類似 Intel P6 解碼器之研究
On the Study of Intel P6-Like Decoders

蔡志明
Chi-Ming Tsai

元智大學資訊工程系
Department of Computer Engineering and Science.
Yuan-Ze University
s861911@mail86.yzu.edu.tw

林榮彬
Rung-Bin Lin

元智大學資訊工程系
Department of Computer Engineering and Science.
Yuan-Ze University
csrlin@cs.yzu.edu.tw

摘 要

最近所發表的一些 Intel x86 及其相容之微處理機中，均設計一個特別的解碼器(decoder)將 x86 指令轉換成類似精簡指令集電腦之微指令(micro-operation)。這篇論文深入探討以各種解碼方法（decoding scheme）、搜尋視窗之大小（scan window size）及翻譯器(translator)的組合對類似 Intel P6 解碼器效能之影響，同時亦評估解碼器之效能與其晶片面積(die area)之取捨。我們發現翻譯器的組合對解碼器的效能有決定性的影響，而解碼的方法及視窗大小對解碼器的效能之影響是非常有限的。

Abstract

Several of the recent Intel x86/compatible microprocessor designs employ a special decoder that translates the x86 instructions into RISC-like micro-operations. In this paper a variety of P6-like decoders are studied in terms of various decoding schemes, instruction scan window size and their constituents. The trade-off between performance and die area of these decoders is also studied in details. We find that the constituent of a decoder dominantly decides its performance, while the influence of the decoding scheme and window size is quite limited.

1. Introduction

The recently announced x86-compatible microprocessors such as Intel P6 and Pentium II, AMD K5 and K6, and NexGen Nx686 (which has been acquired by the AMD) all employ a special decoder that translates an x86 instruction into some micro-operations before the instruction is executed by functional units. The execution of an x86 instructions is fulfilled by performing its corresponding micro-operations. For example, the x86 instruction *ADD Mem, BX* should be translated into a sequence of three micro-operations: *LOAD R2, R1; ADD R3, R2; STORE R1, R3*, where *Mem* is a memory address held in *R1*. Thus, the execution of *ADD Mem, BX* is carried out by executing the three micro-operations (briefly called micro-ops).

The decoder usually consists of several translators that convert x86 instructions into micro-ops. Those instructions that can not be handled by the decoder are passed over to the microcode instruction sequencer (briefly the micro-sequencer). Figure 1 shows the decoder architecture of the Intel P6's microprocessor [1-8]. Example 1 illustrates how Intel P6's decoder translates the x86 instruction into the micro-operations. The NexGen Nx686's [5] and the AMD K5's [6] decoder architectures employ a technique similar to Intel P6's approach.

Example 1: Suppose $Inst_1$, $Inst_2$, $Inst_3$, $Inst_4$, $Inst_5$, $Inst_6$, $Inst_7$, $Inst_8$, $Inst_9$ and $Inst_{10}$ form a stream of x86 instructions that have to be respectively converted into *2, 4, 2, 1, 1, 1, 5, 1, 1* and *2* micro-ops and initially reside in the instruction cache. Table 1 shows the instructions that are translated by P6's decoder at each clock cycle. It takes 6 clock cycle to completely translate these instructions. The $< Inst_1, Inst_2, Inst_3 >$, $< Inst_2, Inst_3, Inst_4 >$ and the like are the instruction sequences inspected by the decoder. Only one instruction sequence is inspected per clock cycle. The last column gives the instructions inspected by the decoder during each cycle. The entry denoted by $(Inst_i, j)$ under each translator represents $Inst_i$ is translated into j micro-ops by the designated translator. If $(Inst_i, j)$ is in the column *micro-sequencer*, it indicates that $Inst_i$ will be translated by the micro-sequencer into at least j micro-ops. In general, the micro-sequencer will put out a certain number of micro-ops per cycle until an x86 instruction is completely decoded. For simplicity, we assume that all micro-ops will be generated by the micro-sequencer at one cycle. □

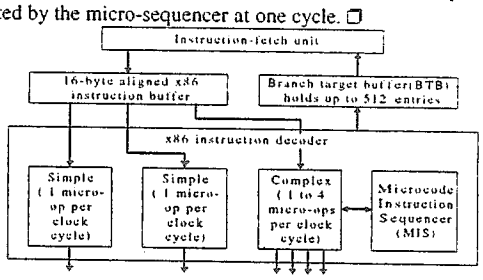

Figure 1. Intel P6's decoder architecture.

Because we are only concerned with whether an x86 instruction can be translated by a translator, the number of its corresponding micro-ops, not the operations of the instruction itself, is of our interest. Therefore, a number *3* will be used in an instruction stream to denote an x86 instruction that must be translated into *3* micro-ops. For example, the instruction stream $<3, 2, 2, \ldots>$ denotes a sequence of x86 instructions that 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 one plus the largest number of micro-ops that can be generated by any translator, i.e., the number of micro-ops generated by the micro-sequencer per cycle.

Table 1. A decoding scenario by P6 decoder

Clock Cycle	Simple Translator	Simple Translator	Complex Translator	Micro Sequencer	Instructions inspected
1			$(Inst_1, 2)$		$< Inst_1, Inst_2, Inst_3 >$
2			$(Inst_2, 4)$		$< Inst_2, Inst_3, Inst_4 >$
3	$(Inst_4, 1)$	$(Inst_5, 1)$	$(Inst_3, 2)$		$< Inst_3, Inst_4, Inst_5 >$
4	$(Inst_6, 1)$				$< Inst_6, Inst_7, Inst_8 >$
5				$(Inst_7, 5)$	$< Inst_7, Inst_8, Inst_9 >$
6	$(Inst_8, 1)$	$(Inst_9, 1)$	$(Inst_{10}, 1)$		$< Inst_8, Inst_9, Inst_{10} >$

The effectiveness of a decoder is measured in terms of the number of x86 instructions translated and the number of micro-ops generated at each clock cycle. In [9] the decoding process of this sort of decoders is modeled as a Markov chain and Markov chain theory is employed to evaluate the performance of a decoder. In [10] the authors study the trade-off between the P6's performance and the die area of the decoders of different constituents. The authors study how dependency between instructions and limited reordering of instructions will influence upon the performance of the decoder. The simulation results demonstrate that the influence of instruction dependency on decoder

performance is quite limited. In [11] simulation is performed to study the performance for several decoders similar to the one used by the P6.

In this paper we will explore more design options that would be possible to enhance the performance of a decoder. The approach proposed in [9] is employed to evaluate the performance of a decoder. The rest of this paper is organized as follows. In Section 2 the proposed works are briefly presented. In Section 3 an abstract model of decoder architectures is introduced [9]. In Section 4 the five decoding schemes are described in detail based on the abstract model. In Section 5 the above four issues are investigated based on a large amount of performance figures. The last section draws some conclusions.

2. Proposed works

In example 2.1 if we inspect the instruction sequence closely and reorder the instruction stream into $Inst_1$, $Inst_4$, $Inst_2$, $Inst_3$, $Inst_5$, $Inst_6$, $Inst_7$, $Inst_8$, $Inst_9$ and $Inst_{10}$, the instructions decoded per clock cycle are tabulated in Table 2.

Table 2. Another decoding scenario of instruction with reordering

Clock Cycle	Simple Translator	Simple Translator	Complex Translator	Micro Sequencer	Instructions inspected
1	($Inst_1$, 1)		($Inst_1$, 2)		< $Inst_1$, $Inst_4$, $Inst_2$>
2			($Inst_2$, 4)		< $Inst_2$, $Inst_3$, $Inst_5$>
3	($Inst_5$, 1)	($Inst_6$, 1)	($Inst_3$, 2)		< $Inst_3$, $Inst_5$, $Inst_6$>
4				($Inst_7$, 5)	< $Inst_7$, $Inst_8$, $Inst_9$>
5	($Inst_8$, 1)	($Inst_9$, 1)	($Inst_{10}$, 1)		< $Inst_8$, $Inst_9$, $Inst_{10}$>

Comparing Table 2 to Table 1, the number of clock cycles spent in this case is one less than that in the previous case. Thus, we guess that the reordering of instructions during decoding will increases the decoder performance in terms of the number of instructions decoded per clock cycle. What "reordering of instructions" means is in fact to bypass some instructions that can't be converted by an available translator during decoding. There are two bypassing strategies: first fit and best fit, which will be discussed in detail in the following paragraphs.

Normally, Intel P6's decoder won't translate the instructions following the one that must be translated by the micro-sequencer. We expect that the number of instructions translated by the decoder will increase if it can translate the instructions behind the one that must be handled by the micro-sequencer. We call this capability "lookahead". Based on whether with or without lookahead, with or without a bypassing strategy, the following five decoding schemes will be studied in this paper.

Decoding scheme 1: Intel P6 decoding scheme.

Decoding scheme 2: Intel P6 decoding scheme without lookahead and first fit.

Decoding scheme 3: Intel P6 decoding scheme without lookahead and best fit.

Decoding scheme 4: Intel P6 decoding scheme with lookahead and first fit.

Decoding scheme 5: Intel P6 decoding scheme with lookahead and best fit.

Moreover, the number of x86 instructions eligibly inspected by the P6's decoder at each clock cycle is equal to the sum of the translators. If the number of instructions inspected can be increased by a number W, we expect the number of instructions decoded per cycle will increase too. We call W the size of a scan window in which the x86 instructions are eligible for translation. Based on the above observations, the following issues with regard to designing of a decoder for higher superscalar capability are investigated in this paper.

(1). The decoder performance in terms of the constituent of a decoder, i.e., the number and the types of translators that form a decoder.
(2). The decoder performance in terms of decoding scheme.
(3). The decoder performance in terms of scan window size.
(4). The relation among decoder performance, die area and the constituents of decoders.

3. Abstract model of a decoder

In order to formally address the performance evaluation problem, the abstract model for the decoder architecture proposed in [9] is extended to model the above five decoding schemes and presence of a scan window. The abstract model shown in Figure 2 consists of three types of translators and a micro-sequencer. The abstract model is denoted by $D(S(I,X), G(J,Y), C(K,Z), H, W)$, where S, G and C are the three types of translators. The numbers of type S, G, and C 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. W gives the size of a scan window. H is a number denoting the decoding scheme that will be adopted. For example, if H = 2, the decoding scheme 2 is used. Based on the abstract model, a particular decoder architecture can be constructed by assigning values to I, J, K, X, Y, Z, H and W.

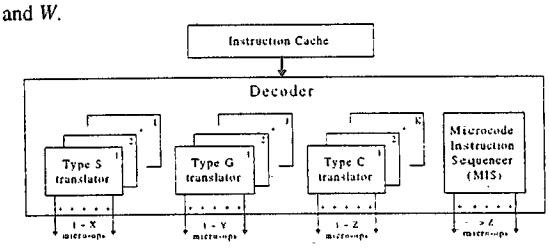

Figure 2. An abstract model of decoder architecture

4. Five decoding schemes

In this section detailed descriptions of the five decoding schemes are presented. Take the decoder $D(S(I,X), G(J,Y), C(K,Z), H, W)$ for instance. First, let $T=\{T_1, T_2, \ldots, T_Q\}$, where $Q=I+J+K$, be the set of translators and $L(T_i)$ denote the maximum number of micro-ops that can be generated by translator T_i. Note that some of these translators may be of the same type. Despite the difference among the five decoding schemes, the following two assumptions are common to all decoding schemes.

(1). Without loss of generality, it is assumed $0 < X < Y < Z$ and $0 < L(T_i) \le L(T_{i+1}) < N$ for all i, where $N = \max_{T_i \in T} L(T_i) + 1$ denotes the number of micro-ops that can be generated by the micro-sequencer.

(2). At each cycle a translator can translate only one x86 instruction. That is, up to Q x86 instructions can be decoded at each cycle. Thus, the number of micro-ops generated by the translators per cycle is up to $\sum_{i=1}^{Q} L(T_i) = I * X + J * Y + K * Z$.

Additionally, let T^* be the set of available translators during the decoding of an instruction sequence and R^* be the set of ($Inst_j$, T_i) or ($Inst_j$, *micro-sequencer*) pairs, where ($Inst_j$, T_i) denotes the translator T_i is assigned to translate $Inst_j$ and ($Inst_j$, *micro-sequencer*) denotes the micro-sequencer is assigned to translate $Inst_j$. And $<Inst_1, Inst_2, \ldots, Inst_M>$ is the instruction stream eligibly inspected by the decoder, where $M=Q+W$.

Decoding scheme 1: The Intel P6 decoding scheme.

Step 1: Let $T^* = T$ and $R^* = \emptyset$.
If $Inst_1$ must be translated by the micro-sequencer.
Then $R^* = R^* \cup \{(Inst_1, micro\text{-}sequencer)\}$ and go to step 3.

Step 2: For (j = 1; ($j < M+1$ and T^* != $\emptyset$); j++) { /* For each instruction */
Find a translator $T_i \in T^*$ such that $\min_{T_i \in T} \left(L(T_i) - Inst_j\right) \ge 0$.
If T_i is found, $R^* = R^* \cup \{(Inst_j, T_i)\}$ and $T^* = T^* - \{T_i\}$.
Otherwise, go to step 3.
}

Step 3: Decode the instructions that have been assigned with translators or the micro-sequencer. Then, form a new instruction sequence by fetching some instructions from the I-cache. Go to step 1.

Decoding scheme 2: Without lookahead and first fit.

Step 1: Let $T^* = T$ and $R^* = \emptyset$.
If $Inst_1$ must be translated by the micro-sequencer.

Then $R^* = R^* \cup \{(Inst_1, micro\text{-}sequencer)\}$ and go to step 3.
Else if $Inst_F$, $F' \in \{2, \ldots, M\}$, is the first instruction that must be translated by the micro-sequencer.
Then $F = F'$.
Else $F = M+1$. /* No instruction that must be translated by the micro-sequencer */

Step 2: For ($j = 1$; ($j < F$ and $T^* \neq \emptyset$); j++) { /* For each instruction */
Find a translator $T_i \in T^*$ such that $\min_{T_i \in T}(L(T_i) - Inst_j) \geq 0$.
If T_i is found, $R^* = R^* \cup \{(Inst_j, T_i)\}$ and $T^* = T^* - \{T_i\}$.
}

Step 3: Decode the instructions that have been assigned with translators or the micro-sequencer. Then, form a new instruction sequence by fetching some instructions from the *I*-cache. Go to step 1.

Decoding scheme 3: Without lookahead and best fit.

Step 1: Let $T^* = T$ and $R^* = \emptyset$.
If $Inst_1$ must be translated by the micro-sequencer.
Then $R^* = R^* \cup \{(Inst_1, micro\text{-}sequencer)\}$ and go to step 3.
Else if $Inst_F$, $F' \in \{2, \ldots, M\}$, is the first instruction that must be translated by the micro-sequencer.
Then $F = F' - 1$.
Else $F = M$. /* No instruction that must be translated by the micro-sequencer */
$I^* = \{Inst_1, Inst_2, \ldots, Inst_F\}$.

Step 2: For ($i = 1$; ($i < Q+1$ and $I^* \neq \emptyset$); i++) { /* For each translator */
Find an instruction $Inst_j \in I^*$ such that $\min_{T_i \in T}(L(T_i) - Inst_j) \geq 0$.
If $Inst_j$ is found, $R^* = R^* \cup \{(Inst_j, T_i)\}$ and $I^* = I^* - \{Inst_j\}$.
}

Step 3: Decode the instructions that have been assigned with translators or the micro sequencer. Then, form a new instruction sequence by fetching some instructions from the *I*-cache. Go to step 1.

Decoding scheme 4: With lookahead and first fit.

Step 1: Let $T^* = T$ and $R^* = \emptyset$.
If $Inst_1$ must be translated by the micro-sequencer.
Then $R^* = R^* \cup \{(Inst_1, micro\text{-}sequencer)\}$ and go to step 3.

Step 2: For ($j = 1$; ($j < M+1$ and $T^* \neq \emptyset$); j++) { /* For each instruction */
Find a translator $T_i \in T^*$ such that $\min_{T_i \in T}(L(T_i) - Inst_j) \geq 0$.
If T_i is found, $R^* = R^* \cup \{(Inst_j, T_i)\}$ and $T^* = T^* - \{T_i\}$.
}

Step 3: Decode the instructions that have been assigned with translators or the micro-sequencer. Then, form a new instruction sequence by fetching some instructions from the *I*-cache. Go to step 1.

Decoding scheme 5: With lookahead and best fit.

Step 1: Let $T^* = T$, $R^* = \emptyset$ and $I^* = \{Inst_1, Inst_2, \ldots, Inst_M\}$.
If $Inst_1$ must be translated by the micro-sequencer.
Then $R^* = R^* \cup \{(Inst_1, micro\text{-}sequencer)\}$ and go to step 3.

Step 2: For ($i = 1$; ($i < Q+1$ and $I^* \neq \emptyset$); i++) { /* For each translator */
Find an instruction $Inst_j \in I^*$ such that $\min_{T_i \in T}(L(T_i) - Inst_j) \geq 0$.
If $Inst_j$ is found, $R^* = R^* \cup \{(Inst_j, T_i)\}$ and $I^* = I^* - \{Inst_j\}$.
}

Step 3: Decode the instructions that have been assigned with translators or the micro-sequencer. Then, form a new instruction sequence by fetching some instructions from the *I*-cache. Go to step 1.

In the last step of the algorithm for each decoding scheme, if the number of translated instructions is k, where $k \in \{1,2,\ldots,M\}$, then the first M-k instructions in the next instruction sequence are just those instructions that are not translated in the current sequence, and the last k instructions are fetched in order from the instruction cache.

5. Experimental results

In order to evaluate the performance of a decoder, an instruction mix in terms of the number of micro-ops generated per x86 instruction should be obtained first. In principal, benchmark programs should be executed to find out the instruction mixes. However, it requires to know exactly how many micro-ops are used to make up an individual x86 instruction. None of this kind of information has been published. Therefore, we simply adopt the set of instruction mixes used in [11] to characterize the typical programs. Table 8 lists the sets of instruction mixes used in our experiments. The instruction mix of a program is denoted as a vector $< y_1, y_2, \ldots y_N >$, where a number y_i, $i \in \{1 \ldots N-1\}$, gives the percentage of x86 instructions that will generate i micro-ops. The last number y_N gives the percentage of x86 instructions that will generate at least N micro-ops. The need of different sets of instruction mix is due to difference in the maximum number of micro-ops generated by the micro-sequencer. Thus, instruction mix is designed in such a way that the percentages of simpler instructions remain the same while the percentages of more complex instructions vary when the maximum number of micro-ops generated by the micro-sequencer changes. In addition, according to numerous published results, most of the instructions executed in an x86 program are simple instructions. This argument is also supported by the fact that Intel P6's decoder consists of two simple translators that convert an x86 instruction into only one micro-op. Thus, the instruction mixes are designed to give higher probabilities to the instructions that are converted into fewer micro-ops.

Table 8. List of instruction mixes.

Instruction mix			
Set 1	Set 2	Set 3	Set 4
80, 10, 05, 2.5, 2.5	80,10,05,05	80,10,10	80,20
75, 15, 05, 2.5, 2.5	75,15,05,05	75,15,10	75,25
70, 15, 10, 2.5, 2.5	70,15,10,05	70,15,15	70,30
65, 20, 10, 2.5, 2.5	65,20,10,05	65,20,15	65,35
60, 20, 10, 05, 05	60,20,10,10	60,20,20	60,40
55, 15, 15, 10, 05	55,15,15,15	55,15,30	55,45
50, 40, 05, 2.5, 2.5	50,40,05,05	50,40,10	50,50
45, 40, 05, 05, 05	45,40,10,05	45,40,15	45,55
40, 35, 15, 05, 05	40,35,15,10	40,35,25	40,60
35, 35, 20, 05, 05	35,35,20,10	35,35,30	35,65
25, 25, 25, 15, 10	25,25,25,25	25,25,50	25,75

In Section 5.1, the performance of various decoders that have been once the candidates for the Intel P6's decoder is evaluated and compared. The purpose is to see if we are the Intel P6 designers, how would the decision be made in choosing a decoder in terms of its performance. In Section 5.2 we will study more decoders in terms of decoding scheme, scan window size, and trade-off between constituent of decoders and die area.

5.1 Performance of Intel P6's decoder

In [10] the following five decoders accompanied by their die area have been once considered to be the alternatives to Intel P6's decoder.
(I). *D(S(1,1), G(0,2), C(1,4), H, W)* whose die area is 75% of (III)'s.
(II). *D(S(0,1), G(1,2), C(1,4), H, W)* whose die area is 87% of (III)'s.
(III). *D(S(2,1), G(0,2), C(1,4), H, W)* whose die area is used as a reference for other decoders.
(IV). *D(S(0,1), G(2,2), C(1,4), H, W)* whose die area is 125% of (III)'s.
(V). *D(S(0,1), G(3,2), C(1,4), H, W)* whose die area is 200% of (III)'s.

It is interesting to see why Intel chooses the decoder specified by (III) as the decoder for P6. Table 9 shows the performance of these decoders (under the decoding scheme 1). The first column gives the instruction mix of a program. For each decoder the first column gives the average number of x86 instructions translated by the decoder per cycle (call *measure 1*); the second column provides the average number of micro-ops generated by the decoder per cycle (call *measure 2*); the third column gives the average number of the micro-ops generated by the translators per cycle (call *measure 3*). Note that *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. The last row represents the average performance of the decoder for each measure.

All of these five decoders have good performance and only (I)'s *measure 2* is less than 3. Although the average of *measure 2* for (II) is greater than 3, the first three entries of *measure 2* are all less than 3. If most of the instruction mixes of the programs would be more like the first three entries, this performance figure is not viable. In terms of *measure 2*, it seems that the decoder specified in (III), (IV) and (V) are

all good candidates for the decoder of Intel P6's processor. To inspect further, Table 10 shows the ratios of decoder performance to the die area for the five decoders. The row named "Performance" denotes the performance measure based on the decoding scheme 1 and the row named "Ratio" denotes the ratios of decoder performance to die area. Apparently, the decoder specified by (IV) has the best ratio. However, Intel chooses the decoder specified by (III) as P6's decoder. This is contrary to what the performance figures should suggest. Our reasoning is as follows. Given 25% increase in die area, the decoder specified by (IV) generates more than four micro-ops in average in terms of *measure 2* per cycle. This may not be best matched to the superscalar capability of Intel P6 execution units that retire only 3 micro-ops per cycle. Therefore, Intel chooses the decoder specified by (III) as P6's decoder.

Note that CPU performance is not only influenced by the performance of a decoder, but also by some other factors such as balancing of a pipeline, clock cycle time, etc. Since only the performance of a decoder can be quantified in terms of the number of micro-ops generated and the number of x86 instructions translated, we will use decoder performance instead of CPU performance in our studies for various decoders. We also assume that these two decoders are all based on the same micro-architecture. That is, an x86 instruction will be translated into the same set of micro-ops by these two decoders.

Table 9. The performance measures of the decoders in [9] under the decoding scheme 1.

Instruction Mix	(I)			(II)			(III)			(IV)			(V)		
80,10,05,2.5,2.5	1.894	2.604	2.367	1.922	2.643	2.403	2.656	3.652	3.320	2.769	3.807	3.461	3.541	4.869	4.426
75,15,05,2.5,2.5	1.870	2.664	2.430	1.922	2.739	2.499	2.562	3.651	3.331	2.769	3.945	3.599	3.541	5.045	4.603
70,15,10,2.5,2.5	1.838	2.803	2.573	1.911	2.915	2.676	2.447	3.731	3.426	2.725	4.155	3.815	3.434	5.235	4.806
65,20,10,2.5,2.5	1.799	2.834	2.609	1.911	3.010	2.771	2.316	3.647	3.358	2.725	4.291	3.951	3.434	5.406	4.977
60,20,10,05,05	1.717	3.005	2.576	1.838	3.217	2.757	2.122	3.714	3.184	2.523	4.414	3.783	3.066	5.361	4.596
55,15,15,10,05	1.670	3.256	2.839	1.791	3.493	3.046	1.994	3.889	3.390	2.356	4.594	4.005	2.719	5.302	4.622
50,40,05,2.5,2.5	1.642	2.750	2.544	1.922	3.220	2.979	1.892	3.170	2.933	2.769	4.636	4.290	3.541	5.928	5.485
45,40,05,05,05	1.578	2.801	2.604	1.911	3.393	3.154	1.741	3.222	2.786	2.725	4.834	4.493	3.434	6.088	5.660
40,35,15,05,05	1.496	2.991	2.617	1.818	3.636	3.182	1.624	3.248	2.842	2.450	4.891	4.280	2.908	5.801	5.076
35,35,20,05,05	1.430	3.004	2.646	1.791	3.762	3.314	1.516	3.183	2.804	2.356	4.936	4.348	2.719	5.693	5.015
25,25,25,15,10	1.285	3.341	2.699	1.566	4.072	3.289	1.321	3.434	2.773	1.807	4.699	3.795	1.897	4.933	3.984
Average	1.656	2.914	2.591	1.846	3.282	2.915	2.017	3.504	3.104	2.543	4.473	3.984	3.112	5.424	4.841

Table 10. The performance/die-area of (I) to (V).

	(I)			(II)			(III)			(IV)			(V)		
Performance	1.656	2.914	2.591	1.846	3.282	2.915	2.017	3.054	3.104	2.543	4.473	3.984	3.112	5.424	4.841
Die area	75%			87%			100%			125%			200%		
Ratio	2.208	3.885	3.455	2.122	3.772	3.351	2.017	3.054	3.104	2.034	3.578	3.187	1.556	2.712	2.421

Table 11. The average performance measures of decoders

	SWS	Decoding Scheme 1			Decoding Scheme 2			Decoding Scheme 3			Decoding Scheme 4			Decoding Scheme 5		
Case (1)	0	1.887	3.371	2.991	1.887	3.371	2.991	1.887	3.371	2.991	1.887	3.371	2.991	1.887	3.371	2.991
	1	1.887	3.371	2.991	1.887	3.371	2.991	1.887	3.371	2.991	1.921	3.435	3.047	1.921	3.435	3.047
	2	1.887	3.371	2.991	1.887	3.371	2.991	1.887	3.371	2.991	1.922	3.439	3.050	1.922	3.438	3.050
Case (2)	0	2.401	4.226	3.326	2.401	4.226	3.326	2.401	4.226	3.326	2.451	4.320	3.396	2.451	4.320	3.396
	1	2.401	4.226	3.326	2.401	4.228	3.327	2.402	4.229	3.328	2.543	4.492	3.526	2.544	4.493	3.527
	2	2.401	4.226	3.326	2.402	4.228	3.327	2.402	4.229	3.328	2.557	4.521	3.546	2.557	4.519	3.545
Case (3)	0	2.752	4.790	3.788	2.761	4.807	3.802	2.761	4.807	3.802	2.884	5.035	3.974	2.883	5.033	3.973
	1	2.752	4.790	3.788	2.784	4.845	3.834	2.802	4.876	3.859	2.997	5.234	4.130	3.013	5.261	4.153
	2	2.752	4.790	3.788	2.794	4.861	3.847	2.815	4.900	3.879	3.033	5.297	4.179	3.054	5.334	4.209
Case (4)	0	2.810	4.886	3.875	2.828	4.918	3.900	2.829	4.920	3.901	2.952	5.146	4.073	2.953	5.148	4.075
	1	2.810	4.886	3.875	2.850	4.959	3.933	2.864	4.984	3.953	3.080	5.376	4.253	3.094	5.400	4.273
	2	2.810	4.886	3.875	2.855	4.969	3.941	2.875	5.001	3.965	3.119	5.450	4.310	3.134	5.478	4.331
Case (5)	0	2.958	5.093	4.048	3.001	5.169	4.108	3.011	5.185	4.121	3.195	5.518	4.376	3.205	5.536	4.390
	1	2.958	5.093	4.048	3.028	5.213	4.145	3.056	5.257	4.181	3.309	5.712	4.531	3.334	5.750	4.561
	2	2.958	5.093	4.048	3.037	5.227	4.157	3.073	5.284	4.202	3.356	5.791	4.593	3.384	5.836	4.627
Case (6)	0	2.336	3.964	3.143	2.365	4.008	3.180	2.365	4.008	3.180	2.463	4.165	3.303	2.460	4.160	3.299
	1	2.336	3.964	3.143	2.374	4.020	3.190	2.381	4.032	3.200	2.495	4.211	3.341	2.495	4.211	3.342
	2	2.336	3.964	3.143	2.828	4.022	3.488	2.389	4.043	3.208	2.510	4.233	3.360	2.513	4.236	3.362
Case (7)	0	2.209	3.820	2.334	2.209	3.820	2.334	2.209	3.820	2.334	2.368	4.105	2.500	2.368	4.105	2.500
	1	2.209	3.820	2.334	2.209	3.820	2.334	2.209	3.820	2.334	2.493	4.322	2.632	2.493	4.322	2.632
	2	2.209	3.820	2.334	2.209	3.820	2.334	2.209	3.820	2.334	2.533	4.395	2.672	2.531	4.392	2.671
Case (8)	0	2.396	4.120	2.536	2.397	4.121	2.537	2.397	4.121	2.537	2.655	4.579	2.807	2.655	4.579	2.807
	1	2.396	4.120	2.536	2.399	4.125	2.540	2.403	4.130	2.543	2.779	4.792	2.939	2.782	4.797	2.943
	2	2.396	4.120	2.536	2.400	4.127	2.541	2.405	4.132	2.545	2.827	4.878	2.989	2.831	4.877	2.984
Case (9)	0	2.436	4.164	2.569	2.442	4.173	2.575	2.442	4.173	2.575	2.776	4.758	2.923	2.776	4.758	2.923
	1	2.436	4.164	2.569	2.445	4.178	2.579	2.448	4.184	2.583	2.871	4.915	3.021	2.873	4.919	3.024
	2	2.436	4.164	2.569	2.445	4.179	2.580	2.451	4.188	2.586	2.912	4.985	3.063	2.917	4.991	3.060
Case (10)	0	2.140	3.637	2.228	2.151	3.652	2.239	2.151	3.652	2.239	2.403	4.071	2.491	2.403	4.071	2.491
	1	2.140	3.637	2.228	2.153	3.655	2.241	2.157	3.660	2.245	2.444	4.134	2.533	2.446	4.137	2.535
	2	2.140	3.637	2.228	2.153	3.655	2.242	2.159	3.664	2.248	2.466	4.167	2.555	2.470	4.173	2.559
Case (11)	0	1.574	2.703	0.929	1.574	2.703	0.929	1.574	2.703	0.929	1.923	3.288	1.141	1.923	3.288	1.141
	1	1.574	2.703	0.929	1.574	2.703	0.929	1.574	2.703	0.929	1.969	3.361	1.171	1.969	3.361	1.171
	2	1.574	2.703	0.929	1.574	2.703	0.929	1.574	2.703	0.929	2.001	3.414	1.191	2.001	3.414	1.191

5.2 Decoders supporting high degree of superscalar capability

After studying the performance of various decoders once considered for P6 processor, we are wondering what kind of decoder architectures would be better in terms of performance and die area cost when higher degree of superscalar capability is required. In this section we would like to address this issue based on the design options mentioned in section 2. The following decoders which could support up to 8-way issues are considered.

Case (1). *D(S(0,1), G(0,2), C(2,4), H, W).*
Case (2). *D(S(0,1), G(1,2), C(2,3), H, W).*
Case (3). *D(S(2,1), G(0,2), C(2,3), H, W).*
Case (4). *D(S(1,1), G(2,2), C(1,3), H, W).*
Case (5). *D(S(3,1), G(1,2), C(1,3), H, W)*
Case (6). *D(S(5,1), G(0,2), C(1,3), H, W).*
Case (7). *D(S(0,1), G(4,2), C(0,3), H, W).*
Case (8). *D(S(2,1), G(3,2), C(0,3), H, W).*
Case (9). *D(S(4,1), G(2,2), C(0,3), H, W).*
Case (10). *D(S(6,1), G(1,2), C(0,3), H, W).*
Case (11). D(S(8,1), G(0,2), C(0,3), H, W).

From now on, the decoder specified by a *case* will simply be referred by its case number if no confusion arises. For example, case (1) is used to indicate the decoder specified by case (1). We assume that the decoders are all based on the same micro-architecture. Table 11 shows the average performance for each decoder. The performance figures will be analyzed in the following four sections.

5.2.1 Performance and constituents of decoders

Figure 3 shows the average performance of decoders based on the decoding scheme 1. Case (5) has the best performance and case (11) has the worst performance. Comparing the performance of case (5) to that of case (11), case (5) is respectively 187.9%, 188.4% and 435.7% better than case (11) in terms of the three measures. Cases (3), (4) and (5) have better performance and case (5) gives the best performance. These three decoder architectures are composed of various types of translators. For example, the decoder architecture of case (5) consists of three type *S* translators, one type *G* and one type *C* translator. Consequently, if a decoder architecture which consists of various types of translators will have better performance.

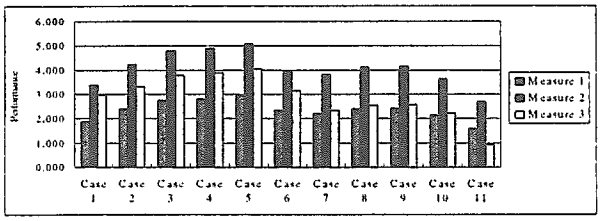

Figure 3. The performance measures of decoders based on the decoding scheme 1 and scan window size 0

5.2.2 Performance and decoding schemes

Based on the results in Table 11, since the relative performance generated by the five decoding schemes is almost invaried with the decoder constituent, the decoder specified by case (5) which has the best performance is selected for further studies. Figure 4 shows the performance of case (5) for the three measures for all decoding schemes without a scan window. The decoding schemes 2, 3, 4 and 5 have better performance than the decoding scheme 1. The performance of the decoder is improved by reordering of the instructions. We can find that the performance figures based on the decoding schemes 2 and 3 are approximately equal. The performance figures based on the decoding schemes 4 and 5 are also approximately equal. However, the performance gap between the decoding schemes 3 and 4 is evident. Thus, the influence of with or without lookahead on the performance is larger than that of bypassing strategy. Nevertheless, the influence of reordering the instructions on the performance is still very limited (about 8.69%). This outcome may be possibly explained as follows. From Example 2, we find that if we maximize the number of instructions decoded (or the number of micro-ops generated) in one cycle, it is equivalent to saying that we could possibly minimize the number of instructions decoded (or the number of micro-ops generated) in another cycle. So the canceling effect makes the decoding schemes having limited influence on decoder performance. Note that since cases (1), (7) and (11) consist of only one type of translators, they have the same performance under the decoding schemes 1, 2 and 3.

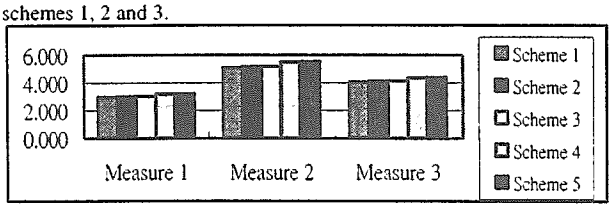

Figure 4. The performance measures of case (9) based on different decoding schemes and scan window size 0.

5.2.3 Performance and scan window size

Similarly, since the relative performance generated by implementing different scan window sizes almost invaries with the decoder constituents, the decoder specified by case (5) is also selected for further studies. Figure 5 shows the performance of case (5) under the decoding scheme 5 with a scan window sizes 0, 1 and 2. We can also find that the influence of increasing scan window size on the performance is also small. This outcome may be possibly explained as follows. The instructions contained in the scan window may have little chance being inspected due to limited number of translators or early encountering of an instruction that must be translated by the micro-sequencer. However, the performance of case (5) under the decoding scheme 5 with a scan window size 2 is increased by 14.5% when compared to that under the decoding scheme 1.

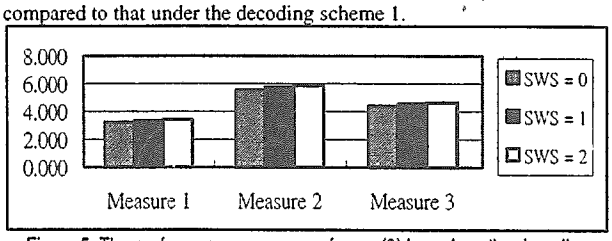

Figure 5. The performance measures of case (9) based on the decoding scheme 5 and different scan window sizes.

5.2.4 Performance and die area

The performance can be measured based on the approach proposed in [9]. The cost is measured in terms of die area. However, the die area is not known for the decoder specified in some cases. Thus, we will derive some information about the die area from [10]. These derived data will be used to compute the total area required by a decoder. Based on data about die area presented in Section 5.1.1, the following information is derived.

(1). The die area is increased by 25% when one type *S* translator is added to Intel P6's decoder. This information can be derived from (I) and (III).
(2). The die area is increased by 38% when one type *G* translator is added to Intel P6's decoder. This information can be derived from (II) and (IV).
(3). The die area is increased by 12% when one type *S* translator is replaced by one type *G* translator. This information can be derived from (I) and (II).
(4). The die area is decreased by 13% when two type *S* translators are replaced by one type *G* translator. This information can be derived from (II) and (III).

The die area of a type *C* translator can not be derived directly from [10]. However, it can be derived indirectly. Here a type *C* translator can have two kinds of decoding capability. One can translate an x86 instruction into one to three micro-ops, while the other can translate an

x86 instruction into one to four micro-ops. Moreover, based on the information presented above we can derived two different die areas for each kind of type *C* translator. All the derived data are distinguished into two groups as shown in the following.

Group (A) Rules:

(A.1). The die area is increased by 50% when one type *C* translator, generating one to four micro-ops, is added to Intel P6's decoder. Note that this die area is derived from subtracting the die area of P6's decoder by the die area of two type S translators, specified in (1).

(A.2). The die area is increased by 44% when one type *C* translator, generating one to three micro-ops, is added to Intel P6's decoder. Based on the die area obtained in (A.1) and the die area specified in (2), the die area is increased by 12% when one type *G* translator is replaced by one type *C* translator which can generate one to four micro-ops. We will assume that the die area is increased by 6 % when one type *G* translator is replaced by one type *C* translator which can generate one to three micro-ops. Thus, the die area of one type *C* translator which can generate one to three micro-ops is 44% of the area of P6's decoder.

Group (B) Rules:

(B.1). The die area is increased by 50% when one type *C* translator, generating one to three micro-ops, is added to Intel P6's decoder. By the same analogy, based on the area specified in (3), replacing one type *G* translator by one type *C* translator which generates one to three micro-ops increases die area by 12%. So based on die area specified in (2), adding one type *C* translator of this sort to P6's decoder increases die area by 50%.

(B.2). The die area is increased by 63% when one type *C* translator, generating one to four micro-ops, is added to Intel P6's decoder. By the same analogy, based on the area specified in (2) and (B.1), replacing one type *C* translator which generates one to three micro-ops by one type *C* translator which generates one to four micro-ops increases die area by 13%. So based on die area specified in (B.1), adding one type *C* translator of this sort to P6's decoder increases die area by 63%.

Based on the observations from Sections 5.2.1, 5.2.2 and 5.2.3, we can find that the influence of decoder constituent on the performance is most evident. The influence of the decoding schemes and scan window size on the performance is quite limited. Therefore, the constituent of a decoder rather than the decoding scheme or scan window size will ditate the performance of a decoder. Thus, in this section we will simply study the performance/cost ratio based on the decoding scheme 1 with scan window size 0.

Tables 12 and 13 show the die areas and performance based on *measure 2*. In Table 12 all the ratios from cases (1) to (5) are greater than 3. The ratios of cases (10) and (11) are less than 2. Among all these cases, case (3) has the best performance/cost ratio, while case (11) has the worst.

In Table 13 all the ratios of cases (2), (3), (4) and (5) are also greater than 3. The ratios of cases (10) and (11) are still less th .. 2. Among all these cases, case (4) has the best performance/cost ratio, while case (11) has the worst. Thus, when designing a new decoder architecture with high degree of superscalar capability, the decoders specified by cases (2), (3), (4) and (5) may be the better choices. If higher decoding performance is targeted, cases (3), (4) and (5) would be good candidates. The decoders specified by cases (10) and (11) will not be viable at all.

6. Conclusions

In this paper about a dozen of decoder architectures that can support up to 8-way issues of micro-ops are thoroughly studied in terms of decoding scheme, scan window size and constituent of a decoder. We find that the constituent of a decoder has a decisive influence on the performance of a decoder while the influence of the decoding schemes and scan window size on the performance is limited. The trade-offs between performance and die area for these decoders are also studied to opt for the best decoder for some underlying micro-architecture. In general, if a decoder has a variety of translator types would give better performance.

References:

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

[2] Tom R. Halfhill, " Intel's P6," BYTE, April 1995, pp. 42-58.

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

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

[5] 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.

[6] Michael Slater, " AMD's K5 Designed to Outrun Pentium: Four-Issues Out-of-Oredr Processor Is First Member of X86 Family," Microprocessor Report, Vol. 8, No. 14, October 24, 1994, pp. 1-11.

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

[8] Tom. R. Halfhill, " AMD K6 Takes on Intel P6," Byte, January 1996, pp. 67-72.

[9] Rung-Bin Lin and Chi-Ming Tsai, "Analytical Study of Performance Evaluation for x86 Instructions to Micro-ops Decoder, " the Proceedings of International Conference on Computer Architecture, International Computer Symposium, Kaohsiung, 1996, pp.113-120.

[10] David B. Papworth, " Tuning the Pentium Pro Microarchitecture," IEEE Micro, April 1996, pp. 8-15.

[11] Wen-Bin 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..

Table 12. The performance/cost of decoders under group (A) rules.

	Group (A) Rules										
	Case (1)	Case (2)	Case (3)	Case (4)	Case (5)	Case (6)	Case (7)	Case (8)	Case (9)	Case (10)	Case (11)
Performance	3.371	4.226	4.790	4.886	5.093	3.964	3.820	4.120	4.164	3.637	2.703
Die area	100%	126%	138%	145%	157%	169%	152%	164%	175%	187%	200%
Ratio	3.371	3.354	3.471	3.370	3.244	2.346	2.513	2.512	2.379	1.945	1.352

Table 13. The performance/cost of decoders under group (B) rules.

	Group (B) Rules										
	Case (1)	Case (2)	Case (3)	Case (4)	Case (5)	Case (6)	Case (7)	Case (8)	Case (9)	Case (10)	Case (11)
Performance	3.371	4.226	4.790	4.886	5.093	3.964	3.820	4.120	4.164	3.637	2.703
Die area	125%	138%	150%	151%	163%	175%	152%	164%	175%	187%	200%
Ratio	2.70	3.062	3.193	3.236	3.125	2.265	2.513	2.512	2.379	1.945	1.352