The Fetch Mechanism Issue Of X86 Superscalar Processors with Fetch Rules

Jih-ching Chiu and Chung-Ping Chung

Department of Computer Science and Information Engineering,
National Chiao Tung University
Hsinchu, Taiwan 30050, R.O.C.
E-mail: chiujihc@ee.nsysu.edu.tw
Tel: +886-7-5252000ext4142 Fax: +886-7-5254199

Abstract

Fetching multiple instructions is the most important job of the superscalar fetcher. However, in x86 superscalar processors, the variable-length instructions and the complex addressing system make fetching multiple instructions in a cycle difficult. The formats of the x86 instruction are issued in the complexity for parallel fetch of multiple instructions. The model of multiple x86 instruction fetch (MIFM86) is defined with the fetch rules instead of issue rules. By this model, the performances of current x86 processors, affected by each fetch methods, are analyzed and are compared.

Index Terms –ILP, superscalar processor, fetch rule, x86 architecture, multiple instruction fetch

1. Introduction

Due to the success of the IBM PC, the x86 architecture has been dominating in microprocessor market over the past years. Intel Inc. introduced a 16-bit microprocessor, the 8086, which is the beginner of the x86 architecture, in 1978. In new generations of the x86 architectures, such as Pentium, Pentium Pro, Pentium II, K5, K6, M1 and M2, superscalar instruction issue is used to achieve higher performance. The idea of superscalar instruction issue was first formulated as early as in 1970. The concepts of the parallel instruction issue, realized in early 1990s, favors the RISC architecture for its fixed instruction length, simple addressing mode, and simple operation in an instruction [1]. While for the x86 architecture, which is the one of CISC architectures, it is mush more challenging if one tries to speed it up with a superscalar processor design.

Because of the recent advances in silicon technology, parallel execution architecture with higher instruction level parallelism can be implemented on a chip. While increasing the execution parallelism, designers have soon reached the point where performance is limited by the system's capability of finding independent instructions, fetching them from code memory, and feeding them to multiple pipelined execution units operating in parallel. Parallel instruction fetch and issue have become the performance bottleneck, particularly in the x86 architecture.

All of today's high-performance microprocessors are superscalar [2]. While a higher issue rate is often used to acquire higher processor performance, at the same time it complicates the control and data dependencies on the processor performance. Designers may opt to eliminate dependencies during instruction issue by using register renaming and speculative branch processing. The hardware complexity for fetching and decoding instructions in a limited time delay is challenging the x86 microprocessor designers for the high-speed circuit system. The variable lengths of the x86 instructions make hardware design for the instruction fetch unit to identify each instruction in a multiple instruction fetch a difficult task. Complex operations of x86 instructions complicate the design of the parallel code translation. For simplifying the hardware complexity, some instruction coupling rules is taken by today's x86 processor. We call them as fetch rules. To study the effect of the fetch rule, we ignore the execution unit performance, and set it as a full out-of-order environment and a resource-free unit. And this will evaluate an upper-bound performance of a processor.

The fetching model proposed by Wallace and Bagherzadeh [14] deals with the RISC architecture, whose instruction has fixed length, and need not fetch rule for them. In our model, we consider today's x86 processors to construct a triple-field structure, the x86 instruction fetch degree, the coupling rule, and the ROP degree, to describe the fetch rule. The x86 instruction degree is the superscalar degree of an x86 processor that is the instruction quantity, which can be scanned parallel. The coupling rule decides what instruction serial can be parallel fetched into processor. The ROP degree describes how many RISC-like instructions can be issued maximally per cycle. These RISC-like instructions are transferred from the fetched x86 instructions. For simple issued pipe architectures like Pentium, M1, and M2, that do not transfer the x86 instructions to RISC-like operations, we tag <None>. Because we concern the performance affected by the fetch rule only, all processors have the smooth instruction stream input buffers, by assumptions, and the resource-free and out-of-order execution environments in our simulation program.

The remaining parts of this paper are organized as follows. In section 2, we study the instruction fetch mechanisms of several currently x86 microprocessors as background. In section 3, we propose our multiple-x86-instruction fetch rule model. In section 4, we show the simulation results and give some analyses. Finally, we give the conclusions in section 5.

2. Background

For analyzing how much performance can be gained due to various numbers of instructions fetched per cycle, the x86 instruction formats are studied [3]. And the model, focusing on each processor fetch mechanism, is built to describe today's x86 superscalar microprocessors on their fetcher designs. In this model, we propose to use the *fetch rule* that is to decide what the instruction series can be fetched into the processor per cycle, instead of the issue rule commonly seen in the traditional performance analysis of the superscalar microprocessors, to analyze the performance limited by the fetch mechanism. The analysis results can be looked as processor's maximum performance.

2.1 The x86 instruction characteristics

The x86 instruction lengths are variable and range from one to fifteen bytes. The format of x86 instructions can be composed of the prefixes, Op-code, ModR/M byte, SIB byte, displacement, and immediate data, as illustrated in Figure 1 [3].

The prefixes are classified to four types; the instruction prefixes, address size prefixes, operand size prefixes, and segment override prefixes. Each prefix is one byte. Because the four types of prefixes can exist or not independently, the size of the prefix field ranges from zero to four bytes in one regular x86 instruction. In a general case, if the prefix bytes appear repeatedly in an instruction, the maximum size of the prefix field can be fourteen bytes in a legal instruction. The length of the Op-code field is one or two bytes. The operands of an Op-code can be two registers, one register and one memory, or immediate data. In complex addressing modes, the ModR/M field may imply a SIB field in the 32-bit address format of the x86 instruction. Each of the ModR/M field and the SIB field is one byte. If the displacement field exists, its length can be one, two, or four bytes. Like the displacement field, if the immediate field exists, its length also can be one, two, or four bytes.

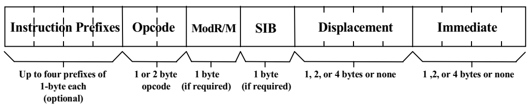


Fig. 1 The format of x86 instructions

Since the x86 instruction format is extremely complex in determining its length, the circuit to determine the instruction boundaries is a speed critical path in an x86 microprocessor. Designing a unit to fetch multiple instructions is a challenging task.

2.2 Instruction-decoded complexity

For the complex instruction format, the general decoder of the x86 instruction is more complex than the RISC's. The designers of today's x86 superscalar architectures divide the x86 instructions to two groups, the complex instructions and the simple instructions. The classifying criteria are different in each microprocessor. The general decoder can decode all x86 instructions. The simple decoder only can decode the simple instructions. By implement results, the hardware complexity rate of the general decoder and the simple decoder is 3:1 by the Pentium Pro classifying criterion [15].

In the SPEC95int trace, we analyze how many simple instructions are executed. Figure 2 illustrates the analysis results. Most of instructions used in SPEC95int applications are simple, above 70%. As a result, the decoder structure will become simpler in order to reduce the hardware complexity.

	PERL	GO	CC1	IJPEG	LI	COMPRESS	M88KSIM	VORTEX	AVG.
ROP#[1]	69.22	80.94	74.09	74.01	76.46	84.97	62.23	66.14	73.51
ROP#[2]	21.46	12.58	15.84	18.83	11.24	6.53	30.29	22.34	17.39
ROP#[3]	7.82	6.00	9.06	3.69	9.84	7.88	4.41	9.83	7.32
ROP#[4]	1.50	0.48	1.01	3.47	2.46	0.62	3.07	1.69	1.79

Fig.2 The ROP distribution of x86 instructions in the SPEC95int traces

2.3 Today's x86 superscalar microprocessors

Many of today's x86 superscalar architectures have a multi-instruction fetch unit. We study the five microprocessors --Pentium, M1, K5, K6, and Pentium Pro -- in two factors that are the coupling rule; and the degrees of RISC-like operations for out-of-order execution system or the number of execution pipes for in-order execution system. We illustrate these in Figure 3.

	Pentium	M1(Cyrix 6x86)	K5	K6	Pentium Pro
Degree of x86 instructions	2	2	4	2	3
The fetch rule	2 simple instructions	2 instructions	4 ROPs	Two 2-ROP instructions	One complex instruction and two simple instructions
Issue rate	2 pipes	2 pipes	4 ROPs	4 ROPs	6 ROPs

Fig. 3 The characters of todays x86 superscalararchitecture

The Pentium instruction fetch unit uses a simple, restrictive approach [3]. Its two pipelines do not operate entirely independently: when one stalls, the other must stop as well, so no out-of-order execution is allowed. The Pentium Pro translates x86 instructions into internal micro-operations [4][9]. Three x86 instructions are fetched in each instruction fetch cycle, in the best case, and are translated to five micro-ops. These micro-ops are then passed to a 40-entry reorder buffer and a 20-entry reservation station, in which the out-of-order execution task takes place. The M1 architecture is similar to the Pentium's, using in-order execution in two execution-pipes [5][11]. The K5 and K6 architectures are designed in a manner similar to the Pentium Pro using out-of-order execution. The rule of fetching x86 instructions in the K5 is limited to four RISC-like operations, called ROPs [7][8]. Since most of the x86 instructions can be converted to an ROP in the one-to-one method [6][12], four x86 instructions can be input to the K5 in the best case. The K6 has two x86 instruction input paths and four ROP paths to the schedule buffer. So in the best case, there are two x86 instructions fetched per clock cycle, and four ROP's can be generated.

The coupling rule is an important factor that affects the number of input instructions per clock cycle. The more general the coupling rule is, the more complex the hardware needs to be. With a simple, restrictive coupling rule, poor average numbers of instructions are fetched per cycle. The Pentium has a simple restrictive fetch rule as the following [3]:

(1) Two instructions must be simple instructions.
(2) Two instructions do not exist RAW and WAW register dependence.
(3) No memory displacement and immediate addressing exist in an instruction pair.
(4) If the instruction has the prefix, except the branch instructions, only it can be executed in the U-pipe.

The M1 has two in-order execution pipes like the Pentium. But its fetch rule is not as simple and restrictive as the Pentium's. Most x86 instructions can be executed in each pipe, and are not limited by the paring rule. But there are three barrier conditions as the following:

(1) Branch instructions, floating point instructions and exclusive instructions only can be executed in the X-pipe.
(2) Branch instructions and floating point instructions can be paired with another instruction that can be executed in the Y-pipe.
(3) Exclusive instructions cannot be paired with each instruction.

The exclusive instructions include the protected mode segment load operations, the special register access operations, the string operations, the push-all and pop-all operations, and inter segment jump, call, and return operations. [5]

The limitation of the K5 fetch rule is looser than the limitation of other microprocessors. If the x86 instruction is not the one requiring more than three ROPs translated, four ROPs can be fetched into the out-of-order execution environment per clock cycle. In the maximum case, four x86 instructions can be fetched in the K5. [7]

Two x86 instructions can be fetched into the K6 per clock cycle and are translated to four RISC86 operations in the maximum case. Each x86 instruction can be translated to one or two RISC86 operations in a pair. If one of the two sequential instructions must be translated to more than two RISC86 operations, only one instruction can be fetched in that clock cycle. When a control transfer occurs, such as a JMP instruction, the entire instruction buffer is flushed and reloaded. [6]

The Pentium Pro allows one complex and two simple instructions to be fetched in a clock cycle. The complex instruction is translated into between one and four micro-ops. The simple instruction only is translated into single micro-op. If none of the x86 instructions are branches or that none of the branches are predicted taken, code fetching will continue along some sequential memory path. Otherwise, the prefetch streaming buffer and the ID queue are flushed. The flushing of the instructions causes a "bubble" in the pipeline, resulting in a temporary decrease in performance. [4]

3. The multiple x86 instruction fetch model

The instruction fetch unit is used to fetch instructions from the instruction memory into the processor, and makes those instructions to the internal structure used for the decoder, as illustrated in Figure4. There are two problems explored in designing the fetch unit of the multiple x86 instructions. One is how to determine the instruction boundaries from the x86 instruction sequence for fetching multiple instructions in the same time, and another is how to assign the multiple instructions to their accurate paths to the decoder.

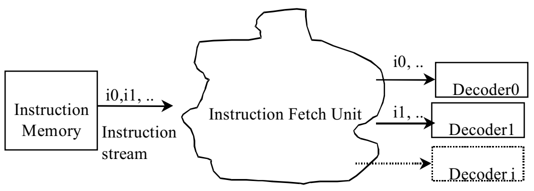


Fig. 4 The role of the fetch unit

3.1 The simple architecture of the fetch unit

To focus our mind on the problems of fetching multiple instructions, we expose the model of the fetching multiple x86 instruction (MIFM86), whose architecture is illustrated in Figure 5.

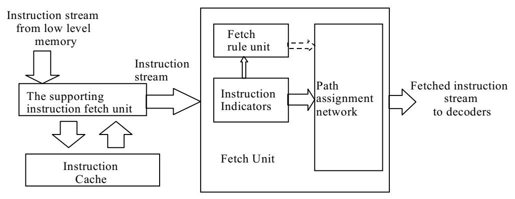

Instruction stream from low level memory
Instruction stream
The supporting instruction fetch unit
Instruction Cache
Fetch rule unit
Instruction Indicators
Path assignment network
Fetch Unit
Fetched instruction stream to decoders

Fig. 5 The architecture of the model of the fetching multiple x86 instruction (MIFM86)

The supporting instruction fetch unit is used to keep the steady flow of the input instruction stream to support the fetch unit speedily to get the multiple instructions in the same time. Like the Pentium and Pentium Pro, the supporting instruction fetch unit is constructed with the special buffer structure to keep the input instruction stream. Like the K5 and K6, this unit can be designed with the instruction pre-decoded function to supply the pre-decoded information of an instruction, such as the instruction type, the instruction boundary, and the number of ROPs, to speed up the fetch rule decision path. To focus the problem in the instruction fetch model, we assume that the unit can support smooth instruction stream for instruction fetch.

The indicating instruction unit is constructed with the instruction input-paths, called the instruction indicator. Each of instruction indicators supports an instruction entry. In the x86 superscalar microprocessors, there are more than one instruction indicators used to determine the starting location of the instructions in the instruction sequence. The instruction indicators take instructions into the fetch unit. For the variable length characteristics of the x86 instruction, to divide the multiple instructions the designers must use the more complex circuit. The circuit will be a critical path because the task to separate each x86 instructions from the instruction sequence is hardly to be a parallel work. So, to support higher degree of instruction entries it is not so easy to be achieved. Some current designs, such as the K5 and K6, use the pre-decoder to supply the information of the instruction boundaries to favor accomplishing this work. We use the notation, *nI*, as that the number of instruction indicators is *n*.

When multiple instruction entries are supported, the fetch rule is used to handle how many instructions are selected to prepare the decoding. The fetch rule unit is designed according the fetch rule. It can use the permutation information of those input instructions to make a group of control signals. Those signals control the path assignment network to assign and duplicate instructions to the accurate paths. Each of the paths connects the general or the simple decoder. The general decoder can translate all types of the x86 instruction to the code structures of the internal form or the internal operations. The symbol *iC*, denotes that the number of the paths used to connect the general decoder is *i*. The simple decoder is limited to translate the simple x86 instruction to the one of the internal operations of .the simple x86 instruction. The simple instruction can be composed of some limited internal operations, such as one or two operations, defined by designers. We use the notation, *jSk*, as the number of the paths of the simple instructions where *k* is the limitation of the simple instructions and *j* is the number of paths. Through the path, a simple instruction will be duplicated to *k* instructions, and each of the duplicated instructions is sent to a simple decoder. If the number of the internal operations of the simple instruction is less than *k*, the *k* paths are still to be assigned to the simple instruction. For the limitations of the fetch rule, the number of the duplicated instructions generated in a multiple instruction fetch is limited. We use the *ROP* as the unit to describe how many duplicated instructions can be passed to the decoder in the same time. The symbol *mROP* is that *m* duplicated instructions are generated in a multiple instruction fetch, in the maximum case.

3.2 The MIFM86 model

In those notations as above, we can use the following structure to describe the fetch mechanism, called the MIFM86 model:

(nI :<iCk , jSl, ...> | <...> : mROP).

The *nI* field means at most *n* x86 instructions can be fetched each clock cycle, and the *mROP* field means there are *m* simple decoding paths in the decoders (i.e. at most *m ROPs* can be generated) each clock cycle. The <*iCk , jSl, …*> list presents the permutation of the each type fetched instructions, which is according to the restrictions of the decoders. The *iCk* term means that *i* complex instructions, each may be mapped up to *k* ROPs, can be decoded in a clock cycle. The *k* may be ignored in dedicated complex decoder architecture. Similarly, *jSl* term means that *j* simple instructions, each may be mapped up to *l* ROPs, can be decoded in each clock cycle. The "| "operator stands for the *OR* operation. The *mROP* field is described as *m* ROP can be decoded in a cycle in maximum. For a microprocessor without the instruction translation, its *mROP* field is described as "None" and the parameters, *k* and *l*, are absent. Those microprocessors have the directed instruction pipes to fetch and execute instructions.

3.3 Modeling Today's x86 Microprocessors

The five today's x86 microprocessors are selected as examples. They are Pentium, M1, K5, K6, and Pentium Pro. The Pentium is described as

(2I: 1C1, 1S1 :<None>).

In this case, this model can be used to describe the fetch rule, and cannot be used to present the execution environments. If the complex instruction path is used to pass the instruction occupying one duplicated instruction paths, the simple instruction can be fetched to the simple instruction path. Otherwise, only one complex instruction can be fetched.

The M1 is described as

(2I: <1C, 1S> |<1S,1C>: <None>).

The M1 series have the out-of-order execution capability. This is accomplished by the switching logic in the D2 stage. This further loosen the instruction pairing restrictions because in a cycle, even if the second instruction is a X-pipe only instruction, it may be switched with the first instruction, which can be executed in the Y-pipe.

The fetch rule of the AMD K5 is described as

(4I:<4C3>:4ROP).

It means that up to four x86 instructions may be passed to the decoders each clock cycle, and up to four ROPs can be generated each clock cycle. Because of the predecode mechanism, the AMD K5 has virtually no pairing rules like the Pentium series and the Cyrix M1 series. The predecode information, which is generated when the instruction line is loaded into the cache is used as the instruction indicators, and the length of instructions can be decided easily by checking the predecode boundary bits. This greatly improves the ability of superscalar fetching of the AMD K5 since multiple instructions can be easily identified. The instruction pairing is only limited by the ability of the ROP Converters (i.e. decoders). Each the four ROP Converter can generate one ROP per cycle. The ROP Converters can decode any instruction that is mapped to less than or equal to 3 ROPs. Complex instructions that are mapped to more than 3 ROPs are decoded by the MROM.

The K6 is described as

(2I:<1C4> | <2S2>:4ROP).

In this case, if the complex instruction path uses less than two duplicated instruction paths, the simple instruction path can use another two duplicated instruction paths. Otherwise, only a complex instruction can be fetched at this time.

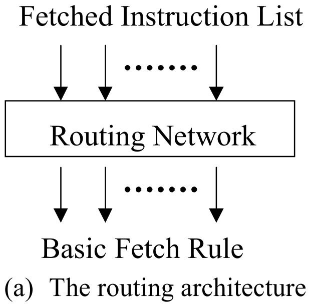
The Pentium Pro is described as

(3I: 1C4, 2S1: 6ROP).

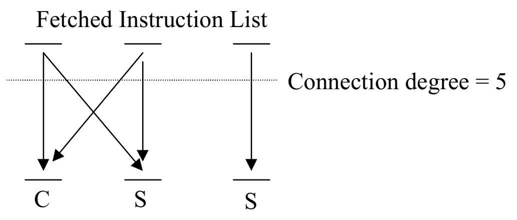
In this case, the first instruction can be a complex or simple instruction with three duplicated paths. If the following instructions are not the simple-instruction with one internal operation, the one or two instructions will be aborted to pass to decoder, and wait another fetch in next cycle.

3.4 The Complexities of the Path Assignment Network

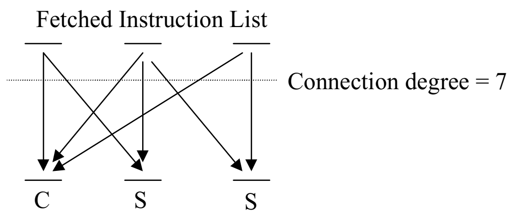
The decoders can be designed as the permutation of the basic fetch rule. In the directed pipe architecture, there are the instruction routing networks to pass the complex instructions to the complex instruction decoders, as the figure 6.a. The hardware complexity is depended on the number of the OR operations of the MIFM86. In the same basic fetch rule as Figure 6.b and Figure 6.c, the connection degree is 5 in two OR operation case and is 7 in three OR operation case.



(a) The routing architecture



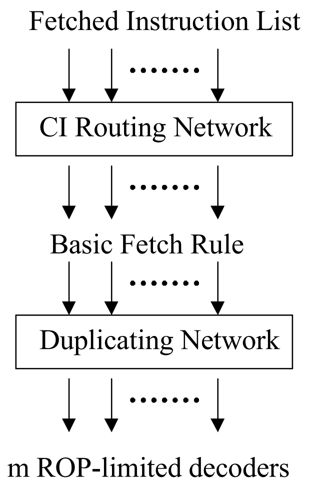
(b) The routing network of the MIFM86 (3I:<1C,2S>|<1S,1C,1S>:<None>)



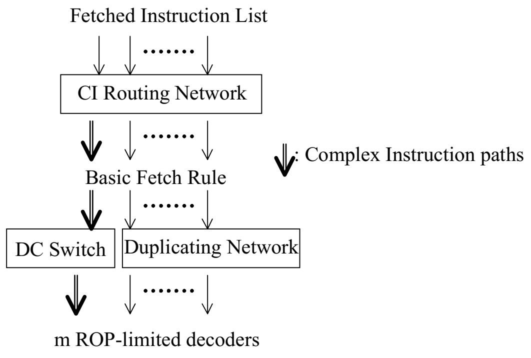
(c) The routing network of the MIFM86 (3I:<1C,2S>|<1S,1C,1S>|<2S,1C>:<None>)

Figure 6. The Path Assignment Network of the directed pipe architecture

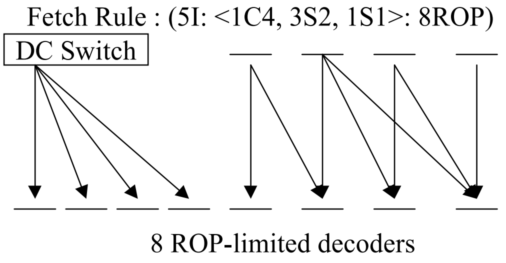
In the instruction-transferred architecture, there are the complex instruction routing network and the duplication network between the fetched instruction list and the basic fetch rule, as Figure 7.a. The CI routing network routes the complex instructions to the complex instruction positions of the basic fetch rule. The hardware complexity of the CI routing network is also depended on the number of the OR operations of the MIFM86. The complex instructions and the simple instructions may be duplicated to the one or many simple decoders by the duplicating network. The complex instructions may be assigned to the dedicated decoder paths, described as the DC switch, in Figure 7.b. The hardware complexity of the duplicating network is depended on the included cases of the basic fetch rule of the MIFM86. Figure 7.c show the duplicating network complexity with the DC switch of the fetched rule (5I: <1C4, 3S2, 1S1>: 8ROP). Fig 7.d show the include case of this duplicating network.



(a) The routing architecture



(b) The routing architecture with the DC switch



(c) The duplicating network complexity with the DC switch of the fetched rule (5I:<1C4, 3S2, 1S1>: 8ROP)

Figure7. The Path Assignment Network of the instruction-transferred architecture

4. Simulations and Analysis

The SPECint95 programs are used as the benchmarks to simulate the selected fetch rules. We compiled the SPECint95 programs and got the execution traces with the Linux C compiler in a Pentium-Pro-based PC. In this paper, because we focus in the problems of the fetch rule, some assumptions are made in our trace-driven programs, as following:

(1) The supporting instruction fetch unit can support the steady instruction stream.
(2) The resources of the out-of-order environment are unlimited.
(3) The serializing instruction is fetched separately in a cycle.
(4) When the branch instruction to be predicted as taken is fetched, it is the last instruction of the fetched multiple instructions in this cycle.
(5) One x86 instruction can be decoded into as few as one ROP, or it can be decoded into several ROPs, depending on its complexity.

4.1 Background Simulation Results

First, the traces of the SPECint95 applications is run in no any fetch rule limitations, described as the nature bound fetch rule, to get the unbounded properties. The results are as Figure 8. The rate of ROP/ x86 is 1.33, which shows that most of the x86 instructions are mapped one ROP in our ROP conversion table. The average degree of x86 instructions fetched in a cycle is 5.54, which shows that using six instruction indicators is a moderate design for those applications. The average degree of ROP is 7.35, which shows that the eight duplicated-instruction paths are enough to pass the internal operations to be executed.

DEGREE	PERL	GO	GCC	IJPEG	LI	COMPRESS	M88KSIM	VORTEX	AVG
x86	5.06	8.22	4.43	4.78	4.68	6.51	5.42	5.24	5.54
ROP	6.95	10.16	6.05	6.38	6.39	7.98	7.7	7.15	7.35

Fig. 8 The simulation results of the SPECint95 applications in the nature bound fetch rule.

In order to study the limit of instruction fetch in x86 architecture, we simulated fetching from one to thirty-two instructions per clock cycle. We assumed all the decoders are complex decoders capable of generating up to four ROPs per cycle, described as the nature bound fetch rule with the port limitations. The results are shown in Figure 9.

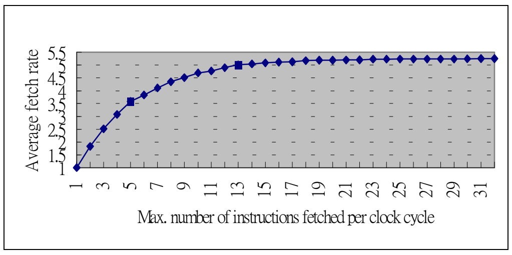


Figure9. The simulation results of the nature bound fetch rule with the fetch port limitations.

From Figure 9, we found that when the maximum number of instructions fetched per cycle increases, the average instruction fetch rate eventually reaches a saturation point. Therefore, fetching too many instructions is not necessary. This result is due to the limitation of a basic block instruction length. In Figure 9 there are two points worth noting. The first is the fetch ports n=5 and the second is n=13. The average fetch rate increases significantly when n < 5. The average fetch rate of n=13 is approach to 5, and it increases very slowly thereafter.

4.2 Simulation Results of Today's X86 Processors

Following the section 3 descriptions, we use the SPECint95 traces to simulate the fetch unit to get the performance of the multiple instruction fetch. The factors of the instruction execution environment are not considered. They are assumed as the very large resource capacity and the out-of-order execution environment to simplify the fetch rule discussion. In those conditions, the simulation results can be seen as the maximum performance case of the superscalar processor, which are only affected by fetch rules. That can give some suggestions to define the fetch rule, and take the trade-off between the fetch rule and the hardware complexity. The simulation results are described as Figure 10. The K5 has a competitive capability. The fetch rule of the K5 is less restrictive. It only limits the number of internal operations to four ROPs. So, the input degree of fetched instructions is four instructions per cycle, in the best case. Each of the decoders is connected with four duplicated-instruction paths and can translate all x86 instructions. From our point of view in the fetch rule, the fetch rule of the K5 is the best. The result of the Pentium is less satisfactory than the M1's, because the fetch rule of the Pentium's is more restrictive. The result of the K6 is almost equal to the M1's. This shows that we can use the fetch rule of the K6's instead of the M1's, because the fetch rule of the K6 is simpler and more restrictive than the M1's. There may be lower hardware complexity in the K6. The fetch rule of the Pentium Pro is like the M1's, but one more simple instruction path limited in one internal operation is added. The simulation results of the Pentium Pro show that such addition is effective in increasing the fetch degree of the multiple instructions.

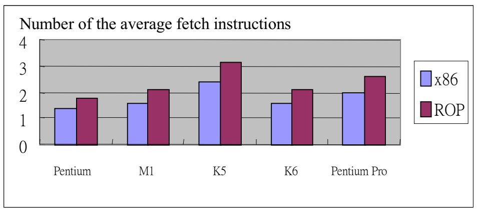


Fig. 10. The simulation results of the fetch rules of the five today's x86 microprocessors

5. Conclusion

The x86 architectures have complex instruction formats, which include the variable lengths, the complex operations, and the complex addressing modes. The design of the fetch unit for the multiple-instruction input becomes the critical work to enhance the microprocessor performance. In this paper, the model of the multiple x86 instruction fetch (MIFM86) is defined to study the fetch rules problems of the multiple-instruction fetch. In the MIFM86 architecture, the fetch unit is divided into the instruction indicator, the fetch rule unit, and the path assignment network. By the fetch rule, instead of by the issue rule, the problems of the multiple instruction fetch are discussed. The simulation results of the SPECint95 applications show that a six-instruction indicator with eight duplicated-instruction paths may be a moderate selection to design the x86 fetch unit. The simulation results of the fetch rules for the five current x86 processors show that the more restrictive fetch rule and the simpler hardware expansion can be selected to enhance the performance of the instruction fetch.

[Reference]

[1] Dezso Sima, "Superscalar Instruction Issue," IEEE Micro, September/October 1997, PP.28-39

[2] Michael Slater, "The Microprocessor Today," IEEE Micro, December 1996, PP.32-44

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

[4] Tom Shanley, Pentium Pro Processor System Architecture, Mind Share, INC., 1997.

[5] Cyrix Corporation, "Cyrix 6x86 Processor Abbreviated Data Book Version 1.1

[6] AMD Corporation, AMD-K6 MMX Enhanced Processor Data Sheet, June 1997.

[7] Dave Christic, "Developing The AMD-K5 Architecture," IEEE Micro, April 1996, PP. 16-26.

[8] AMD Corporation, AMD5K86 Processor Technical Reference Manual, March 1996.

[9] Linley Gwennap, "Intel's P6 Uses Decoupled Superscalar Design," Microprocessor Report, Vol. 9, No. 2, February 16, 1995.

[10] SPEC95 Benchmark Suite Release 1.0, 1995.

[11] B. Ryan, "M1 Challenges Pentium," Byte, Jan. 1994, PP.83-87.

[12] Michael Slater, "K6 to Boost AMD's Position in 1997," Microprocessor Report, Vol. 10, No.14, October 28, 1996.

[13] Albert Yu, "The Future of Microprocessors," IEEE Micro, December 1996, PP.46-53.

[14] Steven Wallace and Nader Bagherzadeh, "Modeled and measured instruction fetching performance for superscalar microprocessors," IEEE Tran. On Parallel and Distributed Systems, Vol. 9, No. 6, June 1998, pp. 570-578.

[15] R-Ming Shiu, Jih-Ching Chiu, Shin-Ki Cheng, and Jyh-Jiun Shann, "The Design of the Decoding Unit with High Issue Rate for an X86 Superscalar Microprocessor," *The IEE Proceedings Computers and Digital Techniques*, Vol. 147, Issue 2, March 2000, pp. 99.