Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

Analytical Study of Performance Evaluation for x86 Instructions to Micro-ops Decoder

Rung-Bin Lin and Chi-Ming Tsai
Department of Computer Engineering and Science
Yuan-Ze Institute of Technology
135 Yuan-Tung Road, Nei-Li, Chung-Li, 320, Taiwan, R.O.C.

Abstract

Several of the recent Intel x86/compatible microprocessor designs employ a special decoder which translates the x86 instructions into RISC-like micro-operations. The decoder consists of several translators of different complexity. Each translator can convert the x86 instructions into their corresponding micro-operations. This paper proposes an analytical method that evaluates the performance of a variety of decoder architectures.

1. Introduction

The designing of the next generation Intel x86 microprocessor is burdened with the backward compatibility to its predecessors. The CISC such as the Pentium microprocessor requires its control circuit to decode instructions of different length and to generate complex timing signals to control the flow of data. An attempt to increase the performance by including the state-of-the-art micro-architectures would lead to tremendously complex hardware implementations. On the other hand, the decoding of RISC instructions and generating of timing signals are simpler than the CISC. The designers of a RISC-based machine can afford to include as many performance-enhanced micro-architectures as possible. In the light of this fact, the Intel's P6, AMD's k5 and NexGen's Nx686 processors all employ the same line of thought that translates the x86 instructions into their corresponding micro-operations by a special decoder. Thus, the execution of x86 instructions is fulfilled by performing their corresponding micro-operations. The micro-operations generated by the decoder are named differently by the vendors. Micro-ops will be used throughout in this paper to denote all kinds of micro-operations. The decoder usually consists of several translators that convert the x86 instructions into micro-ops. Those instructions that can not be handled by the translators are passed over to the microinstruction sequencer.

The Intel p6 decoder [1-5] consists of two simple and one complex translators. The simple translator converts an x86 instruction into one micro-op. The complex translator converts the x86 instructions into one to four micro-ops. The instructions that must be translated into more than four micro-ops are translated by the micro-sequencer. The decoder can translate at most three x86 instructions and produces at most six micro-ops per cycle. The NexGen's Nx686 [6] decoder architecture consists of two translators of the same kind. These two translators are able to convert two x86 instructions in parallel if these two instructions respectively can be translated into one or two micro-ops. If either of the two instructions must be converted into more than two micro-ops, only the first instruction is translated. The two translators can generate up to four micro-ops per cycle. The AMD's k5 [7, 8] decoder architecture consists

of four translators of the same kind. The translator in k5 is named ROP. The ROP converters can translate the x86 instructions into one, two, or three micro-ops. Each ROP converter produces a micro-op. The micro-sequencer is used to translate the instructions that must be translated into four or more micro-ops. The four ROP converters together can translate up to four x86 instructions.

The effectiveness of a decoder is measured by the number of x86 instructions translated and the number of micro-ops generated per cycle. In this paper we propose an analytical method to evaluate the performance of the decoder architectures. Section 2 defines an abstract model for the decoder architecture. Section 3 shows how Markov chains are employed to evaluate the decoder's performance. Some decoder architectures are investigated in section 4. The last section draws some conclusions.

2. Abstract Model of Decoder Architectures

First, an abstract model of a more general decoder architecture is derived from the Intel's p6 decoder architecture. The abstract model shown in figure 1 consists of three types of translators and a micro-sequencer. More types can be included. We will use *D(S(I,X), G(J,Y), C(K,Z))* to denote the abstract model of the decoder, where *S*, *G* and *C* are the three types of translators. Since every decoder has one micro-sequencer, the micro-sequencer is not denoted in the abstract model. The numbers of type *S*, *C*, and *G* translators are respectively equal to *I*, *J*, and *K*. The type *S* translator can convert an x86 instruction into one to *X* micro-ops. The type *G* translator can convert an x86 instruction into one to *Y* micro-ops, and the type *C* translator can convert an x86 instruction into one to *Z* micro-ops. The following assumptions are made for the abstract decoder model(fig. 1):

1. The number of x86 instructions sent to the decoder is at most equal to $I+J+K$ per cycle and a translator can translate only one x86 instruction per cycle.
2. The instructions sent to the decoder must be in the order as they are in the cache.
3. Without loss of generality, it is assumed $0 < X < Y < Z$.
4. If an instruction can be translated by an available type *S* translator, it should not be translated by the type *G* or *C* translator; if an instruction can be translated by an available type *G* translator, it should not be translated by the type *C* translator; if an instruction can be

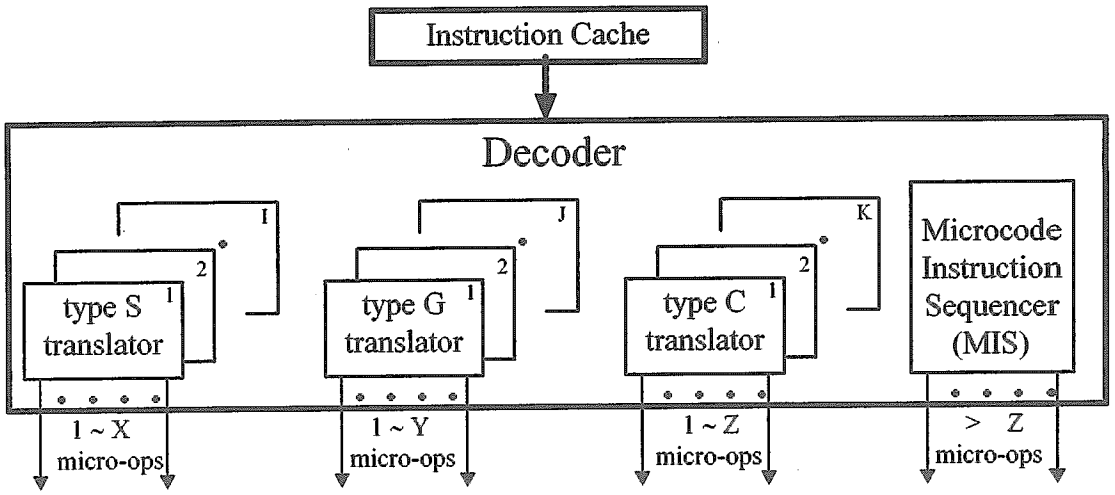

Figure 1. Abstract model of a decoder architecture.

translated by an available type C translator, it should not be translated by the micro-sequencer. If an instruction can not be translated by an available type C translator, it will be passed over to the micro-sequencer. If an instruction can be translated by any of the three type translators, but none of them are available, the instruction will be left untranslated.

5. Let $< Inst_1, Inst_2, ..., Inst_M >$ be an instruction sequence, where $M=I+J+K$. if $Inst_R$, $R \in \{2,...,M\}$, is the first instruction that can not be translated by any available translator. The instructions $Inst_R, Inst_{R+1}, ..., Inst_M$ are not translated. In this case, the number of x86 instructions translated is equal to $R-1$. If R is equal to 1, $Inst_1$ is sent to the micro-sequencer for translation and the number of x86 instructions translated is equal to 1.
6. If $< Inst_1, Inst_2, ..., Inst_M >$ is not the first instruction sequence and the number of instructions translated in the previous instruction sequence is R, $R \in \{1,2,...,M\}$, then the first $M-R$ instructions in the current sequence are just those instructions $Inst_{R+1}, Inst_{R+2}, ..., Inst_M$ of the previously instruction sequence, and the last R instructions are fetched in order from the instruction cache. Otherwise, the instructions of the first instruction sequence are fetched from the cache.

The above assumptions imply some hardware implementation complexity. For example, the assumption 4 is based on the provision of capability of precoding instructions prior to storing the instructions into the cache. The cache should also provide storage capacity to save the precoding information. This hardware implementation may influence the delay on the critical paths. We assume the designing of the machine's micro architecture has already taken this into account to minimize the delay on the critical paths.

Because we are only concerned with whether an x86 instruction can be translated by a translator, a number x will be used to denote an x86 instruction that must be translated into x micro-ops. For example, the instruction sequence $< 3, 2, 2 >$ denotes the x86 instructions will be translated into a sequence of 3 micro-ops, 2 micro-ops, 2 micro-ops. However, if an x86 instruction must be converted by the micro-sequencer, it will be denoted by $max(X, Y, Z) + 1$.

Based on the abstract model, a particular decoder architecture can be constructed by assigning values to I, J, K, X, Y, and Z. For example, if $I=2$, $J=0$, $K=1$, $X=1$, $Y=2$, and $Z= 4$, this particular decoder has two type S translators, no type G translator and one type C translator. The type S translator can only convert an x86 instruction into one micro-op. The type C translator can convert an x86 instruction into one to four micro-ops. Those x86 instructions that must be converted into more than four micro-ops are translated by the micro-sequencer. At most three x86 instructions can be translated and at most six micro-ops are generated per cycle. This particular decoder is equivalent to the Intel's P6 decoder. Since there are only three translators, an instruction sequence is denoted by $< Inst_1, Inst_2, Inst_3 >$; where $Inst_h \in \{1,2,3,4,5\}$, $1 \le h \le M=I+J+K=3$. If $Inst_h \le 4$, the x86 instruction represented by $Inst_h$ should be translated into $Inst_h$ micro-ops. If $Inst_h = 5$, the x86 instruction should be translated into more than four micro-ops.

An $Inst_h$ in $< Inst_1, Inst_2, ..., Inst_M >$ is said to be effectively mapped if it can be translated by an available translator. If the first instruction $Inst_1$ is translated by the micro-sequencer, it is also treated as an effectively

mapped instruction. For each instruction sequence, the decoder should always find out the largest number of effectively mapped instructions.

Example 1: Suppose the Intel's p6 decoder *D(S(2,1), G(0,2), C(1,4))* is considered. According to the above assumptions, the number of effectively mapped instructions for the instruction sequences $< 1, 2, 1 >$, $< 1, 4, 3 >$, $< 3, 5, 3 >$ and $< 5, 2, 4 >$ is respectively equal to *3, 2, 1,* and *1.*

3. Markov Chain Approach to Performance Evaluation

Assume the occurrence of a certain instruction sequence depends on the instruction mix measured in terms of the number of micro-ops to which an x86 instruction should be translated. If N denotes the smallest number of micro-ops that can be generated by the micro-sequencer, the instruction mix is defined as a probability vector $< y_1, y_2, \ldots, y_N >$ such that any given x86 instruction would be translated into r micro-ops with probability y_r for $r < N$. If $r = N$, the given x86 instruction would be translated into at least N micro-ops. Let $S(D) = \{s_1, s_2, \ldots, s_n\}$ be the set of different instruction sequences $< Inst_1, Inst_2, \ldots, Inst_M >$ for a decoder D, where the value of $n = N^M$ is determined by M, the number of translators and N, the number of micro-ops generated by the most complex translator. For the Intel's p6 decoder, $N = 5$, $M = 3$, $n = 125$ and $S(p6) = \{s_1, s_2, \ldots, s_{125}\}$.

Let $x_0, x_1, x_2, \ldots, x_k$ be the instruction sequences sent to the decoder D, where $x_i \in S(D)$ for $0 \le i \le k$ denotes the instruction sequence processed by the decoder at time instant i. Then, the average number of x86 instructions translated by the decoder for the duration of $k+1$ clock cycles is equal to $\frac{\sum_{i=0}^{k} EM(x_i)}{k+1}$, where $EM(x_i)$ is the number of effectively mapped instructions for the instruction sequence x_i. For a long-run process, we are interested in evaluating the expected value of $\lim_{k\to\infty} \frac{\sum_{i=0}^{k} EM(x_i)}{k+1}$, i.e. to compute $E(\lim_{k\to\infty} \frac{\sum_{i=0}^{k} EM(x_i)}{k+1})$.

The instruction sequences $x_0, x_1, x_2, \ldots$, sent to the decoder can be treated as a sequence of $X_0, X_1, X_2, \ldots$ random variables that form a discrete-time Markov chain [9] $MC(V, P)$, where $V=\{v_1, v_2, \ldots, v_n\}$ is the finite set of states. The transitions among states satisfy the Markov property, i.e., $P_{t(k), t(k+1)} = P(X_{k+1}=v_{t(k+1)} \mid X_k=v_{t(k)}, X_{k-1}=v_{t(k-1)}, \ldots, X_1=v_{t(1)}, X_0=v_{t(0)}) = Pr(X_{k+1}=v_{t(k+1)} \mid X_k=v_{t(k)})$, where $v_{t(i)} \in V$ and $t(i)$ is a function that gives the state number at time instant i, $0 \le i \le k+1$. The Markov property implies that the transition from one state to another depends only on the current state and its input. $P_{t(k), t(k+1)}$ is called one step transition probability from state $v_{t(k)}$ to $v_{t(k+1)}$ at time k. If the instruction mix is not changed with time, the one step transition probability will be independent of k. Thus we can write $t(k) = i$, $t(k+1) = j$, and $P_{i,j} = Pr(X_{k+1}=v_j \mid X_k=v_i)$ for all $k \ge 0$. The matrix $P=||P_{i,j}||$ is called state transition probability matrix. According to the assumption 6 made for a decoder, the state transition probability between v_i and v_j is defined as $y_{Inst_{M-R}} y_{Inst_{M-R+1}} \ldots y_{Inst_M}$, where R is the number of effectively mapped instructions for the state v_i. The instructions $inst_{M-R}, inst_{M-R+1}, \ldots, inst_M$, are newly fetched from the cache and serve as the (M-R)-th to the M-th instructions of the state v_j. It is clear that the state v_i corresponds to the instruction sequence s_i and vice versa. Thus, state and instruction sequence

will be used interchangeable unless specified otherwise.

The initial probability vector of the Markov chain can be easily obtained once the instruction mix is given. For a given state $< Inst_1, Inst_2, ..., Inst_M >$, its initial probability is equal to $y_{Inst_1} y_{Inst_2} \cdots y_{Inst_M}$. It can be shown that if none of the y_r in the vector $< y_1, y_2, .., y_N >$ is zero, all of the states in the Markov chain communicate. If some of the y_r's are equal to zero, the instruction sequences that consist of instructions denoted by the number r won't happen at all, i.e. the initial probability of these sequences is equal to zero. Thus, the states of the Markov chain can be classified into two categories. The first category consists of the instruction sequences that do not contain any instruction which would be converted into r micro-ops. The rest of instruction sequences constitute the second category. It can be shown that all the states in the first category form a communicable subclass, while all the states in the second category are transient and no transitions from the states in the first category to these states are possible. Only the states in the first category have non-zero steady state probability. The steady state probabilities of those states in the second category are equal to zero. To find the steady state probability distribution P_{MC} of a time-invariant, discrete state and discrete time Markov chain, one can solve the equation $P_{MC} = P_{MC} \bullet P$, where $\bullet$ denotes the inner product of matrix operation. In fact, P_{MC} is one of the eigenvectors of the transition probability matrix P. Let $P_{MC}(0) = (P_1(0), P_2(0), \ldots, P_n(0))$ denote the initial state probability and the $P_{MC}(k) = (P_1(k), P_2(k), \ldots, P_n(k))$ be the state probability at time k. According to the theories of Markov chain[9], if all the states v_i of a Markov chain communicate and are aperiodic, the Markov chain has a steady state probability distribution $P_{MC} = \lim_{k\to\infty} P_{MC}(k) = (P_1, P_2, \ldots, P_n)$. Then, the average number of x86 instructions translated is equal to $E(\lim_{k\to\infty} \frac{\sum_{i=0}^{k} EM(x_i)}{k+1}) = \sum_{j=1}^{n} EM(v_j) \bullet P_j$, where $EM(x_i)$ and $EM(v_j)$ are respectively the number of effectively mapped instructions for instruction sequence x_i and state v_j of the Markov chain.

The Markov chain can be further simified by bringing the equivalent states into the same class. Any two states are said to be equivalent if they have the same number of effectively mapped instructions, and have the same input and output transition pairs. Two states are said to have the same input and output transitions if their output transitions are all the same for any given input. The states in the same class will be treated as a single state. Thus, the number of states can be reduced considerably and the computation of the steady state probability distribution is easier. The Markov chain with reduced number of states will be called a reduced-state Markov chain. For a reduced-state Markov chain, the initial probability of a certain class is simply the sum of initial probability of the sates in the same class prior to state reduction. The state transition probability of a certain class to the rest of the classes is also the sum of the transition probability of the states in the same class to the rest of the classes prior to state reduction. All the states of the reduced-state Markov chain communicate if all the states of the original Markov chain communicate. It can be shown that if none of the y_r of the vector $< y_1, y_2, .., y_N >$ is zero, all of the states in the reduced-state Markov chain communicate. In case some of the y_r's are equal to zero, this situation is just the same as the original Markov chain. Thus the steady state probability distribution of the reduced-state Markov chain can be computed just the same as the original Markov chain.

The number of x86 instruction translated by a decoder per cycle is one of the important measures of the performance for an x86

compatible microprocessor. Beside this measure, the average number of micro-ops generated per cycle is also an important indicator. The average number of micro-ops generated must be comparable to the number of micro-ops that can be consumed by the execution units. The proposed method can also be employed to compute the average number of micro-ops generated per cycle. What we need to do is to modify the definition of the equivalent states. Two states are said to be equivalent if they generate the same number of micro-ops and have the same input and output transition pairs. Thus, the state reduction can be performed to simify the original Markov chain into a reduced-state Markov chain. The steady state probability distribution of the reduced-state Markov chain can be computed as usual. However, associated with each state is the probability that the number of micro-ops would be generated by the decoder at that state. Let m be the number of states for the reduced-state Markov chain and $OP(v_i)$ be the number of micro-ops generated at state i. Then, the average number of micro-ops generated per cycle is equal to $\sum_{i=1}^{m} OP(v_i) \cdot P_i$, where P_i is the steady state probability of state i.

4. Applications of the Proposed Method

More decoder architectures can be studied by setting the values of *I, J, K, X, Y,* and *Z* for the abstract decoder model *D(S(I,X), G(J,Y),C(K,Z))*. Consider the cases (I). *D(S(2,1), G(0,2), C(1,4))*, (II). *D(S(1,1), G(1,2),C(1,3))*, (III). *D(S(0,1), G(3,2), C(0,3))*, (IV). *D(S(0,1), G(2,2), C(0,3))*, (V). *D(S(1,1), G(0,2), C(2,3))* and (VI). *D(S(0,1), G(1,2), C(2,3))*. Cases (I), (II), and (III) all have three translators and at most can generate *6* micro-ops per cycle. Cases (IV), (V) and (VI) can respectively generate up to *4, 7* and *8* micro-ops per cycle.

Table 1 shows the performance figures obtained by the proposed method. The first column of each case gives the average number of x86 instructions translated per cycle (called measure 1), the second column provides the average number of micro-ops generated per cycle (called measure 2), while the third column gives

Inst. Mix (%)	Case I		
80, 10, 05, 2.5, 2.5	2.656	3.652	3.320
75, 15, 05, 2.5, 2.5	2.562	3.651	3.331
70, 15, 10, 2.5, 2.5	2.447	3.731	3.426
65, 20, 10, 7.5, 2.5	2.316	3.753	3.481
60, 20, 10, 05, 05	2.123	3.714	3.184
55, 15, 15, 10, 05	1.994	3.889	3.390
50, 40, 05, 2.5, 2.5	1.892	3.170	2.933
45, 40, 05, 05, 05	1.741	3.222	2.786
40, 35, 15, 05, 05	1.624	3.248	2.842
35, 35, 20, 05, 05	1.516	3.183	2.804
25, 25, 25, 15, 10	1.321	3.434	2.773
Average	2.017	3.508	3.131

Inst. Mix (%)	Case II		
80, 10, 05, 05	2.600	3.510	2.990
75, 15, 05, 05	2.596	3.634	3.115
70, 15, 10, 05	2.564	3.846	3.330
65, 20, 10, 05	2.555	3.960	3.449
60, 20, 10, 10	2.267	3.854	2.947
55, 15, 15, 15	2.010	3.819	2.613
50, 40, 05, 05	2.518	4.155	3.651
45, 40, 10, 05	2.464	4.313	3.820
40, 35, 15, 10	2.159	4.210	3.346
35, 35, 20, 10	2.085	4.274	3.440
25, 25, 25, 25	1.568	3.919	2.352
Average	2.308	3.954	3.187

Inst. Mix (%)	Case III			Case IV		
80, 10, 10	2.314	3.009	2.314	1.743	2.266	1.743
75, 15, 10	2.314	3.124	2.43	1.743	2.353	1.83
70, 15, 15	2.082	3.018	2.082	1.641	2.739	1.641
65, 20, 15	2.082	3.122	2.186	1.641	2.461	1.723
60, 20, 20	1.895	3.031	1.894	1.552	2.483	1.552
55, 15, 30	1.614	2.824	1.372	1.405	2.459	1.194
50, 40, 10	2.314	3.703	3.009	1.743	2.789	2.266
45, 40, 15	2.082	3.539	2.602	1.641	2.789	2.05
40, 35, 25	1.741	3.221	1.915	1.474	2.726	1.621
35, 35, 30	1.614	3.147	1.695	1.405	2.74	1.475
25, 25, 50	1.273	2.864	0.955	1.200	2.7	0.9
Average	1.939	3.146	2.312	1.563	2.591	1.636

Inst. Mix (%)	Case V			Case VI		
80, 10, 05, 05	2.608	3.521	2.999	2.611	3.524	3.002
75, 15, 05, 05	2.604	3.645	3.125	2.611	3.655	3.133
70, 15, 10, 05	2.600	3.895	3.376	2.610	3.915	3.393
65, 20, 10, 05	2.587	4.009	3.492	2.610	4.045	3.523
60, 20, 10, 10	2.293	3.899	2.981	2.314	3.933	3.008
55, 15, 15, 15	2.063	3.919	2.681	2.079	3.95	2.703

50, 40, 05, 05	2.525	4.166	3.661	2.610	4.307	3.785
45, 40, 10, 05	2.491	4.359	3.861	2.610	4.567	4.045
40, 35, 15, 10	2.210	4.31	3.426	2.312	4.508	3.583
35, 35, 20, 10	2.174	4.457	3.588	2.308	4.732	3.809
25, 25, 25, 25	1.661	4.153	2.492	1.732	4.33	2.598
Average	2.347	4.03	3.244	2.401	4.133	3.326

Table 1. Performance measure of decoders

the average number of micro-ops generated by translators per cycle (called measure 3). Measure 3 does not include the micro-ops generated by the micro-sequencer and is used to evaluate the performance of the translators in a decoder. We define the translating efficiency of the translators as the average number of micro-ops generated by the translators per cycle (i.e., measure 3) divided by the maximal number of micro-ops that would be generated by the translators per cycle. Thus, the translating efficiency for all the cases are respectively 0.522, 0.531, 0.385, 0.409, 0.463 and 0.416.

Since the complexity and hardware cost of each translator are not known to the public, we will assume the amount of hardware used to generate one micro-op is the same for all the translators. Those machines that have the same peak number of micro-ops generated per cycle will be said to have the same hardware design complexity. Thus Cases (I), (II), and (III) have the same hardware complexity. On this basis, Case (II) studied in [10] has the best performance among the three cases in terms of all the measures and the translating efficiency. Note, two different micro-architectures may have different micro-ops. Cares must be taken for performance comparisons between two different micro-architecture.

Case (I) which represents the Intel's p6 decoder also demonstrates excellent performance. Especially, measure 2 is quite stable and all the entries are greater than *3*. This is an excellent match to the *3*-issue capability of the p6 execution units. Measure 3 and the translating efficiency are also quite good. Note that the average number of x86 instructions translated per cycle in Case (I) is monotonically decreased, while it is not true for the other cases.

Case (IV) has two type *G* translators, which is quite similar to the NexGen's Nx686. It is clear that the number of micro-ops generated and the number of x86 instructions translated are only sufficient for a superscalr processor with two issues per cycle.

Case (V) and (VI) demonstrate that for two different decoder architectures D_1 and D_2 if D_2 covers D_1, then the performance of decoder D_2 will not be worse than decoder D_1. Thus decoder D_1 can be excluded from investigation if the minimum decoder performance must be as good as decoder D_2's. Decoder D_2 is said to cover decoder D_1 if every translator of D_1 can be covered by a distinct translator of D_2. A translator T_1 is covered by a translator T_2 if $W(T_1) \le W(T_2)$, where the function W denotes the maximal number of micro-ops that can be generated by a translator.

5. Conclusions

In this paper we proposed an analytical method based on Markov chain to evaluate the performance of the decoder that translates the x86 instructions into their equivalent micro-operations. The average number of x86 instructions translated and the number of the micro-ops generated per cycle can be accurately and effectively measured. In our studies, the performance of decoder architectures for the Intel's p6's and NexGen's Nx686 is quite matched to their superscalar capability.

References:

[1] S. Pupley, and J. Clyman, "P6: The Next Step," PC Magazine, September 12, 1995, pp. 102-118.

[2] T. R. Halfhill, "Intel's P6," BYTE, April 1995, pp. 42-58.

[3] N. Stam, "Inside the P6," PC Magazine, September 12, 1995, pp. 118-137.

[4] L. Gwennap, "Intel's P6 Uses Decoupled Superscalar

Design: Next Generation of x86 Intergrates L2 Cache in Packcge with CPU," Microprocessor Report, February 16, 1995, pp. 9-15.

[5] D. B. Papworth, "Tuning the Pentium Pro Microarchitecture," IEEE Micro, Apr 1996, pp. 8-14.

[6] L. 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.

[7] D. Christie, "Developing the AMD-K5 Architecture," IEEE Micro, Apr 1996, pp. 16-26.

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

[9] W. Feller, "An Introduction to Probability Theory and Its Applications," John Wiley & Sons, Inc., 1968.

[10] W. B. Jian, "A Method of Improving Translating Performance in the CISC/RISC Hybrids," Master Thesis, Department of Computer Science and Information Engineering, National Chiao-Tung University, Taiwan, R.O.C., 1996.