# Issue Logic with Issue Table

Shiao, Feng-Jiann

Department of Computer Science and Engineering

Tatung Univeristy

maco@amigo.ttu.edu.tw

Shieh, Jong-Jiann

Department of Computer Science and Engineering

Tatung University

shieh@ttu.edu.tw

## Abstract

In order to enhance the computer performance, nowadays microprocessors use superscalar architecture. But the superscalar architecture is unable to enhance the performance effectively due to two reasons. One reason is the complexity design will reduce the clock frequency seriously and another reason is data dependency makes the instructions parallelism unable to break the dataflow limitation.

In this paper, a speculative wakeup logic is used to exploit the instructions parallelism. In order to issue more instructions every cycle, an issue table is added to help the select logic select the suitable instructions to issue. Simulation results show the average IPC is increased by 22.5% in SPECInt and 45% in SPECfp over a conventional architecture. If the issue table is removed from our model, the IPC will reduce 6.4% in baseline and 14% in perfect configurations

**Keywords:** issue logic, issue table, superscalar, speculation

## 1 Introduction

In order to achieve higher processor performance, researches have been focused on increasing the instruction-level parallelism (ILP). ILP is measured as the average number of instructions committed per cycle (IPC). In the past twenty or more years, pipeline depths have grown from 1 (Intel 286), now up to more than 20 (Intel Pentium 4) [1]. In the future the pipeline depth will continue increasing to exploit more parallelism.

A high IPC rate implied hardware has to fetch and issue multiple instructions in parallel. The conventional RUU (Register Update Unit) architecture [2] was set up to solve the problem of data and control dependence, and enhances the effective of issue logic.

In this paper we will introduce a new issue logic to select instructions for execution. It adopts the speculative aspect [3] to wakeup more instructions and then use the issue table to select the suitable instructions to issue. Construction of the dynamic table depends on the instructions of different priority level. The issue table is so small that compare, search, and update table can be done in a small time that will not affect the clock cycle time. That is, we can improve the issue logic and it doesn't cost us much.

The rest of the paper is organized as follows: Section 2 introduces related works necessary for understanding this study. Section 3 describes the design of the proposed issue logic. Section 4 presents the evaluation methodology, and section 5 analysis the performance. Finally, we summarize this study in section 6.

## 2 Related work

This section we will introduce the related works necessary to help for understanding our study. Section 2.1 introduces the pipeline model. Section 2.2 presents the conventional scheduling. Section 2.3 presents the related paper to guide our research.

### 2.1 Pipeline Model

Figure 2.1 shows the pipeline of a typical conventional superscalar out-of-order processor. The pipeline has 7 stages: fetch, decode, rename, schedule, register read, execute/bypass, and commit. Some stages of the pipeline require more than one cycle.


Figure 2.1: Processor Pipeline

First, fetch unit fetches instructions from the instruction cache. The instructions are decoded, registers are renamed and then all these information are placed in the RUU.

When the resources are available, the

instructions will be issued to functional unit for execution. After finishing the execution, the RUU checks whether other instructions in RUU depend on the completed instructions. If some instructions are ready, they will be scheduled (that is, this instruction wakeup and wait for select) to issue.

2.2 Conventional Scheduler

Figure 2.2 shows the conventional execution core [4]. The dynamic scheduler includes three pieces of logic: rename logic, wakeup logic, and select logic.

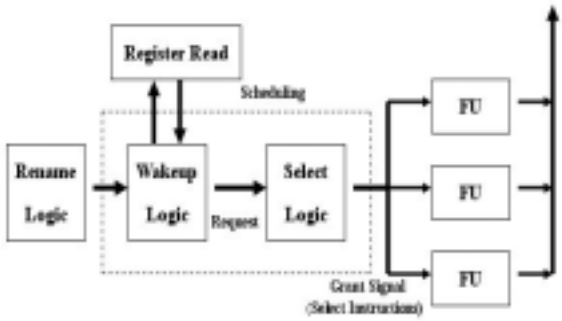


Figure 2.2: Conventional Execution Core

**Wakeup Logic**: Figure 2.3 [3] shows the conventional scheduling which focus on wakeup logic. It detects dependencies between instructions in the same cycle. The sources of each instruction are compared to the destinations of all previous instructions in the same cycle.

In decode stage, each RUUE (Register Update Unit Entry) contains information about instruction's sources, such as whether the source is ready, source tag which was used to compare the result tag, the readyL bit and readyR bit of instruction are initialized, and physical register identifier(tag) for the source is recorede.

If the Ins(R) has the result written but Ins(L) doesn't, the ready(R/L) of instruction are initialized to 1/0. That is, instruction sleeps in the RUU waiting for the readyL to be set. The readyL bit and readyR bit for each source are set when both the data for that source are available in the register file or is available for bypass from a function unit. This time, the wakeup logic sends the request signal indicating which instructions are ready for execution.

**Select Logic:** In Figure 2.3, we can see the select logic choices instructions from those marked Request signals for execution onto a given functional unit. In conventional architecture design, the select logic contains a prioritizer which typically takes the oldest instructions from the request signals.

It is possible to select more than one instruction per cycle to execute on multiple functional units. After execution, instruction's destination tag will be broadcasted to the common data bus (CDB). Instructions have different execution latencies, the wakeup logic just need wait the tag broadcasted to CDB.

2.3 Advanced Scheduler

Select-Free scheduling logic [14] describes a technique that break the scheduling (wakeup and select) logic into two smaller loop: a critical loop for wakeup and a non-critical for select. With select-free scheduling logic, this will solve the collisions (where more instructions wakeup than can be select, resulting in a mis_speculation) and pileups (dependents of the collision victims may wake up before they are really ready to be scheduled, that is entering the scheduling pipeline too early) problems. The authors introduce the select-N schedulers and predict another wakeup (PAW) to avoid the collision.

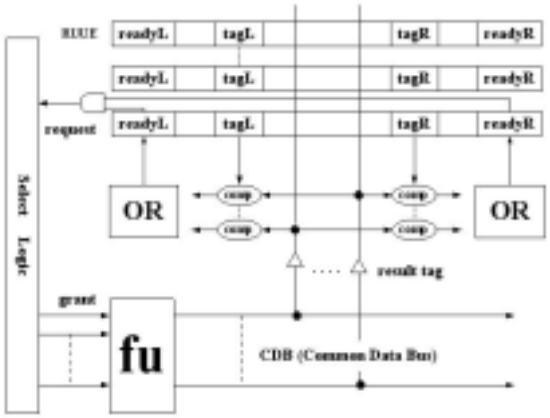


Figure 2.3: Detail Circuit of Conventional Wakeup Logic

Low-Complexity Issue Logic [5] reduces the hardware load of associative look-up, the consumption of power, and clock-rate. More effective issue logic used the dynamic manners to control the instruction issue logics were proposed. These mechanisms [3, 6] can save power and issue multiple instructions. In the framework of Wang and Wu [6], by avoiding the delay of pre-decode stage the issue rate will be improved. Recently, issue logic with cluster of instructions promotes more effective performance but takes too much time [7] [8].

Pipelined scheduling with speculative wakeup was proposed in [9], which pipelines this logic over 2 cycles while still allowing back-to-back execution of dependent

instructions. This technique pipelines the scheduling logic without eliminating its ability to execute dependent instructions in consecutive cycles. This will exploit more parallelism in processors, and then allow processors to achieve higher performance.

A new scheduling scheme was introduced that uses matrices, to represent the dependent between instructions [10]. The distances between two dependent instructions are generally short. This will reduce the effective size of the matrices for small IPC penalties.

## 3 The Design of Issue Logic

In this section, we will explain how to design the issue logic and how to operate in superscalar in detail.

### 3.1 Speculative Wakeup

In superscalar architecture, wakeup logic can wake up several instructions in one cycle and then select logic will select the higher priority instructions that waked up. If we can wake up more instructions efficiently in one cycle, how to select the suitable instructions for execution is very important.

If the sources of an instruction are ready as shown in figure 2.3, the Grant signal will be triggered in wakeup circuit. There are three kinds of situations in Speculative wakeup logic: one, the left source is ready, the parents of right source are ready, but right source is not ready; another, the right source is ready, the parents of left source are ready, but left source is not ready; the third, the right source and left source are not ready, but the parents of left source and the parents of right source are ready.

### 3.2 Issue Logic

After fetching the instructions, the instructions will be dispatched to RUU. The RUU will be allocated some basic information such as instruction bits, instruction op, spec_mode of the instruction, status of the instruction, and so on. In decode stage, we proposed the Priority table as shown in Figure 3.1. The Priority table is established to help issue logic select the suitable instructions from many waked up instructions for execution.

| Instruction' Category | level |
|---|---|
| Loads or Stores Instructions | 4 |
| Control Instructions | 4 |
| Long Latency Instructions | 4 |
| Other Instructions | 1 |

Figure 3.1: Priority Table

| Slot tag | Spec_mode | level |
|---|---|---|

Figure 3.2: Entry of Issue Table

The Priority table was divided into levels, according to Loads or Stores Instructions、Control Instructions 、Long Latency Instructions and general instructions. In process of allocation in decode stage, the decoder will refer the Priority table to judge what kind of level the instruction belongs to.

We add the issue table which was divided into 4 level to help the issue logic select the suitable and highest priority instruction for execution. Figure 3.2 shows the entry of issue table. The level value and slot tag of RUU are recorded in the issue table. A 2-bit level is added to each entry in the RUU, this will easy the finding of the right issue level table.

Spec_mode bit of the issue table entry shows whether the instruction is speculative or not. If mis_speculatve happen, the instructions will be flushed which have spec_mode value 1 in the RUU and in issue table.

Figure 3.3 shows an execution core with issue table. Before allocating information to the RUU, the processing instruction compares the instructions which are in RUU to see whether the instruction depends on another instruction which is not issued. If the instruction depends on another instruction, the instruction's entry of Dslot tag will record the dependent instruction's slot tag and level. If the issue logic finds such instruction, the issue logic will find the entry's level of the instruction in the RUU.

When the issue logic finds the issue level table where the instruction is, the issue logic enhances one level not only in the RUU but also in issue level table. In addition, the issue logic will move the entry from low issue level table to high issue level table.

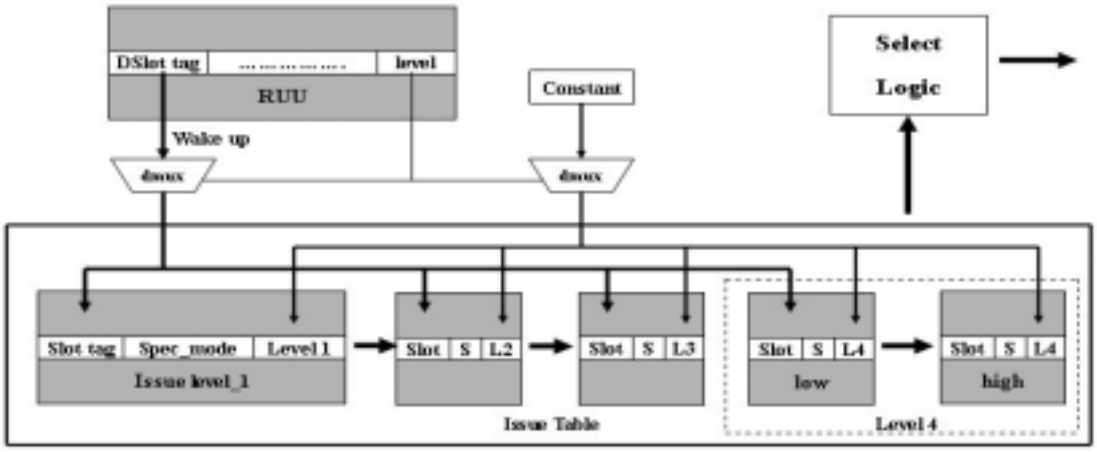


Figure 3.3: Execution Core with Issue table

That is, if the instruction is in issue level 1 table and another instruction depends on this instruction, the issue logic will add 1 to the level in the RUU and add 1 to the level in the issue level table. Finally, the issue logic moves the instruction from the level 1 to level 2.

The issue table is divided into 4 levels. When the instruction is wakeuped, the instruction's information is allocated onto issue table by means of the level of RUU. For example, the load instruction's level value is 4 in the entry of RUU. The load instruction is dispatched into the issue level_4 table. The general instructions are only dispatched to issue level_1 table. When other instructions depend on this instruction, the instruction's level is increased and the instruction is moved into high issue table.

The select logic picks instructions for execution from the highest level (issue level 4 table). If there is no instruction in level 4 table, the select logic goes to the level 3 table and picks the instructions from level 4 table every cycle.

There are two situations we need pay attention in select stage when the select logic picks 4 instructions to execute from issue level 4 table. First, the select logic issues instructions which didn't get the functional unit. That is, the instructions need be issued next cycle. If the instructions which are not issued belong to the general instruction, the issue logic increases level value up to level 3 by the constant. In order to ensure the instruction will be executed early, the level value is enhanced directly to level 3.

Table 4.1: Baseline Architectures Configuration

| | |
|---|---|
| **Inst. Fetch** | 4 lines per cycle. Only one taken branch per cycle. |
| **Branch Predictor** | 1. gshare, with 10-bit history register and 16K entry counter table.<br>2. hybrid: gshare + bimodal (default)<br>3. perfect<br>Return stack with 64 entries. |
| **Out-of-Order Execution Mechanism** | Issue of 4 operations/cycle, 128 entry RUU (which is the ROB and the IW combined), 32 entry load/store queue. Loads executed only after all preceding store addresses are known. Value bypassed to loads from matching stores ahead in the load/store queue. |
| **Architected Registers** | 32 interger, hi, lo, 32 floating point, fcc. |
| **Functional Units (FU)** | 4-integer ALUs, 2 load/store units, 2-FP adders, 1-Integer MULT/DIV, 1-FP MULT/DIV |
| **L1 D & I-cache** | 1024K bytes, 4-way set assoc., 64 byte line, LRU 1 cycle hit latency. |
| **L2 D & I-cache** | Unified, 2048K bytes, 4-way set assoc., 128 byte line, LRU, 6 cycles hit latency |
| **Memory** | Memory access latency (first-18, rest-2) cycle. Width of memory bus is 8 bytes. |
| **BTB** | 512-entry |
| **Mispredict penalty** | 3 cycles |
| **TLB Miss** | 30 cycles |

Table 4.2: Instruction Class Latencies

| Instruction Class | Latency (in Cycles) |
|---|---|
| integer arithmetic | 1 |
| integer multiply | 4, pipelined |
| fp arithmetic | 8, pipelined |
| fp divide | 16 |
| loads and stores | 1 + dcache latency |
| all others | 1 |

## 4 Simulation environment

We use an executable-driven simulator to simulate the enhanced design. The simulator is simplescalar 3.0 tool suit [11] which was implemented by Wisconsin University.

To perform our experimental study, we simulate all models with SPEC 2000. The programs were compiled with the gcc compiler include in the tool set. We simulated all benchmarks 500 million instructions. All machines are superscalar processors with out-of-order execution. Table 4.1 summarizes the parameters used in our baseline architectures.

They required 2 cycles for fetch, 2 for decoded, 2 for rename, 1 for register read, and 1 for commit. The ideal machine required 1 cycle for scheduler, and the other required 2 cycles. Table 4.2 present the different classes of instruction latency.

## 5 Performance Analysis

Figure 5.1 shows the IPC of the two machines over the SPEC2000 benchmarks. We compare the baseline with our model (maco), baseline_p, and maco_p IPC. The result shows maco enhance the average IPC of 22.5% in SPECInt and IPC of 45% in SPECfp than baseline. The baseline_p and the maco_p are not limit in instruction widths, RUU entries, and functional units. The result shows maco_p enhance the average IPC of 55% than baseline_p. The only limits on ILP in such a processor are true data dependences without speculation.

Figure 5.2 shows the ratio of wakeuped instructions numbers between the baseline model and our proposed model (maco). More than one time of instructions are wakeuped. As a result of more instructions are wakeuped, we can use our model to select suitable instructions for execution.

Results show more instructions are issued by means of speculative wakeup and issue table. Figure 5.3 shows that on the average maco increases 10% in total numbers of instructions to issue. If more instructions belong to the int-multiply, int-divide, FP-multiply, FP-divide, FP-sqrt, or Mem-port, there are still a lot of instructions waiting to get functional unit.

Figure 5.4 shows that whether the issue table in our model affects the performance. We compare our model with issue table in baseline and perfect (maco_p) architectures configuration. If the issue table is removed from our model, the IPC will reduced 6.4% in baseline (maco) and 14% in perfect (maco_p) configurations.

Figure 5.5 shows our model without limitation in instruction widths, RUU entries, but with different functional units. Issue_p_2fu have 2 times functional units than issue_p. Due to our model wakeups more instructions, when we increase the number of functional units, the performance will be enhanced.

Figure 5.6 shows the two machines with 4, 8, and 16 instructions width, respectively. We can see obviously that enhance the width from 4 to 8 will improve the performance. In baseline model, enhance width from 8 to 16 will not improve the performance notably in SPECint2000. That is, if we just enhance the width, the performance will enhance less and less. This is because there are not enough instructions waked up for issue. Furthermore, the performance will lower in SPECfp2000. One reason is not enough instructions are waked up and another is more instructions are long latency. Our model will solve this problem and select suitable instructions, so when we enhance the width from 8 to 16, the performance is still promoted.

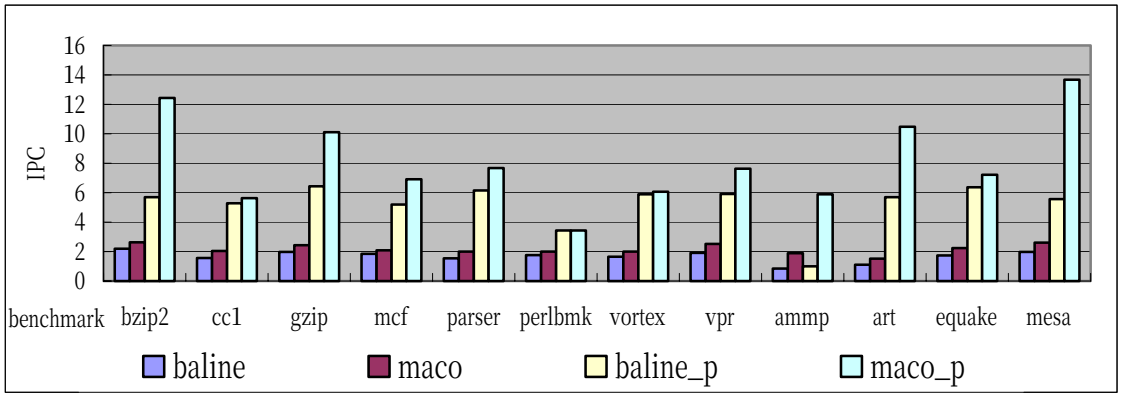


Figure 5.1: ILP available in a processor

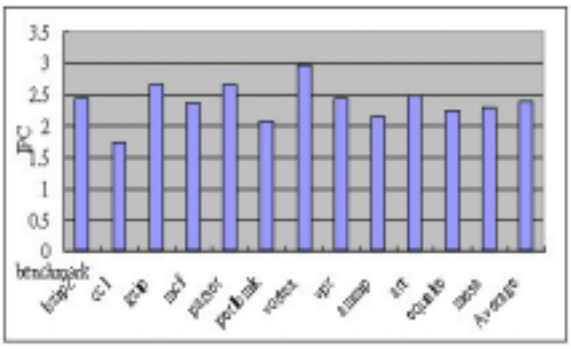


Figure 5.2: Ratio of instructions wakeuped

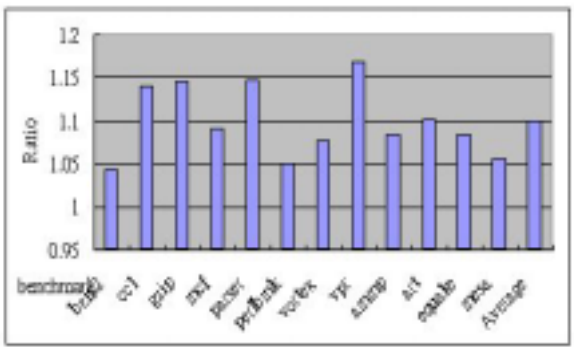


Figure 5.3: Ratio of instructions issued

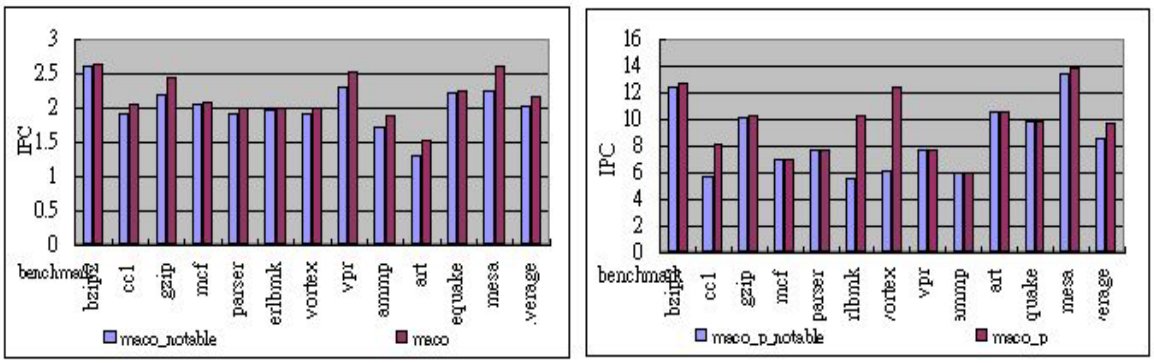


Figure 5.4: Issue table effectiveness

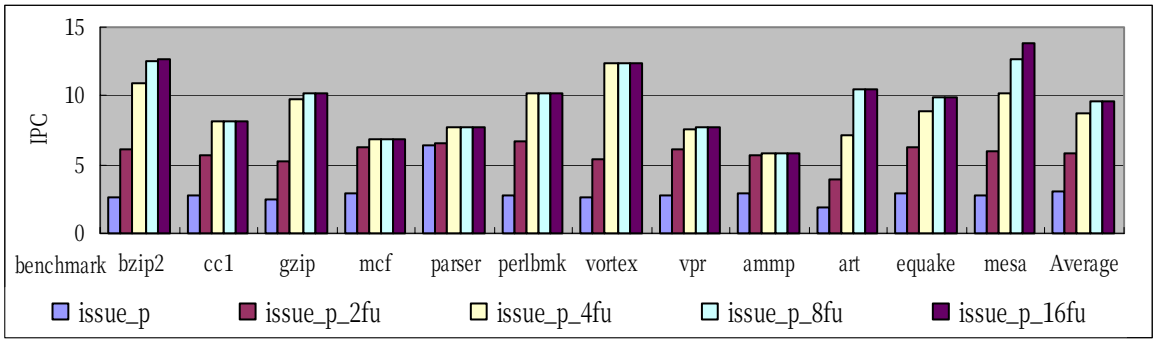


Figure 5.5: The effect of numbers of Functional Units

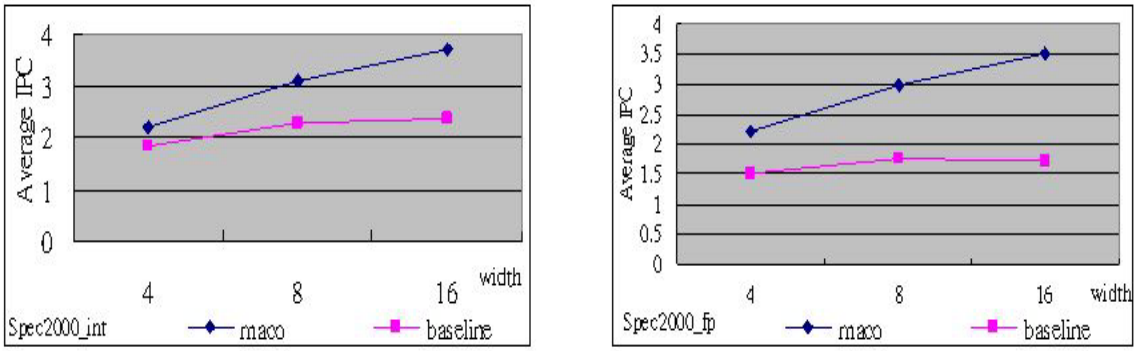


Figure 5.6: Comparison of the Two Machines for Different Instruction Width

## 6 Conclusions

In modern superscalar processor, the main part of dynamic scheduling is wakeup and select logic. Issue logic in out-of-order is the most important part of our research. The issue table is attached to the RUU to help select logic issues the suitable instructions to functional units. Conventional issue logic is not very efficient because they don't issue the suitable instructions to functional units. That is, there are many instructions to wait to issue. This will cost a lot of time and then drop the performance.

Simulations of the 2 machines using both 4 and 16 functional units and several different decode, issue, and commit width are conducted. The result for a superscalar microprocessor present that these mechanisms in issue logic wake up more than one time instructions and increase 10% in total number instructions to issue over the typical machine. Compare our model with baseline machine, the average IPC is increased by 22.5% in SPECInt and 45% in SPECfp. If the issue table is removed from our model, the IPC will reduced 6.4% in baseline (maco) and 14% in perfect (maco_p) configurations.

## 7 Acknowledgements

This work was supported by National Science Council, Taiwan, under contract NSC 91-2213-E-036-004.

## References

[1] Inter Corporation. IA-32 Intel Architecture Software Developer's Manual Volume 1: Basic Architecture, 2001.

[2] Gurindar S. Sohi, “Instruction Issue Logic for High-performance, Interruptible, Multiple Functional Unit, “ Pipelined Computers, *IEEE Transaction on Computers*, 39(3): 349 – 359, March 1990.

[3] Folegnani,D.;Gonzalez,A., ”Energy-effe ctive issue logic” , *Computer Architecture, Proceedings. 28th Annual International Symposium on*, pp. 230 -239, 2001.

[4] Brown, M.D.; Stark, J.; Patt, Y.N.; “ Select-free Instruction Scheduling Logic,“ *Microarchitecture, MICRO-34. Proceedings. 34th Annual ACM/ IEEE International Symposium on*, pp.204 - 213, 2001.

[5] Ramon Canal ; Antonio Gonzalez, ”A Low-Complexity Issue Logic”, *Proceedings of the 14th international conference on Supercomputing*, pp. 327-335, 2000.

[6] Wang ; C.L. Wu, ”I-NET mechanism for issuing multiple instructions”, *Proceedings of the 1988 ACM/IEEE conference on Supercomputing* , pp. 88–95, 1988.

[7] Baniasadi, A.; Moshovos, A., ”Instruction distribution heuristics for quad-cluster, dynamically-scheduled, superscalar processors”, *Microarchitecture, MICRO-33. Proceedings. 33rd Annual IEEE/ACM International Symposium on*, pp. 337-347, 2000.

[8] Cotofana, S.; Vassiliadis, S., ”On the design complexity of the issue logic of superscalar machines”, *Euromicro Conference, Proceedings. 24th*, vol.1 , pp. 277-284, 1998.

[9] Stark, J.; Brown, M.D.; Patt, Y.N., ”On pipelining dynamic instruction scheduling logic”, *Microarchitecture, MICRO-33. Proceedings. 33rd Annual IEEE/ACM International Symposium on*, pp. 57 -66, 2000.

[10] M. Goshima, K. Nishino, Y. Nakashima, S.I. Mori, T. Kitamura, S. Tomita, “A High-Speed Dynamic Instruction Scheduling scheme for Superscalar Processors” *Microarchitecture, MICRO-34. Proceedings. 34th Annual ACM/ IEEE International Symposium on*, pp. 225 – 236, 2001.

[11] Doug Burger*, " Evaluating Future Microprocessors: the SimpleScalar Tool Set," *Technical Report CS-TR96 -1308*, University of Wisconsin Madison, 1996.