Proceedings of International Conference
on Computer Architecture

A Method of Improving Translating Performance in the CISC/RISC Hybrids

Wen-Bin Jian and Chang-Jiu Chen

Department of Computer Science and Information Engineering
National Chiao Tung University
Hsinchu, Taiwan, R.O.C.

Abstract

This paper proposes a method to improve the translating performance in the CISC/RISC hybrid processors. This method introduces the design of a new decoder architecture which contains an instruction scheduler and a new translator type with a combination of one simple, one general and one complex translators (1S+1G+1C). To improve the translating performance is to increase the number of CISC instructions translated per cycle, which is the goal of this paper.

We build a decoder architecture model for the proposed method to measure its translating performance. Besides, two different decoder architecture models are built for comparison. These three models are used to evaluate the effect of the different design issues for the translating performance. These issues include instruction mix, instruction dependencies, translator type, scheduler, and search window size.

The evaluation results show that the model for the new decoder architecture with 1S+1G+1C translators and a scheduler performs better than other models. In addition, the results also show that the new decoder architecture has the ability to translate more instructions every cycle than other current CISC/RISC hybrid microprocessors do in average.

1 Introduction

In recent years, CISC (*Complex Instruction Set Computer*) /RISC (*Reduced Instruction Set Computer*) hybrid architectures have been one of the main trends in the computer architectures[1,2,3,11]. A CISC/RISC hybrid processor means that the processor has a RISC execution core but executes CISC programs. In othre words, the CISC/RISC hybrid processor executes CISC programs in the RISC manners. Many companies have introduced their CISC/RISC hybrid microprocessors. NexGen's Nx686, AMD's K5 and Intel's P6 are the most well-known hybrid microprocessors [2,3,11]. They have a common feature in which x86 programs are executed with RISC manners. However, the methods of how these x86 programs are translated and how they are executed differ from one processor to another.

Improving the execution performance in the CISC systems is difficult because of the inherent restrictions in CISCs. Every performance enhancing trick developed recently in the research for microprocessors is hard to be implemented in a CISC system. One way to execute CISC programs efficiently is to execute them in a RISC system. It implies that it is necessary to translate instructions from CISC to RISC.

Some CISC instructions are simple operations which can be translated directly to internal instructions which roughly correspond to the RISC instructions. However, some CISC instructions are complex. They usually contain more than one simple operations. These instructions are cracked and translated into internal instructions before they are executed.

We use the term "microinstruction" to describe the internal format of the internal instruction words, since they are actually like the microinstructions in the microprogrammed control unit. For convenience in the representation in the article, the internal instructions are expressed much like RISC instructions rather than microinstructions.

2 The Proposed Method

2.1 The New Decoder Architecture

In this section, we introduce a new decoder architecture to improve the translating performance, and describe how it works. The decoder architecture is shown in Figure 1.

The prefetch/predecode unit is used to predecode CISC instructions as they are fetched from memory and feed them to the instruction cache. The predecode information is stored in the instruction cache.

The scheduler unit arranges the instructions fetched from the instruction cache to get better instruction mix such that at most time there are as many instructions translated as possible. The scheduling algorithm schedules instructions by knowing the information of an instruction boundary and the number of microinstructions. The instructions scheduled are stored in the mapping table. The details for the scheduler are described in the next section.

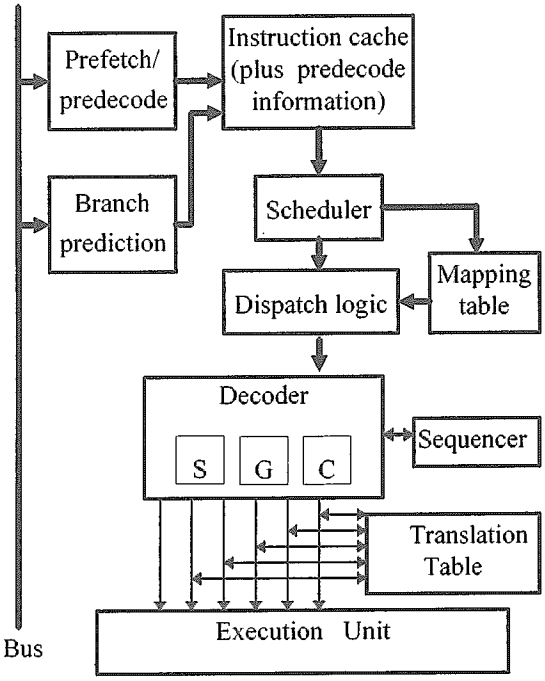
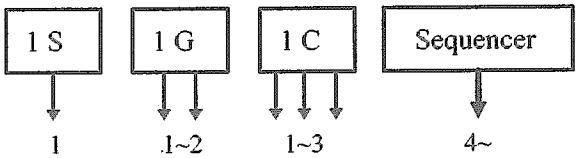


Figure 1 The block diagram of the new decoder architecture.

The dispatch unit dispatches as many instructions to the translators every cycle from the mapping table. In fact, the translators can handle three instructions per cycle in maximum. However, whether an instruction can be dispatched or not depends on the number of microinstructions it requires and what kind of translators are available. A CISC instruction requiring three microinstructions can be translated only by the complex translator (C); an instruction implemented with two microinstructions can be translated by the complex (C) or the general (G) translator; and the simple instruction requiring one microinstruction can be translated by the complex (C), the general (G) or the simple (S) translator. On the other hand, instructions with more than three microinstructions can be translated only by the sequencer. Before dispatchingthe instruction sequences are recorded in the mapping table. The mapping table functions as a buffer storing the scheduled instructions which are ready to be sent to the translators by the dispatch unit.



The translation table is a two-dimension array. Each entry of the table has four fields. These fields record the information about instructions ready to be issued to the execution unit. Keeping the information has two purposes. One purpose is to issue instructions to the execution unit, and the other is important for recovery while interrupts occur. This part is described in more detail in Section 5.1. Figure 2 is the diagram of the translation table.

Instruction address	Instruction code	Translator Number	D

Figure 2 The structure of the translation table. Each entry in the mapping table has four fields: Instruction Address records the location of the instruction, Instruction Code field stores the instruction code, the possible translator number for that instruction is in the Translator Number field, and the last field, D field, is one-bit width and keeps the information of whether the instruction is the start of a dispatch stage.

2.2 Scheduler

The scheduler is mainly to schedule (rearrange) instructions within a basic block such that the rearranged instructions can be translated in full speed; that is, no translators are idle in each cycle. In full translating speed, three CISC instructions can be translated and issued pre cycle which results in the highest translating performance. However, many constrains are involved in rearranging instructions so that it is sometimes difficult to get the full translating speed. The scheduler is similar to an optimized compiler for code optimization that involves rearranging instructions [4].

2.2.1 Considerations

Many constrains are involved in rearranging instructions such that the scheduler is not able to schedule instructions to get the full translating speed. These constrains are basically considered to maintain the correctness of program execution. They include:

- Data Dependencies
- Branches
- Interrupts
- Flags or Control Status Words

We can simplify these considerations by analysis. There are three kinds of operand data dependencies to be considered: true data dependencies, antidependencies and output dependencies. However, antidependencies and output dependencies can be eliminated by register renaming. The scheduler can cooperate with the reservation station in performing register renaming. Thereby the considerations for data dependencies

can be simplified to only the true data dependencies. We should note that the register renaming performed here is only for tagging the registers that causing output and antidependencies. The physical register assignment is carried out after the instructions are issued to the reservation station [5].

Branch instructions affect the program execution flow and, therefore, would affect the instruction rearrangement. Instructions preceding a branch, for instance, cannot be moved to the positions succeeding that branch instruction. One way to deal with this problem is to divide the program into basic blocks. The scheduling algorithm is performed in rearranging instructions within the basic block [4].

Considering the instruction scheduling and the occurring of interrupts, the scheduler should maintain the correctness of program execution results rather than the instruction sequence. The scheduler, thus, has to rearrange instructions carefully such that the program with rearranged instructions has the same execution results with the original program when the interrupts occur. In fact, keeping the program execution correct is the basic requirement for the scheduler.In Chapter 5, we have a topic discusse what problems the interrupts would cause and how we could deal them with.

The last consideration for the scheduler is of flags. Some instructions would set the flags or control status words (CSW) after execution, for example (the x86 instructions), ADD, AND, CMP, etc., and some instructions would check the flags or CSW as the execution base before execution, for example (the x86 instructions), JGE, JL, LOOPE, LAHF, etc [6,7]. The instructions of setting flags and checking flags after and before execution form another dependency relationship. As a result, the scheduler has to take care of these instructions to keeps these sequences and to avoid inserting any instruction that would modify the flags and that would check flags for executing into the coupled (setting and checking) instructions.

2.2.2 The Instruction-Mix Types

There are five types of instruction mixes depending on how many instructions can be translated in a cycle classed by us for the scheduler. Besides, it is important that the classification is also based on the translator type of the new decoder architecture.

Type-1 instruction mix indicates that only one instruction, of course the first one, can be translated. This is because that the second instruction requires four or more microinstructions or conflicts with the first one in needing the complex translator. For type-2 instruction mix, two instructions in the arrangement window can be translated per cycle. This is because that the third instruction requires more than three microinstructions or conflicts with the first two instructions in needing of translators. Type 3 is the perfect case that all the three instructions in the arrangement window can be translated in a cycle. Several conditions have to be satisfied to form an instruction mix of type 3. The three instructions in the arrangement window with the first instruction having four or more microinstructions are classified to type 4. As a result, only the first instruction is dispatched to the sequencer, and the other two are stalled. The type 5 is similar to the type 1. It is separated from type 1 because of the convenience for the scheduling algorithm.

2.2.3 Scheduling

The scheduler first reads in a basic block of instruction sequence plus predecoding information from the instruction cache into the instruction queue. The first three entries in the instruction queue are marked as the Arrangement Window. The other entries that the scheduler can search for candidates for arranging are marked as Search Window. The scheduler, then, find the instruction-mix type of the instructions in the arrangement window. If the instruction-mix type is 3, 4 or 5, no rearrangement is necessary. The scheduler records the information for the instructions that would be retired from the arrangement window according to the instruction-mix types in the mapping table. The arrangement window and search window are moved to reflect the fact that some instruction are retired from the arrangement window (or the instruction queue), and to be ready to perform the next scheduling.

Otherwise, if the instruction-mix type is 1 or 2, the rearrangement process starts. The key point of the scheduler is to find the instructions in the search window with the number of needed microinstructions to satisfy the requirements in the arrangement window forming the instruction mix of type 3. If the scheduler finds the proper instructions they are exchanged The exchange implies that the scheduler has to rearrange the instruction sequence within the basic block. Several requirements or constrains described in the above section are required for the instruction rearrangement to maintain the correctness of instruction execution.

Following is an example explaining the scheduling process. Figure 3 lists the x86 instruction sequence of a basic block plus a conditional jump instruction. The number in the front of each instruction indicates the number of microinstructions required for that instruction.

i	3	PUSH	MEM_1
i+1	2	ADD	CX, MEM_2
i+2	3	SUB	MEM_3, CX
i+3	1	MOV	BX, AX
i+4	3	SHR	MEM_3, 1
i+5	2	ADD	AX, MEM_3
i+6	2	XOR	DL, MEM_4
i+7	1	DEC	CL
i+8	4	LODSB	STRING
i+9	1	CMP	AX, BX
i+10	1	JNE	LABEL

Figure 3 Instruction sequence in a basic block plus a conditional branch instruction before scheduling.

Note that the instruction *i+1* is true dependent with instruction *i+2* and *i+7*. So are instruction *i+4* with *i+5*, and instruction *i+5* with *i+9*. Instruction *i+3* and instruction *i+5* are antidependent. The arrangement window size is three because three instructions can be translated at most per cycle. Assume the search window size is six (tunable).

First, instruction *i+2* and instruction *i+3* are exchanged. So now instructions PUSH, ADD and MOV (instruction *i*, *i+1* and *i+2* respectively) are in the arrangement window as show in Figure 4(a), and they will be retired to the mapping table later. In the next step, instruction *i+4* and instruction *i+6* are exchanged, and instruction *i+5* and instruction *i+7* are exchanged, too. It is shown in Figure 4(b).

After retired instructions *i+3*, *i+4* and *i+5*, instructions *i+6*, *i+7* and *i+8* are in the arrangement window. The instruction *i+8*, then is exchanged with instruction *i+9*. Note that, although, instruction *i+9* will set the flags and instruction *i+10* will check the flags before executing, instruction *i+8* can still exchange with instruction *i+9* because instruction *i+8* does not affect the flags. This step is shown in Figure 4(c). Finally, for the basic block, instructions LODSB and JNE are sent to the mapping table.

In contrast, if the instruction sequence in Figure 3 does not be scheduled by the scheduler, these instructions are then directly dispatched to the translators. Figure 5 gives a detailed description for this case. Note that in the case of without scheduling, the instructions do not have to be retired to the mapping table first. In fact, the mapping table is not needed anymore, however. Besides, we do not have to care the instruction dependencies because we do not rearrange instructions. They will be directly dispatched to the translators according to the translators' translating ability. In addition, the translation table is not needed, either.

i	3	PUSH	MEM_1
i+1	2	ADD	CX, MEM_2
i+2	1	MOV	BX, AX
i+3	3	SUB	MEM_3, CX
i+4	3	SHR	MEM_3, 1
i+5	2	ADD	AX, MEM_3
i+6	2	XOR	DL, MEM_4
i+7	1	DEC	CL
i+8	4	LODSB	STRING
i+9	1	CMP	AX, BX
i+10	1	JNE	LABEL

Figure 4(a) Instructions *i+2* and *i+3* are exchanged. Now the instructions *i*, *i+1* and *i+2* form the instruction mix of type 3.

i+3	3	SUB	MEM_3, CX
i+4	2	XOR	DL, MEM_4
i+5	1	DEC	CL
i+6	3	SHR	MEM_3, 1
i+7	2	ADD	AX, MEM_3
i+8	4	LODSB	STRING
i+9	1	CMP	AX, BX
i+10	1	JNE	LABEL

Figure 4(b) Instructions *i*, *i+1* and i+2 are retired to the mapping table. Instruction *i+4* exchanges with *i+6*, so does instruction *i+5* with *i+7* such that instruction *i+3*, *i+4* and *i+5* form the instruction mix of type 3.

i+6	3	SHR	MEM_3, 1
i+7	2	ADD	AX, MEM_3
i+8	1	CMP	AX, BX
i+9	4	LODSB	STRING
i+10	1	JNE	LABEL

Figure 4(c) Instructions *i+3*, *i+4*, and *i+5* are retired to the mapping table. Instruction *i+8* exchanges with instruction *i+9*. Note that, instruction LODSB does not affect the flags.

i+9	4	LODSB	STRING
i+10	1	JNE	LABEL

Fingre 4(d) The final scheduling step. Instructions *i+6*, *i+7* and *i+8* are retired to the mapping table. Instruction *i+9* and *i+10* will be retired individually in the later two steps.

In comparison, the numbers of translation for the eleven instructions for the case of with scheduling and the case of without scheduling are five and six respectively. We can evaluate the translating performance for the two cases by dividing the number of instructions by the number of translation. Thus, the performance of the case with scheduling is 11/5 = 2.2 , and on the other hand, the performance of the case without scheduling is 11/6 = 1.833. Accordingly, the case with scheduling has better performance than the one without scheduling.

i	3	PUSH	MEM_1
i+1	2	ADD	CX, MEM_2
i+2	3	SUB	MEM_3, CX
i+3	1	MOV	BX, AX
i+4	3	SHR	MEM_3, 1
i+5	2	ADD	AX, MEM_3
i+6	2	XOR	DL, MEM_4
i+7	1	DEC	CL
i+8	4	LODSB	STRING
i+9	1	CMP	AX, BX
i+10	1	JNE	LABEL

Figure 5 The instruction sequence without scheduling of Figure 3. The instructions *i* and *i+1* are dispatched first. Instructions *i+2* and *i+3* are the second, then, instructions *i+4* and *i+5* are the third, and the following are instructions *i+6* and *i+7*, then instruction *i+8* is dispatched individually, and finally, instructions *i+9* and *i+10* are dispatched.

3 The Evaluation Method

3.1 Performance Issues

Designing the decoder architecture involves several issues. These issues can be classified into two types: issues for architecture design and issues for program characteristics. The issues for architecture design include whether a scheduler is combined with the translators, what types of the translators, the number of the translators, and the search window size. The issues for program include the ratio of instruction dependencies and the ratio of simple instructions.

In summary, we list these issues mentioned above for the convenience of later discussion.

- I1: Scheduler
- I2: Translator Type
- I3: Search Window Size
- I4: Instruction Mix
- I5: Instruction Dependency

3.2 The Evaluation Method

We construct three models to evaluate the translating performance. The evaluation results provide us some guides or hints in determining or discussing the effects of the issues (I1~I5) to the decoder architecture. The three models are:

- M1: A decoder architecture with a scheduler and one simple translator, one general translator and one complex translator.
- M2: A decoder architecture with a scheduler and with x simple translators, y general translators and z complex translators.
- M3: A decoder architecture without a scheduler and with one simple translator, one general translator and one complex translator.

The x, y and z can be any value depending on how many translators and what types we design. For the reasons of balancing hardware costs for each model and referencing Intel P6's decoder architecture, we assign the x, y and z values as 2, 0 and 1 respectively in model M2. The M1 is our target model, but M2 and M3 are the opposite models.

Three simulators are built for the three models. Two options, translator type and with/without scheduler, are dedicated to each simulator by default. Other options, basic block size, instruction dependency ratio, instruction mix type, and search window size, are controlled as variables in these simulators.

4 The Evaluation Results

Table 1 lists the translating performance in different instruction mix (I4), different search window sizes (I5) and different instruction dependencies (I3) in model M1 which has a scheduler and one simple, one general and one complex translators. In Table 1, (45, 40, 10, 5) means that the number of CISC instructions with one microinstructions is 45%of all, the number of CISC instructions with two microinstructions is 40%, the number of CISC instructions with three microinstructions is 10 % , and the number of instructions with four and more microinstructions is 5 % , and DEP means the ratio of instruction dependency, and SWS indicates the size of the search window.

From Table 1, we will see that the performance is increased if we widen the search window or decrease the instruction dependency ratio or increase the number of simple instructions and decrease the number of complex instructions.

Window Size	DEP	(25,25,25,25)	(35,35,20,10)	(45,40,10,5)
SWS = 1	0 %	1.717	2.242	2.560
	20%	1.696	2.200	2.541
	40%	1.654	2.171	2.519
	60%	1.646	2.155	2.511
	80%	1.632	2.143	2.480
SWS = 3	0%	1.735	2.292	2.603
	20%	1.721	2.287	2.575
	40%	1.702	2.220	2.563
	60%	1.674	2.203	2.535
	80%	1.663	2.188	2.528
SWS = 6	0%	1.753	2.293	2.610
	20%	1.744	2.296	2.602
	40%	1.729	2.289	2.581
	60%	1.718	2.249	2.582
	80%	1.695	2.217	2.556

Table 1 The list of performance under different issues: instruction mix, dependency ratios and search window sizes in M1. Each performance value is the average number of instructions being able to be translated per cycle.

Table 2 compares the translating performance in different instruction mix (I4), different search window sizes (I5) and different instruction dependencies (I3) in model M2. The results in Table 2 are used to compare with the results in Table 1 in evaluating the effects of translator types (1S + 1G + 1C vs. 2S + 1C) (I2) under the considerations of issues I3, I4 and I5. Note that models M1 and M2 have the same scheduler, but they have different translator types.

Window Size	DEP	(25,25, 25,15,10)	(35,35, 20,5,5)	(45,40, 5,5,5)	(55,15, 15,10,5)
SWS = 1	0 %	1.349	1.564	1.822	2.110
	20%	1.337	1.553	1.820	2.088
	40%	1.336	1.539	1.814	2.057
	60%	1.336	1.535	1.801	2.031
	80%	1.328	1.527	1.804	2.030
SWS = 3	0%	1.348	1.560	1.840	2.150
	20%	1.354	1.570	1.845	2.145
	40%	1.349	1.566	1.826	2.113
	60%	1.347	1.547	1.834	2.088
	80%	1.341	1.552	1.817	2.067
SWS = 6	0%	1.355	1.571	1.841	2.157
	20%	1.354	1.570	1.853	2.145
	40%	1.348	1.572	1.844	2.144
	60%	1.353	1.570	1.858	2.132
	80%	1.346	1.564	1.833	2.092

Table 2 The list of performance under different issues: instruction mix, dependency ratios and search window sizes in M2. Each performance value is the average number of instructions being able to be translated per cycle.

Figure 6 to Figure 8 show the performance comparison of models M1 and M2 under different issues according to the values in Table 1 and Table 2.

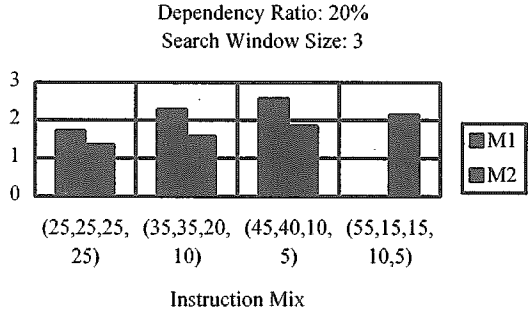


Figure 6 The performance comparison for models M1 and M2 in different instruction mix Obviously, The translating performance in M1 is better than that in M2 for every case of instruction mix. Note that the performance results in M1 for the instruction mix (55,15,15,10,5) are not tested.

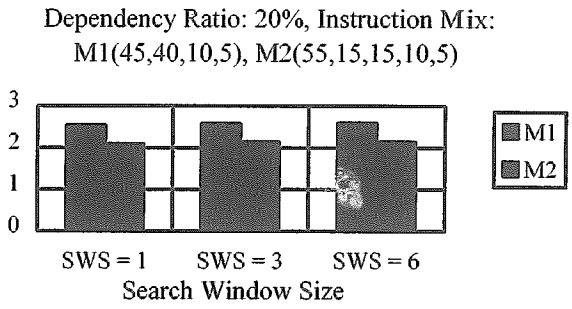


Figure 7 The performance comparison for model M1 and M2 in different search window sizes. Model M1 has better performance than M2 in every case of search window sizes.

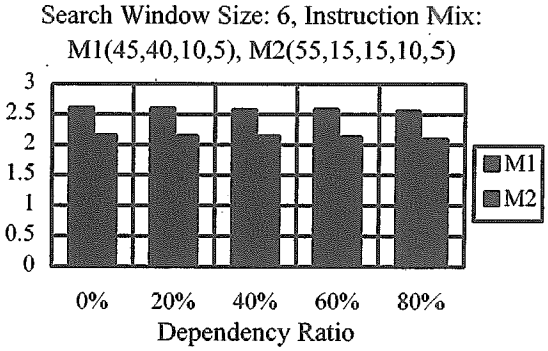


Figure 8 The performance comparison for model M1 and M2 in different dependency ratios. Note that in all cases of dependency ratios, model M1 has better performance than M2.

Table 3 compares the translating performance in different instruction mix (I4) in model M3. Since the model M3 has no scheduler, issues I3 (the search window size) and I5 (the dependency ratio) which relate to the design of the scheduler do not affect the performance results. The results in Table 3 are used to compared with the results in Table 1 in evaluating the effects of the issue of with or without scheduler (I1) under the same translator types (1S + 1G + 1C) (I2).

(25,25,25,25)	(35,35,20,10)	(45,40,10,5)
1.563	2.082	2.428

Table 3 The list of performance for the three cases of instruction mix in model M3. The values represents the average number of instructions being able to be translated per cycle.

Figure 9 and Figure 10 show part of the comparison results. In these figures, we compare M3's performance with M1's in the consideration of

search window sizes and dependency ratios for M1 because model M1 involves a scheduler but model M3 does not.

4.1 Performance Summary

First, Table 1 reveals messages that the translating performance is improved if we widen the search window size, and the programs have the characteristics of low number of instruction dependencies and have more simple instructions. In Table 1, we can find that the best performance is appeared in the case of the search window size of being six, the instruction dependency ratio of being 0%, and the instruction mix of being (45,40,10,5). In this case, the performance is about 2.61 which means that the decoder architecture can decode about 2.6 CISC instructions per cycle in average. On the other hand, the performance improvement of increasing the search window size from one to three, from three to six and from one to six is 2.2%, 1.5% and 3.2% respectively. Similarly, the performance improvement caused by changing the mixture of instructions from (25, 25, 25, 25) to (35, 35, 20, 10) and from (25, 25, 25, 25) to (45, 40, 10, 5) is about 31.4% and about 49.5% respectively. Besides, the performance improvement resulted from decreasing the instruction dependency ratio from 80% to 20% and from 80% to 0% is about 3.0% and 4.3% respectively.

The results of Table 2 also supports the conclusion drew in the above paragraph. However, Table 2 is used to compare with the results in the Table 1. The comparison between them explains what kind of translator types, 1S+1G+1C or 2S+1C, has better performance. Figures 6, 7 and 8 show part of the comparison results. From these figures, we find that the model M1 always has better performance than model M2 no matte which issue is considered. In summary, M1 has more than 20% better translating performance than M2.

Table 3 is the results of evaluating the model M3. The results are also used to compare with the results in Table 1 for comparing the effects of with or without scheduler for performance. Figure 9 and Figure 10 show part of the comparison results. In summary, model M1 can decode about 8.0% more instructions than model M3.

Table 4 lists the factors or issues that affects the translating performance. These values are calculated by fixing the instruction-mix types and the dependency ratios that relate to the program characteristics from Table 1,Table 2 and Table 3. The overall effects of the three factors is 37.9%. In other words, our new model M1 can improve performance by near 38% better than other current CISC/RISC hybrid microprocessors.

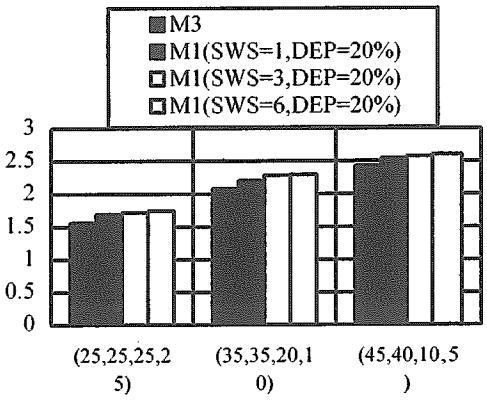


Figure 9 The performance comparison for model M3 and M1 in different instruction mix and search window size under fixed dependency ratios. From this figure, M1 has better performance than M3. In fact, we can compare the performance of M3 with that of M1 in other cases of dependency ratios, for example, 0%, 40%, 60% or 80%.

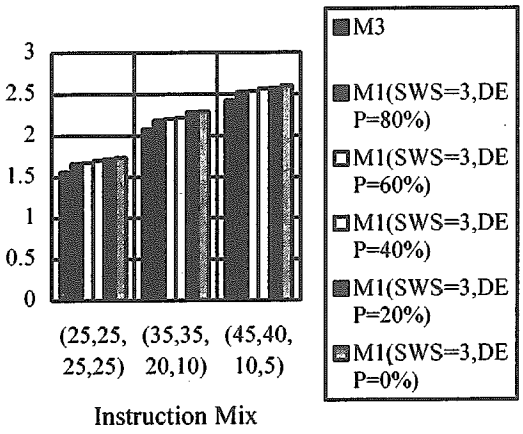


Figure 10 The performance comparison of model M3 and M1 in different instruction mix and dependency ratios under fixed search window size (3). From this figure, M1 has better performance than M3. On the other hand, we can compare the performance of M3 with that of M1 in the case of search window size of 1 or 6.

Factors	Effects
Translator Type (1S+1G+1C vs. 2S+1C)	26.5%
Scheduler (with vs. without)	8.2%
Search Window Size (6 vs. 1)	3.2%
Total	37.9%

Table 4 The list of the factors and their effects that affects the translating performance according to the values in Table 1 to Table 3.

Finally, we compare the translating performance of the decoder architecture, model M1, with that of other current x86/RISC hybrid microprocessors' decoder architectures [8]. Table 5 shows that model M1 has better decoding performance (2.56) than others but the K5 and Nx686 have better translator usage. The translator usage is got by dividing the number of instructions physically being translated by the maximal number of instructions being able to be translated per cycle.

	Model M1	Nx686	P6	K5
instr/cycle	2.56	1.9	2.1	1.9
Max instr/cycle	3	2	3	2-3
translator usage	85.3%	95.0%	70.0%	100%

Table 5 The translating performance comparisons for model M1, Nx686, P6 and K5. The value of the field inst/cycle for model M1 is under the conditions: instruction mix is (45,40,10,5), search window size is 3 and the dependency ratio is 40%. On the other hand, the values of the field inst/cycle for Nx686, P6 and K5 are for the 32-bit codes.

5 Conclusion

This paper proposes a method to improve the translating performance for the CISC/RISC hybrid microprocessors. In this architecture, there is predecoding information describing the number of microinstructions needed to implement the CISC instructions like the K5 [11]. Using this information we can avoid the uncessary translating delay in the P6 due to the instruction hand-off from a simple translator to a complex one in the instruction queue [10]. These predecode bits, however, increase the size of instruction cache array by about 30%(the data cache array is increased by 3/8, but tags and prediction bits are not increased). It does not increase seriously the instruction cache size compared to the K5 that increases the instruction cache size by about 50%.

Besides, the evaluation results show that the decoder architecture with the combination of the translator type (1S+1G+1C) and the scheduler has better translating performance. The results also exhibit that if programs have more simple instructions, there will be much better translating performance. The effects of other issues like dependency ratio and search window size is relatively small.

References

[1] Sebastian Rupley, and John Clyman, "P6: The Next Step", PC Magazine, September 12, 1995, pp.102-118.

[2] The Complete x86: The Definitive Guide to 386, 486, and Pentium-Class Microprocessors, vol. 1, Micro Design Resources, CA, 1994.

[3] Linley Gwennap, "NexGen Enters Market with 66-Mhz Nx586: First Pentium Competitor Uses RISC-like Core and Optional FPU", Microprocessor Report, March 28, 1994, pp. 12-17.

[4] Leland L. Beck, *System Software: In Introduction To System Programming*, 2nd ed., Addison-Wesley Publishing Co., Reading, Mass., 1990.

[5] Brian Case, "x86 Has Plenty of Performance Headroom: Aggressive Superscalar Techniques Just Beginning to Appear", Microprocessor Report, August 22, 1994, pp. 9-14.

[6] Michael Thorne, *Programming the 8086/8088 for the IBM PC and Compatibles*, Benjamin/Cummings Publishing Company, 1986.

[7] Pentium Processor User's Manual Volume 3: Architecture and Programming Manual, Intel, 1993.

[8] Linley Gwennap, "Nx686 Goes Toe-to-Toe with Pentium Pro: NexGen Rolls Out First Competitor to Intel's High-End Chip", Microprocessor Report, Vol. 9, No. 14, October 23, 1995, pp. 1-10.

[9] A. Silberschatz, J. Peterson and P. Galvin, *Operating System Concepts*, 3rd ed. Addison-Wesley Publishing Co., Reading, Mass., 1991.

[10] Linley Gwennap, "Intel's P6 Uses Decoupled Superscalar Design: Next Generation of x86 Intergrates L2 Cache in Package with CPU", Microprocessor Report, February 16, 1995, pp. 9-15.

[11] Michael Slater, "AMD's K5 Designed to Outrum Pentium: Four-Issues Out-of-Order Processor Is First Member of K86 Family", Microprocessor Report, Vol. 8, No. 14, October 24, 1994, pp. 1-11.