slightly different cache controllers.

For a data read, we estimate the word line and bit line power consumption as I-cache. For a data write, the word line power is the same as I-cache, but the bit line power will be a little different because of the equalizing transistor. In this case, in the limit, we save half of the energy stored in the bit line that was at V_{dd} during the write operation, so that

$$P_{wl,D-cache} = (C_{wl,tag} + C_{wl,data})V_{dd}^2 f$$

$$P_{bl,D-cache,read} = \frac{1}{2}\alpha_{bitline} N_{bits} C_{bl,cache} V_{dd}^2 f$$

$$P_{bl,D-cache,write} = \frac{1}{2} N_{bits} C_{bl,cache} V_{dd}^2 f$$

The D-cache bit line capacitance is the same as the I-cache, except that the write pass transistors have the minimum size independent of the size of the register file.

Chapter 4
Simulation Results

The VHDL simulator is the basic engine for tracking various signals in the microprocessor. The 2.5V and 0.25μm CMOS technology is used here to estimate the capacitance. The clock frequency of the microprocessor is 100MHz.

Two factors served as the basis for selection: accessibility, and importance for power calculations. It turns out that the signals that are important for power estimation are also easy to capture during simulation. Since the busses themselves do not influence each other's values during this data flow, the relative frequencies of the outputs to these routing elements can be accurately assessed. Only when values of one bus affect values of another within a module, does correlation begin to change.

In order to gather all of this data, the original VHDL simulator has been modified. As instructions pass through the CPU, additional code is required to track and record the necessary information. For each individual instruction in the ISA, various information is recorded based on the elements in the processor that the instruction affects. For instance, if the simulator is about to execute a load instruction, that instruction uses the I-cache the D-cache and the register file. The information being recorded for each of these elements must be updated to reflect the bit transition of that particular load instruction.

4.1 Activity Factor

Due to the speed limitation of the simulator, SPEC95 benchmark isn't used here. The following benchmark will be used to monitor the signal flow, shown in Table4.1. VHDL simulator will automatically load the binary file generated by the assembler into the system memory.

Benchmark	Description	Execution Time (Cycles)
ALU	Basic ALU operations, add, sub…	72
Branch	Branch/Jump operations	104
Bubble	Bubble sort algorithm	9770
Fibinarci	Fibinarci array generation	161
LoadStore	Memory access operations	121
Prime	Prime number generation	5529

Table4.1 Test Benchmark

The address and data will be traced to obtain to activity factor of each block,

$$\alpha = \frac{N_{tran}}{2N_{bits}N_{cycle}}$$

where N_{tran} is the number of bit transition of the signal, N_{bits} is the signal width and N_{cycle} is the program execution time in clock cycle.

Instruction	Average Frequency
Load	26%
Store	9%
ALU(Integer)	46%
Branch	19%

Table4.2 Typical Instruction Mix [8]

Register File. The ratio between register reads and writes is typically reported as about 2:1, compared to the typical 3:1 ratio between the number of loads and stores of memory access, shown in Table4.2. The results are shown in Table4.3 and Table4.4. The 32-entry register file reflects that in microprocessor design, the smaller hardware is faster and more power efficient. The data activity is 0.18 for register file read in average and 0.31 for register write. For ALU and LoadStore, we can see more operations on the register file. The address activity isn't taken into consideration here. The reason will be discussed later. Because we look into two instructions at a time, $N_{bits} = 128$.

Benchmark	Read				Activity Factor (Data)
	times	Data (128 bits)	Address (32 bits)	Frequency	
ALU	34	3206	-	0.47	0.17
Branch	45	5452	-	0.43	0.21
Bubblesort	5242	421089	-	0.54	0.17
Fibinarci	62	7792	-	0.39	0.19
LoadStore	60	7090	-	0.50	0.23
Prime	2548	125628	-	0.46	0.09
Average				0.47	0.18

Table4.3 Activity Factor for Register File Read

Benchmark	Write			Activity Factor* (Data)
	times	RF address	RF data (128 bits)	
ALU	34	-	856	0.47
Branch	41	-	328	0.39
Bubblesort	3901	-	154934	0.39
Fibinarci	29	-	136	0.18
LoadStore	30	-	594	0.25
Prime	1035	-	18394	0.19
Average				0.31

* For a write, the activity factor from data is meaningless. Times of write is used instead.

α= times of write/execution time (in cycles)

Table4.4 Activity Factor for Register File Write

Instruction Cache. For I-cache, only the cache read operation is considered because we ignore the interface between I-cache and main memory while there is a cache miss. The results are shown in Table4.5. Due to the branch/jump frequency, over 80% of the instructions are executed sequentially. The average address activity is only as low as 0.01, so it can be ignored with little effect on the estimation accuracy. The data activity is 0.29 in average. There are at least two ways to improve the I-cache power efficiency considering address/data bus. One is that the offset can be sent instead of the whole effective address if the program is executed in sequence or within some range. The other is a power efficient instruction set architecture by improving the encoding.

Benchmark	Read				Activity Factor	
	times	address (32 bits)	Data (64 bits)	frequency	address	data
ALU	36	22	1249	0.5	0.005	0.14
Branch	64	41	3132	0.62	0.006	0.24
Bubblesort	7437	16071	778362	0.76	0.025	0.62
Fibinarci	109	88	5675	0.68	0.009	0.28
LoadStore	62	31	3283	0.51	0.004	0.21
Prime	3640	3765	175620	0.66	0.01	0.25
Average				0.62	0.01	0.29

Table4.5 Activity Factor for I-cache

Data Cache. For a cache write, the activity factor is only related to the times of write operation because there will be a whole cache line transitions for a write no matter what is written. As shown in Table4.6, the activity factor for address is small compared to data, thus they can be ignored. The activity for data read is 0.19 in average and it is 0.16 for data write.

4.2 Capacitance

Based on the TSMC0.25μm technology and the layout, the parameters used in the capacitance estimation are given in Table4.7 and Table4.8.

Benchmark	Read times	Address (32 bits)	Data (64 bits)	Write Times	Read Frequency	Activity Factor		
						address	read	Write
ALU	24	0	794	0	0.33	0	0.09	-
Branch	26	0	1302	0	0.25	0	0.10	-
Bubblesort	5221	9382	326437	3403	0.53	0.015	0.26	0.35
Fibinarci	105	126	6100	21	0.65	0.012	0.30	0.13
LoadStore	93	51	4424	19	0.77	0.007	0.29	0.16
Prime	1135	445	66054	45	0.21	0.001	0.09	0.008
Average					0.46	0.006	0.19	0.16

* For data write, times of write is used for activity factor instead.

α= times of write/execution time (in cycles)

Table4.6 Activity Factor for D-cache Read/Write

Now, the capacitance of the register file, I-cache and D-cache can be estimated based on the empirical equation. Results are shown in Table4.9. The bitline capacitance and wordline capacitance are the biggest contributors to the total cache capacitance. With the estimation equation, the bitline/wordline is still a major source of power consumption when the memory array is big. Based on the statistics, 71% of the total capacitance is found in the integer datapath and I-cache.

Parameter	Value
$C_{bitmental}$	35 aF/bit
$C_{wordmental}$	17 aF/bit
C'_{gate}	5800 aF/μm^2
$C'_{gatepass}$	4312 aF/μm^2
$C_{ndiffarea}$	1872 aF/μm^2
$C_{ndiffside}$	440 aF/μm
$C_{ndiffgate}$	627aF/μm
$C_{pdiffarea}$	1877 aF/μm^2
$C_{pdiffside}$	352 aF/μm
$C_{pdiffgate}$	559aF/μm
$C_{polywire}$	97 aF/μm^2
L_{eff}	0.25 μm
V_{dd}	2.5 V

Table4.7 0.25μm CMOS Process Parameter

Stage	Symbol	Value
Tag wordline driver	$W_{tagwordp}$	3μm
	$W_{tagwordn}$	1.5μm
Memory cell	W_a	0.25μm
	Bit width	2.5μm
	Bit Height	5μm
Bitlines	$W_{bitpreequ}$	25μm
	$W_{bitmuxn}$	3μm
Sense Amp.	$W_{senseQ1to4}$	1.3μm

Table4.8 CMOS Circuit Parameter

Module	Symbol	Value
Register File	$C_{wl,read}$	0.011pF
	$C_{bl,read}$	0.011pF
	$C_{wl,write}$	0.021pF
	$C_{bl,write}$	0.011pF
I-cache and D-cache	$C_{wl,tag}$	0.027pF
	$C_{wl,data}$	0.054pF
	$C_{bl,cache}$	0.063pF

Table4.9 Module Capacitance

4.3 Power Consumption

Compared to 3.3V 0.35μm CMOS technology, due to the lower voltage supply, shown in Table4.10, the power consumption will drop correspondingly. I-cache and D-cache still are main sources of the power consumption compared to Register File. Combining the logic level switching activity estimation and technology level capacitance calculation, we can estimate the power consumption of a microprocessor concerning the time complexity and accuracy.

Benchmark	Power (μW)				
	Register File		I-cache	D-cache	
	Read	Write	Read	Read	Write
ALU	58.8	58.2	181.2	130.1	-
Branch	62.4	48.3	298.7	138.7	-
Bubble	59.2	48.3	729.1	354.4	458.7
Fibinarci	60.2	22.3	346.3	410.9	170.4
LoadStore	64.8	30.9	259.7	404.4	209.7
Prime	50.9	23.5	311.9	124.0	10.5
Average	59.7	38.4	354.4	262.7	209.7

Table4.10 Power Consumption

Also if we combined all the terms and obtain the average access power of RF as a sum of read access power and the write access power according to the typical read/write ratio of 2:1. The same method can be used for D-cache. But the typical read/write ratio will be 3:1.

Further speaking, the same power estimation method can be applied to the RF with more read/write ports and larger volume I-cache and D-cache as long as the activity factor and capacitance can be extracted. For RF, we assume that the same design architecture is used except we need more transistors (one pass transistor for each read port and two for each write port) for multiple read/write access. Detailed description can be found in [24]. We see that the power doesn't increase a lot because the register file is still the same size and there is just some overhead due to the multiple read/write access. For I-cache and D-cache, we assume that the cache is direct mapped and the block size is 64bits. For cache, the word line capacitance for tag array and data array has no change, but the bit line capacitance will increase drastically. There will be considerable power increase due to the cache size.

In summary, power estimation is very important in low power microprocessor design. This model can be easily modified to extract the activity factor of any logic modules. At the same time, by studying the correspondent circuits and the selected technology, we can obtain the capacitance. The power consumption can be estimated combining all these factors together.

Chapter 5
Future Work

Taking the power consumption down another factor of ten while maintaining the throughput is the next challenge in low power microprocessor design. While some of this can be achieved with a technology shrink of the processor, a significant part will have to be achieved with architecture changes.

With VHDL implementation and power analysis of the register file, I-cache and D-cache, the same method can be applied to ALU and even the whole microprocessor. In our research, The future direction might be answering what can be applied on a more universal scale and what can be learned from power estimation in the development of a low power microprocessor.

- Applications to other function modules, like Instruction Window, ALU, Datapath Control Logic etc.
- Applications beyond the PowerPC603 architecture. The same method can be applied to other superscalar RISC processors with remodeling the microprocessor with VHDL and study the inside structure.
- Including cache-memory interface power in the analysis. In this study, we didn't look into the interface between I-cache/D-cache and off-chip main memory when there is a cache miss. The power consumption for the interface is no longer neglectable due to the relatively big capacitance.

BIBLIOGRAPHY

[1] S. Palachara, N. P. Jouppi and J. E. Smith, "Complexity-Effective Superscalar Processors", *Proceedings of the 24th Annual International Symposium on Computer Architecture*. pp206-218, June 1997

[2] K. Itoh, K. Sasaki and Y. Nakagome. Trends in Low Power RAM Circuit Technologies. *Proceedings of the IEEE*, pp524-543, 1995

[3] K. Kimura, "Power Reduction Techniques in Megabit DRAM's". *IEEE Journal of Solid State Circuits*. Vol SC-21, pp381-389, June 1986.

[4] A. Karandikar, "Low Power SRAM Design using Hierarchical Divided Bit-Line Approach". *IEEE Inter. Conference on Computer design* , pp82-88, October, 1998.

[5] T. Lang, E. Muoll and J. Cortadella. "Extension of the Working-Zone-Encoding Method to Reduce the Energy on the Microprocessor Data Bus". *IEEE Inter. Conference on Computer Design. pp414-419*, October, 1998.

[6] R. Radjassamy and J. D. Carothers. "A fractal Compaction Algorithm for Efficient Power Estimation". *IEEE Inter. Conference on Computer Design*, pp542-547, October, 1998

[7] Trevor Pering, Tom Burd and Robert Brodersen. "Dynamic Voltage Scaling and the Design of a Low-Power Microprocessor System". *The 25th Annual International Symposium on Computer Architecture.* pp375-380, June, 1998.

[8] Tom Burd. "Power Analysis of a Microprocessor: A Study of an Implementation of the MIPS R3000 Architecture". *ERL Technical Report,* University of California, Berkeley, 1994

[9] J. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Second edition. Morgan Kaufman Publishers, 1996.

[10] Mike Johnson. Superscalar Microprocessor Design. 1991.

[11] Jan M. Rabaey. *Low Power Design Methodologies*. 1996.

[12] R. K. Watts. *Submicron Integrated Circuits*, John Wiley & Sons, New York, 1989

[13] J. Rabaey, *Digital Integrated Circuits: A Design Perspective*, Prentice Hall, Englewood Cliffs, N.J., Nov. 1995

[14] H. J. M. Veendrick, "Short-Circuit Dissipation of static CMOS Circuitry and its Impact on the design of Buffer Circuits". *IEEE Journal of Solid static Circuits*, pp. 468-473, August 1984.

[15] S. M. King. "Accurate Simulation of Power Dissipation in VLSI Circuits". *IEEE Journal of Solid state Circuits, 21(5):* pp889-891, Oct. 1986.

[16] A. Tyagi. "Hercules: A Power Analyzer of MOS VLSI Circuits". *Proceedings of the IEEE International Conference on Computer Aided Design.* pp530-533, Nov. 1987.

[17] S. Rajgopal and G. Mehta. "Experiences with Simulation-based Schematic level current Estimation". *Proceedings of the 1994 IEEE International Workshop on Low Power Design.* pp9-14, April 1994.

[18] R. Burch, F. N. Najm, P. Yang and D. Hocevar. "A Monte Carlo Approach for Power Estimation". *IEEE Transactions on VLSI Systems*, 1(1): pp63-71, March 1993.

[19] A. Salz and M. A. Horowitz. "IRSIM: An Incremental MOS Switch-level Simulator". *Proceeding of the 26th Design Automation Conference*. pp173-178, June 1989.

[20] C. Deng. "Power Analysis for CMOS/BiCMOS Circuits". *Proceedings of the 1994 IEEE International Workshop on Low Power Design.* pp3-8, April 1994.

[21] B. J. George, D. Gossain, S. C. Tyler, M. G. Wloka and G. K. H. Yeap. "Power Analysis and Characterization for Semi-custom Design". *Proceedings of the 1994 IEEE International Workshop on Low Power Design.* pp215-218, April 1994.

[22] J. Bhasker. *A VHDL Primer*. Third Edition. 1999.

[23] Steven J.E. Wilton and Norman P. Jouppi, "An Enhanced Access and Cycle Time Model for On-Chip Caches". *WRL Research report*, May 1993

[24] V. Zyuban and P. Kogge, "The Energy Complexity of Register Files", *Proceedings of the 24th Annual International Symposium on Computer Architecture*. pp.130-145, June 1997

[25] S. Palacharla, N. Jouppi and J. Smith. "Complexity-Effective Superscalar Processor". *Proceedings of the 24th Annual International Symposium on Computer Architecture*. pp.206-218, June 1997.

[26] S. Asai and Y. Wada, "Technology Challenges for Integrated Near and Below 0.1um". *Proceedings of the IEEE*, Vol85, No.4, 1997.

APPENDIX

SIMULATION BENCHMARK

Benchmark 1: ALU

```
.text 0x0       lhi     r31, 0xffff
                addui   r30, r0, 0xff00
                or      r29, r31, r30
                add     r28, r29, r30
                addi    r27, r28, 0x7777
                addi    r26, r0, -0x7fff
                addui   r25, r0, 0x7fff
                and     r24, r26, r25
                ori     r23, r24, 0xFF00
                andi    r22, r27, 0x7
                sra     r21, r29, r22
                srai    r20, r29, 0x3
                srl     r19, r29, r22
                srli    r18, r29, 0x3
                sll     r17, r29, r22
                slli    r16, r29, 0x3
                sub     r15, r29, r27
                subu    r14, r29, r27
                subi    r13, r29, 0x7777
                subui   r12, r29, 0xE777
                xor     r11, r26, r22
                xori    r10, r26, 0x3
                trap    0
```

Benchmark 2: Branch

```
.text 0x0           addi   r1,r0,0x1111
                    addi   r2,r0,0x0003
                    multu  f3,f1,f2
                    bnez   r3,LowWordLabel     ; taken, predict as not taken
                    multu  f3,f3,f3            ; not in program flow
LowWordLabel:       beqz   r1,ShutdownLabel1 ; not taken, resolved as not taken
                    addi   r4,r0,0x0001
                    beqz   r4,ShutdownLabel1   ; not taken, predict as not taken
HighWordLabel:      multu  f3,f2,f2
                    add    r4,r3,r3
                    subi   r2,r2,0x0001
                    bnez   r2,HighWordLabel    ;1. taken, predict as not taken
                                               ; 2. taken, predict as taken
                                               ; 3. not taken, predict as taken
                    addi   r31,r0,0x0077
                    multu  f31,f31,f31
                    addi   r4,r2,0x7fff
                    addi   r4,r2,0x7fff
                    addi   r4,r2,0x7fff
                    addi   r4,r2,0x7fff
                    addi   r4,r2,0x7fff
                    j      ShutdownLabel2
                    multu  f3,f3,f3            ; not in program flow
                    addi   r7,r0,-0x0001
ShutdownLabel1:     multu  f4,f3,f3
                    trap   0        ; wait until pipeline is empty, set HaltFlag
ShutdownLabel2:     add    r2,r2,r2         ; source B is destination of A
                    multu  f3,f2,f2
                    bnez   r4,ShutdownLabel1   ; taken, resolved as taken
```

Benchmark 3: Bubble

```
.text 0x0           jal     Program
                    trap    0x104
                    trap    0
.text 0x2000
Program:
                    addui   r1,r0,50        ; Sort 50 numbers ( Bubble-Sort )
                    subi    r1,r1,1         ; Index 0 ... n-1
                    sll     r1,r1,1         ; Sort half-words
                    add     r2,r0,r0        ; r1: SortedFlag, list not sorted
RunAgain:           slti    r3,r1,0         ; List sorted ?
                    bnez    r3,Ready        ; Yes
                    add     r3,r0,r0        ; Index := 0
                    addui   r2,r0,1         ; SortedFlag, list sorted
Compare:            sge     r4,r3,r1            ; End of unsorted head of list?
                    bnez    r4,EndCompare       ; Yes
                    lh      r5,List(r3)         ; Read
                    lh      r6,List+2(r3)       ; Read successor
                    addui   r3,r3,2             ; Increment pointer (half-word)
                    sle     r4,r5,r6            ; Right order ?
                    bnez    r4,Compare          ; Yes, compare next

                    sh      List-2(r3),r6       ; Wrong order, swap, r3 was
                                                ; incremented
                    add     r2,r0,r0            ; SortedFlag, list not sorted
                    sh      List(r3),r5
                    beqz    r0,Compare          ; Branch always

EndCompare:         bnez    r2,Ready            ; Ready when list sorted
                    subi    r1,r1,2             ; Next run to previous index
```

```
                j       RunAgain
Ready:          ; Print sorted list to file, output file is 'dlx.dump'
                lhi     r16,0xffff
                addui   r17,r0,0xff00
                add     r16,r16,r17         ; Base address of external
registers
                addui   r17,r0,List
                trap    0x104               ; Wait until Write-Buffer is
empty,
                                            ; avoid merge
                sw      0(r16),r17          ; Start address of memory
block
                addui   r17,r0,60
                trap    0x104
                sw      4(r16),r17          ; Number of elements ( 60 )
                trap    0x104
                sw      12(r16),r0          ; Start transfer of half-words
                jr              r31         ; Done
.text 0x3000
List:           .word   0x7b878e63
                .word   0x4d36a53a
                .word   0x34a56f27
                .word   0x8902ba5f
                .word   0x02bd8efb
                .word   0x296ac736
                .word   0x77777777
                .word   0x878e28df
                .word   0x0fc46ba8
                .word   0x14fd3556
                .word   0x348efc22
```

```
.word  0x1904f6dc
.word  0x28b2d2ab
.word  0x6f278e63
.word  0x2972befa
.word  0x293c62f3
.word  0x8c7f22b9
.word  0xa3289281
.word  0x4d36bedc
.word  0xfff27191
.word  0x282fca29
.word  0x1c227101
.word  0x21761efd
.word  0x7b8728fd
.word  0xd27a3bc2
```

Benchmark 4: Fibinarci

```
.text 0x0000        ; Reset
                    jal     Program         ; Never executed
                    trap    0x104           ; Wait until Write-Buffer is empty
                    trap    0               ; Halt
.text 0x0100        ; Store Transfer-Error
                    trap    0x104           ; Wait until Write-Buffer is empty
                    trap    0               ; Halt
.text 0x0A00        ; External Interrupt
                    lw      r30,-0xc0(r0)   ; 0xffff_ff40, Interrupt Acknowledge
                                            ; r30 can be used to determine the
device
                                            ; that caused the interrupt
                    nop
                    rfe     0               ; Assembler needs dummy parameter

.text 0x2000
Program:            ; Evaluate some Fibonacci numbers
                    addui   r1,r0,List + 4  ; r1: Pointer to list of numbers (Word )
                    addui   r2,r0,1         ; Initialise register
                    sw      -0x80(r0),r2    ; 0xffff_ff80 Interrupt-Enable register,
                                            ; enable Interrupt
                    add     r3,r0,r2                ; Initialise register
                    lhu     r4,CountLoops(r0)       ; Run loop 3 times
                    sw      -4(r1),r2               ; Store first Fibonacci number
(1)
Loop:               ; Two Fibonacci numbers per run
                    sw      0(r1),r3
                    add     r2,r2,r3        ; Compute next Fibonacci number
                    sw      4(r1),r2
```

```
                addui   r1,r1,8         ; Increment pointer
                add     r3,r2,r3        ; Compute next Fibonacci number
                sub     r4,r4,1         ; Decrement counter
                                        ; Yes, it would be better to do this two
                                        ; instructions earlier. But in this
                                        ; program I want to demonstrate
unresolved
                                        ; branches and speculative execution.
                bnez    r4,Loop         ; Run again ?
                ; Print list to file, output file is 'dlx.dump'
                addui   r7,r0,List
                trap    0x104    ; Wait until Write-Buffer is empty,  avoid
merge
                sw      -0x100(r0),r7 ; Start address of memory block
                addui   r7,r0,20
                trap    0x104
                sw      -0xfc(r0),r7    ; Number of elements ( 20 )
                trap    0x104
                sw      -0xf8(r0),r0    ; Start transfer of words
                                        ; Force Store-Error
                sw      -0x1000(r0),r0 ; The exception handler  will halt the
DLX
                jr              r31     ; Done
CountLoops:  .word  0x00030000
.text 0x3000
List:
```

Benchmark 5: Loadstore

```
.text 0x0
        lw      r1,Data0(r0)
        lb      r2,7+Data0(r0)
        lbu     r3,7+Data0(r0)
        lb      r4,Data1(r0)
        lbu     r5,Data1(r0)
        lh      r6,6+Data0(r0)
        lhu     r7,6+Data0(r0)
        lh      r8,Data1(r0)
        lhu     r9,Data1(r0)
        sw      DataOut(r0),r8      ; Canceled by next sw
        sw      DataOut(r0),r1      ; Merge
        lw      r10,DataOut(r0)     ; Forward
        sb      DataOut+8(r0),r5    ; Not merge
        lw      r11,DataOut+4(r0)   ; Wait and merge
        multu   f12,f4,f5
        multu   f12,f4,f5
        sh      DataOut+8(r0),r1
        sb      DataOut+8(r0),r2    ; Merge entrance and queue
        lw      r13,DataOut+8
        ; Print numbers to file, output file is 'dlx.dump'
        lhi     r16,0xffff
        addui   r17,r0,0xff00
        add     r16,r16,r17         ; Base address of external registers
        addui   r17,r0,0x200
        trap    1     ; Wait until Write-Buffer is empty, avoid merge
        sw      0(r16),r17          ; Start address of memory block
        addui   r17,r0,72
        trap    1
```

```
                sw      4(r16),r17      ; Number of elements ( 72 )
                trap    1
                sw      8(r16),r0       ; Start transfer of words
                trap    0
.text 0x200
Data0:
        .word 0x88776655
        .word 0x0000ffff
Data1:
        .word 0x44332211

.text 0x300
DataOut:
```

Benchmark 6: Prime

```
.text 0x0
                jal     Program
                trap    0x104
                trap    0
.text 0x2000
Program:
                addui   r1,r0,20        ; Find 20 Prime numbers
                                        ; Number of Modulo-Operations: 177
                add     r2,r0,PrimeNumberList   ; Address for next number

                sll     r3,r1,2         ; Multiply by 4
                add     r3,r3,r2        ; End-address of list

                addui   r4,r0,2         ; 2 is the first Prime number
                sw      0(r2),r4        ; Write into list
                addui   r2,r2,4         ; Increment address

                addui   r5,r0,3         ; r5: TestNumber, start with 3
SearchNext:     addui   r6,r0,PrimeNumberList+4 ; Check found numbers, do
                                                ; not check 2
TestLoop:       sge     r7,r6,r2        ; All previous numbers checked ?
                bnez    r7,NumberFound          ; Yes
                lw      r7,0(r6)        ; Load found Prime number
                divu    f8,f5,f7    ;Quotient := TestNumber/OldPrimeNumber
                multu   f8,f8,f7    ; Product := Quotient * OldPrimeNumber
                seq     r8,r8,r5    ; Set flag if remainder of division is zero
                bnez    r8,NoPrimeNumber
                addui   r6,r6,4     ; Increment pointer to next found number
                j       TestLoop
```

```
NumberFound:    sw      0(r2),r5        ; Store new prime number
                addui   r2,r2,4   ; Increment pointer for next Prime number
NoPrimeNumber:  addui   r5,r5,2         ; Increment TestNumber
                slt     r7,r2,r3        ; Search next Prime number ?
                bnez    r7,SearchNext           ; Yes
                ; Print numbers to file, output file is 'dlx.dump'
                lhi     r16,0xffff
                addui   r17,r0,0xff00
                add     r16,r16,r17     ; Base address of external registers
                addui   r17,r0,0x3000
                trap    0x104           ; Wait until Write-Buffer is empty,
avoid
                                        ;merge
                sw      0(r16),r17      ; Start address of memory block
                addui   r17,r0,32
                trap    0x104
                sw      4(r16),r17      ; Number of elements ( 32 )
                trap    0x104
                sw      8(r16),r0       ; Start transfer of words
                jr      r31             ; Done

.text 0x3000
PrimeNumberList:
```