간단한 바이패싱 회로를 보상하는 VLIW 구조

김 석주
혜천대학 멀티미디어 전공

VLIW architecture for compensating simple bypassing paths

Seock-Ju Kim
Multimedia major, Haechon college
E-mail : kimse@hcc.ac.kr

요 약

본 논문에서는 NOP 이 차지하는 슬롯에 의미 있는 명령어를 중복 할당하여 자료의존 관계를 해소하고 프로그램 실행 사이클을 단축시키는 명령어 중복 스케쥴링 기법을 적용할 수 있는 VLIW 구조인 TiPs(Tiny Processors) 구조를 제안하였으며 TiPs는 회로의 복잡도를 증가시키지 않으면서 실행시간을 단축시켜 가상의 바이패싱 회로를 추가한 효과를 얻을 수 있다. 실험 결과 TiPs에서 명령어 중복 스케쥴링 기법을 적용할 경우 8% ~ 25%의 성능 향상 효과가 있음을 알 수 있었다.

1. 서론

최근의 프로세서 구조는 좀 더 큰 ILP (Instruction-Level Parallelism)을 활용하기 위한 기술들을 채택하고 있다. 수퍼스칼라 구조[1]나 VLIW 구조[2], 수퍼스칼라형인 SVLIW[3]가 현재의 방법들이다. 이러한 프로세서들은 여러 개의 실행 유니트에 의해 지원되는 n-way(보통 4-way) instruction issue를 지원한다.

VLIW(Very Long Instruction Word) 형태는 제어 흐름이 간단하고 각 연산처리기가 동기적으로 동작하기 때문에 매우 광범위하게 연구되고 있다[1,2]. 특히 근래에 들어와 멀티미디어를 빠르게 처리하는 프로세서 구조의 경우에는 미디어 정보가 가지고 있는 서브워드 병렬성(subword parallelism)을 활용하는 구조로 각광을 받고 있다[4].

이러한 VLIW 구조의 특징을 충분히 이용하기 위해서는 실행 시 동시에 실행이 가능한 명령어들을 모아 긴 명령어로 구성되는 목적 코드를 생성할 수 있어야 한다. 이들 구조에서 실행되는 프로그램들은 보다 큰 ILP를 도출하기 위한 적극적인 컴파일 기법을 필요로 한다. ILP를 활용하기 위한 구조에서 프로그램의 성능은 컨트롤과 자료의 의존성에 따라 제약을 받게 된다. VLIW 프로세서의 공통된 문제점은 코드가 늘어난다는 것이다. 컴파일러가 매 사이클마다 4개의 명령을 늘 스케쥴링할 수 없으므로 사용하지 않는 명령 슬롯에 NOP을 채워야 한다.

이 논문에서는 정수 연산 처리기로만 구성된 VLIW 프로세서를 위해 컴파일 된 코드의 NOP슬롯에 실행 코드를 넣어 데이터 의존성에 의해 야기되는 지연을 해결하는 VLIW 프로세서 구조를 제안하고 Tips (Tiny Processors) 라 명하였다.

TiPs는 VLIW구조에서 목적 코드에 존재하는 NOP(No Operation) 명령어 슬롯에 의미 있는 명령어를 중복 삽입하도록 함으로써 원래의 방법에서 존재하였던 자료의존 관계를 해소시켜 실행시간을 앞당길 수 있다.

이 경우에 동일한 단위 명령어가 둘 이상 긴명령어에 포함될 수 있으므로 쓰기 단계에서 여러 개의 단위명령어가 동일한 레지스터 파일로 쓰기를 함에 따라 충돌이 발생한다. 이때 여러 개중 하나의 명령만이 레지스터 파일로 쓰도록 허용하는 회로를 갖는 VLIW 구조를 제안하였다.

TiPs에서는 bypassing 회로를 사용하면서 목적코드 내에 존재하는 자료의존 관계로 인한 NOP슬롯에 의미 있는 명령어를 삽입하여 중복된 단위명령어를 실행시키면 그 수행 결과를 다른 연산처리기에 빠르게

전달한다. 이는 한 연산 처리기의 결과를 다른 연산 처리기에서 바로 사용할 수 있으므로 연산처리기 사이에 bypassing 회로가 만들어진 것과 같은 효과를 얻게된다.

이 논문에서는 TiPs의 하드웨어특성과 함께 명령어 중복 스케쥴링을 통해 프로그램의 전체 실행사이클이 줄어듦을 모의실험을 통해 보였다.

2. VLIW 프로세서 구조 : TiPs

이 장에서는 VLIW 프로세서가 정수연산처리기(IPU : Integer Processing Unit)만으로 구성된다고 할 때 bypassing 회로를 각 연산처리기 마다 하나씩만 갖는다고 가정한다.

그림 1 은 언급한 기능을 갖춘 n 개의 정수 연산 처리기만으로 구성된 VLIW 프로세서의 구성을 나타내며 이를 이 논문에서는 TiPs(Tiny Processors)라 하였다.

Tips의 연산처리기 마다 붙어 있는 bypassing 회로는 연산 처리기의 실행 결과가 레지스터 파일로 쓰기가 이루어질 때 그 결과를 다음 연산에서 입력으로 이용한다면 레지스터 파일에 쓰기 전에 연산처리기에서 사용할 수 있도록 한다.

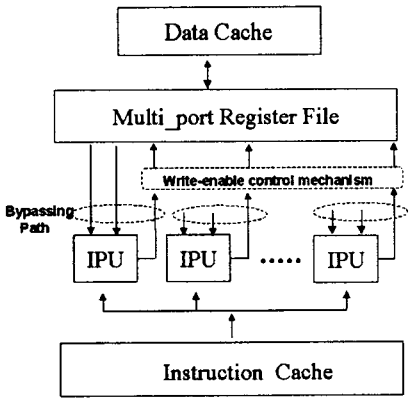

그림 1. VLIW 프로세서 — TiPs

각 연산처리기는 IF(instruction fetch), ID(Instruction Decode), EX(Execute)와 WB(Write Back)의 4단계로 명령어를 처리하는 파이프라인을 갖는 구조라고 가정하였다. 각 연산처리기 파이프라인의 매 단계는 한 사이클을 점유하며 EX단계에서 다른 연산처리기에서 수행중인 연산의 결과를 필요로 할 때 바이패싱 회로가 존재할 때에는 다른 연산처리기에서 WB단계를 거치기까지 기다릴 필요 없이 다른 연산처리기의 EX 단계의 결과를 피연산자로 EX단계에서 사용할 수 있다. TiPs의 파이프라인 단계는 그림 2 와 같다.

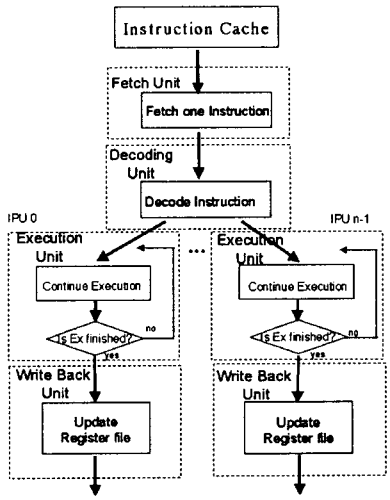

그림 2. TiPs의 pipeline 실행 단계

3. Bypassing 회로

bypassing 회로는 percolation scheduling compiler에 기반을 둔 VIPER 구조[6]의 기본 개념으로 사용된다.[5] bypassing mechanism을 그림 3 에 보였다.

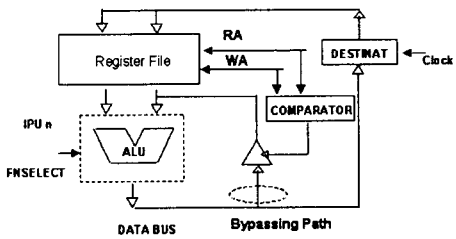

그림 3. bypassing path

bypassing 회로를 사용하면 정수 연산 처리기의 실행 결과(파이프라인 EX 단계의 결과)가 레지스터 파일로 write back 되기 이전에 연산 결과를 execution 단계의 입력으로 돌려 줄 수 있으므로 자료의존 관계로 인한 별도의 지연이 발생하지 않는다.

bypassing 하드웨어는 EX 단계에 진입하는 명령어의 입력(source) 레지스터 주소와 이전 명령어의 출력

(destination) 레지스터의 주소를 비교하여 만일 같다면 레지스터 파일로부터의 피연산자를 읽어 오지 않고 이전 EX 단계의 결과를 사용한다.

n 개의 연산 처리기로 구성된 머신에서 모든 연산처리기의 사이에 bypass 회로를 만들려면 $2dn^2$ (d는 decode와 write back 사이의 파이프라인 단계, 2는 명령어의 입력 operand의 개수를 2로 가정함.) 개 만큼 비교기(comparators)가 필요하므로 오버헤드가 매우 크다.[5]

따라서 TiPs는 각 정수 연산처리기마다 그 출력을 그 연산처리기의 입력으로 연결하는 bypassing 회로만을 갖는다고 가정하였다.

레지스터 파일은 multi-ported register file을 사용하여 1 unit time 내에 레지스터와 메모리에 관련된 입출력을 동시에 수행할 수 있다. 각 연산처리기마다 2개의 read와 1개의 write 및 명령어 dispatch unit로부터 1개의 write가 요구되며 이들을 동시에 수행할 능력을 갖고 있다. 또한 동시에 메모리 인터페이스를 통해 1개의 read/write가 처리된다.

4. 명령어 중복 스케쥴링

◈ 명령어 복사와 자료의존관계

Tips에서 사용하는 코드 복사 기법[7][8]을 보이기 위해 B = 3*(A+10)을 계산하기 위한 코드를 그림4.에 나타내었다.

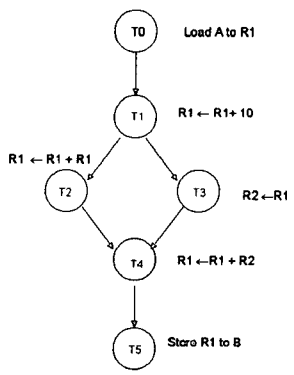

그림 4. 명령어 그래프

명령어 그래프는 하나의 기본 블록으로 구성된 그래프로 노드(node)는 명령어, 간선은 명령어간의 자료의존 관계를 나타낸다. 그림 4 의 코드를 긴명령어로 만들면 표 1 과 같다. 이때 같은 명령어 그래프로부터 코드 복사 방법을 통해 만들어진 긴 명령어는 표 2 와 같다.

	IPU0	IPU1
1 cycle	add $1, ($3) ,0	
2	add $1, $1 ,10	
3	move $2, $1	
4	add $1, $1, $1	
5	add $1, $1, $2	
6	add ($4), $1, 0	

표 1. 코드 복사 없음

	IPU0	IPU1
1 cycle	add $1, ($3) ,0	add $1, ($3) ,0
2	add $1, $1 ,10	add $1, $1 ,10
3	move $2, $1	
4	add $1, $1, $1	
5	add $1, $1, $2	
6	add ($4), $1, 0	

표 2. 코드 복사 있음

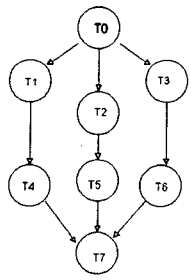

그림 5. 명령어 그래프

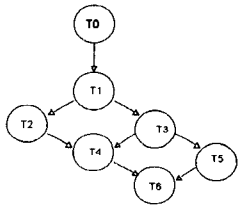

그림 6. 프로그램 의존 그래프

	IPU0	IPU1	IPU2
1			T0
2			T1
3		T3	T2
4		T6	T4
5			T5
6			T7

표 3. 그림 5 의 실행 결과(복사 없음)

	IPU0	IPU1	IPU2
1	T0복	T0복	T0
2	T3	T2	T1
3	T6	T5	T4
4			
5		T7	

표 4. 그림 5 의 실행 결과(복사 있음)

	IPU0	IPU1	IPU2
1			T0
2			T1
3			T3
4			T2
5		T5	T4
6			
7			T6

표 5. 그림 6 의 실행 결과(복사 없음)

	IPU0	IPU1	IPU2
1		T0복	T0
2		T1복	T1
3		T2	T3
4			T5
5		T4	
6		T6	

표 6. 그림 6 의 실행 결과(복사 있음)

그림 5 에 나타난 명령어 그래프를 갖는 프로그램을 코드 스케쥴링할 때 빈 명령어에 코드를 복사할 경우 나타나는 효과를 표 3 과 표 4 에 보였다. 표 5 와 표 6 은 그림 6에 나타난 는 코드 복사(명령어 중복 스케쥴링) 를 사용함으로써 실행 사이클의 수가 각각 1개 씩 줄어들게 됨을 보여준다.

◆ 실험

명령어 중복 스케쥴링의 효과를 보이기 위해 4개의 정수형 연산처리기로 구성된 타겟 머신을 대상으로 제안한 명령어 중복 할당 기법의 성능을 평가하였다. 명령어 중복 스케쥴링 알고리듬[7,8]을 적용할 때 성능에 영향을 끼칠 수 있는 요소로서 명령어 수준 병렬성, 명령어들 간의 자료의존성의 빈도, 그래프의 크기를 고려하였으며 이들 요소들에 따라 다양한 명령어 그래프를 발생할 수 있는 프로그램을 개발하여 실험하였다.

바이패싱 회로가 없을 때 다른 연산처리기로부터 피연산자를 가져와야 하는 경우에는 그 연산처리기가 WB단계를 수행한 후에 그 결과를 사용할 수 있으므로 EX 단계가 한 사이클 지연되는 것으로 가정하였다. 즉, 피연산자를 필요로 하는 EX가 다음 사이클에 수행되어야 하므로 NOP 명령어가 삽입되게 된다.

또한, 타겟 머신은 명령어 처리를 파이프라인 방식으로 처리하므로 매 사이클마다 각 연산처리기가 IF, ID, EX, WB의 단계 중 동일한 하나의 단계를 수행하므로 명령어의 인출이나 피연산자의 인출 시 캐시미스가 발생하지 않는 것으로 간주하였다. 즉, 캐시미스가 발생하면 파이프라인이 정지되므로 이는 실험에서 캐시 또는 피연산자를 저장하는 레지스터 파일의 크기가 무한대임을 가정하는 것과 마찬가지 의미를 지닌다.

또한, 공정한 실험을 위하여 실험에서는 임의의 자료의존관계를 가지는 명령어 그래프를 임의로 발생시키도록 하였으며 실험의 대상이 되는 명령어 그래프를 구분하는 요소로서 다음과 같은 항목을 고려하였다

데이터 의존 관계의 변화를 모의하기 위하여 명령어 그래프에 포함된 간선의 수를 변화시켰다. 이는 명령어 그래프 밀도(density) d 로 나타내며 이는 인접된 그룹에 속한 노드들 사이에 존재할 수 있는 최대 간선의 수에 대한 실제로 존재하는 간선 수의 비이다. 따라서 d 가 100%이면 이 그래프는 주어진 명령어 수준 병렬성을 만족하되, 인접 레벨의 그룹에 속한 모든 노드들 사이에 자료의존관계가 존재하는 그래프이다. 만일 d 가 0%이면 이는 결국 그래프 상에 어떤 노드간에도 자료의존관계가 존재하지 않으므로 결국 하나의 레벨로 구성된 그래프임을 의미한다.

명령어 수준 병렬성 δ 는 경로길이가 같은 노드들을 하나의 그룹이라고 했을 때, 각 그룹에 포함되는 명령어의 최대 값으로 이는 그래프 전체의 병렬성과 같은 값이다. 즉, $\delta = \max(\delta_l)$, $l = 1,2,\cdots,L$, 여기서 δ_l는 레벨이 l 인 노드들의 개수이다.

여기서 L 은 그래프의 길이를 나타내는데 명령어 그래프의 뿌리 노드에서 마지막 노드까지의 경로의 길이 중에서 가장 큰 값이다. 즉, 그래프의 시작 노드에서 각 명령어 i_i 에 이르는 경로의 최대 길이를 그 명령어가 속한 그룹의 레벨이라고 하면 레벨의 최대값을 그래프의 길이 L 로 정의하였다.

표 7은 L이 10인 경우에 δ 를 3~5, d 를 60%~80%로 변화시키면서 생성한 명령어 그래프를 대상으로 스케쥴링한 결과이다. 각 경우에 대하여 주어진 특성 값에 대해 무작위로 30개 씩 명령어 그래프를 생성시켜 실행 시간의 합계를 구한 것이다.

(a) d = 60%

병렬성 (δ)	바이패싱 토폴로지		
	P_1		P_2
	org	dup	org
3	441	407	359
4	492	393	365
5	534	452	438
6	544	436	420

(b) d = 70%

병렬성 (δ)	바이패싱 토폴로지		
	P_1		P_2
	org	dup	org
3	468	433	358
4	513	421	395
5	544	471	446
6	576	514	497

(a) d = 80%

병렬성 (δ)	바이패싱 토폴로지		
	P_1		P_2
	org	dup	org
3	494	458	374
4	521	427	396
5	565	514	409
6	553	489	443

표 7. 실험 결과

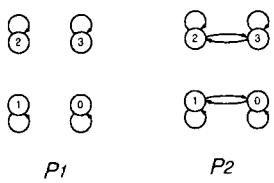

그림 7. bypassing topology

그림 7 에서 $P1$ 은 TiPs의 bypassing 회로의 토폴로지를 나타내며 $P2$ 는 초기 VLIW 구조의 대표적 머

신인 VIPER의 bypassing 회로의 토폴로지를 나타낸다. 그림 8 은 표 7의 실험결과를 그래프로 표시한 것이다.

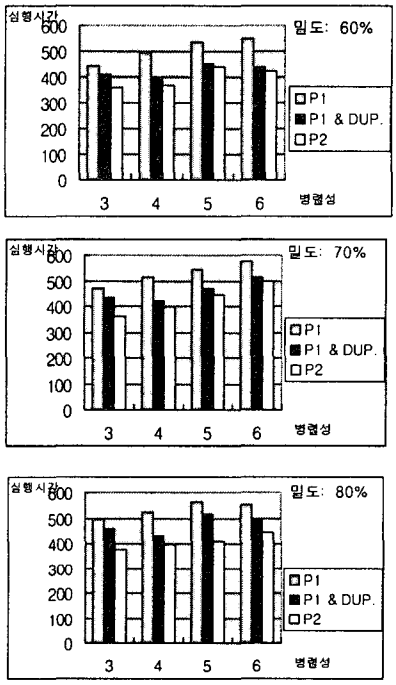

그림 8. 실험 결과

명령어 수준 병렬성이 각각 3과 4인 경우는 명령어 그래프의 간선의 밀도의 변화에 따른 상대적 속도 향상의 변화가 큰 차이를 보이지 않는다. 이는 작은 병렬성을 갖는 프로그램에서는 자료의존성이 증가하더라도 상대적으로 연산처리기의 수가 많으므로 여분의 연산처리기에 명령어 중복 스케쥴링에 의한 가상의 바이패싱 회로가 많이 만들어지는 효과가 유지되어 지연이 발생할 소지가 적다고 판단된다.

그러나 명령어 수준 병렬성이 5와 6인 경우에는 밀도가 높아질수록 상대적 속도 향상이 낮아지는 것을 알 수 있다. 이는 명령어 그래프 밀도가 높아짐에 따라 명령어간의 자료의존성이 높아 순차 처리해야 하는 명령어의 수가 늘어나기 때문인 것으로 판단된다.

5. 동시 접근에 따른 하드웨어적 고려

어떤 레지스터파일의 구조는 비용을 줄이기 위해 일정 뱅크에 대해 제한된 포트를 통해서만 접근되도록 설계되는 경우가 많다. 즉 레지스터 파일의 포트가 늘어나면 크기(die size)의 증가 뿐 아니라 fan-in 과 fan-out 문제를 발생시킨다. fan-out 을 갖는 회로의 지연은 load capacitance의 증가에 로그적(logarithmically)으로 증가하도록 만들 수 있지만 fan-in 문제는 fan-out 문제만큼 쉽게 해결할 수 없다. 즉 fan-in이 증가하면 bus line의 voltage swing 을 줄이는 기술을 요구하므로 어느 정도 속도에 대한 voltage margin을 포기해야 한다.[9] 따라서 멀티포트 레지스터 파일의 포트 수를 늘리는 것은 기술적으로 어려운 문제로 남아 있다.

Tips 구조에서 명령어 중복 스케쥴링을 적용할 경우 여러 개의 연산처리기에서 동일한 명령어를 수행하므로 같은 레지스터에 대한 read/write 접근이 동시에 이루어지게 된다. 연산처리기들이 중복된 명령으로 인해 동시에 레지스터 파일로 쓰기를 할 때 이를 제어하는 기능에 대해 논한다.

이 경우 모든 연산처리기로부터 같은 클럭 내에서 여러 번의 접근을 허용하도록 설계된 monolithic 레지스터파일 구조는 동일 명령어에 의한 동시 접근이 문제가 되지 않는다.

그러나 뱅크 형태로 설계된 레지스터 파일에서는 중복된 명령어에 의한 레지스터 접근의 빈도가 높아질 수 있으므로 이러한 경우는 register renaming을 사용하는 방법을 고려할 수 있다. 그러나 register renaming 역시 제한된 개수의 레지스터를 포함한 레지스터 파일을 고려해 보면 이 방법은 제한적이다.

따라서 여러 개의 연산처리기가 중복된 명령으로 인해 같은 결과를 동시에 같은 레지스터에 write하고자 함으로 write back단계에서 연산 결과의 유효 여부를 결정하는 정보가 필요하다.

이를 위하여 명령어 별로 별도의 1 bit를 추가하여 write back 단계에서 레지스터 파일에 연산 결과가 쓰여질 수 있는지 여부를 판단하도록 하였다. Tips구조의 긴 명령어의 형태는 그림 9 와 같다.

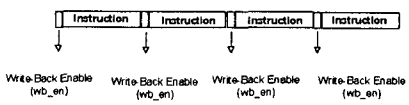

그림 9. TiPs의 명령어 구조

즉 각 단위 명령어는 *wb_en.Instruction* 와 같이 기존의 명령어 *Instruction* 에 write back 단계에서 레지스터 파일에 연산 결과가 쓰여질 수 있는지 결정하는 정보를 갖는다. 이를 위해 데이터 버스에 1 bit가

추가 되게 된다. 원래의 단위명령에 추가된 *wb_en* 을 1로 하고 복사된 명령은 *wb_en* 을 0 으로한다.

그림 10 의 회로를 이용하여, 여러 개의 연산처리기의 실행결과가 동시에 같은 레지스터로 전송이 될 때 복사된 명령과 원본을 구분하여 원본이 할당된 연산처리기에서 전송된 결과가 쓰여지게 된다.

그림 10 의 명령레지스터에는 각 단위 명령어에 추

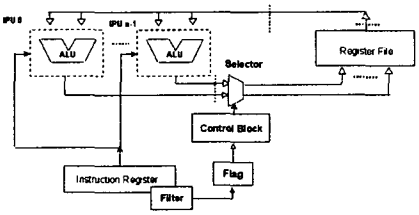

그림 10. wb_en 정보에 의한 쓰기 제어 회로

가된 wb_en 비트를 추출하는 필터가 붙어있다. wb_en정보는 latch 에 flag로 저장되며 단위 명령어가 연산 처리기에서 수행된 후 이 정보를 이용하여 제어회로가 결과가 레지스터 파일로 보내지는 것을 조정한다.

데이터패스 상에서 연산처리기와 레지스터 파일은 물리적으로 분리되어 있는데 이는 연산처리기의 결과는 레치에 저장되어 있다가 각 명령 사이클에 따라 해당 레지스터로 전송됨을 의미한다. 실제적인 데이터패스의 연결 구조는 레지스터 파일이 여러 개의 뱅크로 구성된 형태를 갖는 등 다양한 레지스터 파일의 구조에 따라 그 구성이 바뀔 수 있다.

6. 연구방향

VIPER는 연산처리기 자신과 인접한 연산처리기 사이에 바이패싱 회로를 갖는데 이것은 하드웨어 부담이 TiPs보다 높고 연산처리기의 수가 증가할 경우 TiPs처럼 떨어져 있는 경우에 연산 처리기 사이에 바이패싱 회로가 만들어지지 않으므로 연산 처리기의 수가 6개나 8개정도 증가하는 경우에 TiPs의 유용성을 조사할 예정이다.

5. 결론

TiPs에서 명령어 중복 스케쥴링으로 NOP슬롯에 명령어를 복사하면 복사된 명령을 처리하는 연산처리기와 원래 명령이 처리된 연산처리기 사이에는 바이패싱 회로가 존재하는 것과 같은 효과를 갖는다.

TiPs 의 방법은 프로그램에 따라 NOP 슬롯이 있는 경우에 제한적으로 바이패싱 회로가 만들어지는 것과 같은 효과가 있지만 TiPs가 비교적 적은 하드웨어 부담(쓰기제어 회로) 과 간단한 명령어 중복 스케쥴링을 사용하므로 경제적이라고 말 할 수 있다.

제안된 TiPs 구조는 비교적 간단한 하드웨어의 증가와 복사 코드 스케쥴링을 사용하여 그림 9에 보인 대로 더 복잡한 bypassing path를 갖는 구조와 거의 대등한 성능을 가질 수 있음을 보였다.

[참고문헌]

[1]Nicolau Alexandru, Roni Postasman,"An environment for the development of microcode for pipelined architectures," In *Proc. ISCAS '92,* Vol.6, pp. 2677-2680, 1992

[2] Arthur Abnous, Nader Bagherzadeh,"Special features of a VLIW architecture," In *Proc. Fifth International Parallel Processing Symposium*, Vol. 1 No. 1, pp. 224-227, 1991

[3] Boyoun Jeong, Joongnam Jeon and Sukil Kim, "Design of VLIW architectures minimizing dynamic resource collisions," *Journal of KISS,* vol.24, No.4, pp. 357-368, April 1997

[4]Nicolau Alexandru, Roni Postasman, "Realistic Scheduling: compaction for pipelined architectures," *IEEE Micro,* Vol. 23, No.4, pp. 69-79, 1990

[5] Arthur Abnous, Nader Bagherzadeh,"Pipelining and bypassing in a VLIW Processor," *IEEE Transactions on Parallel and Distributed Systems.* Vol. 5, No.6. pp. 658-663, 1994

[6] Jeffrey Gray, Andrew Naylor, Arthur Abnous a nd Nader Bagherzadeh,"VIPER: a VLIW integer mic roprocessor," *IEEE Jouranal of Solid_State Circuit s,* Vol.28, No 12, pp. 57-68, 1993.

[7]. 문현주, 전중남, 김석일, 황인재, "통신의 영향을 줄이기 위한 이기종 태스크 스케쥴링 기법", 정보처리 학회 논문지, 제 5권 제 10호, pp 2521~2532, 1998년

[8]. S. Manoharan, "Augmenting work greedy assignment schemes with task duplication." In *Proc. ICPADS'97,* Vol.5 N0.6, pp. 772-777, 1997

[9] James Meindl, Low Power CMOS Design, Kluwer Academic Publishers, 1995