# VLIW 시스템에서의 최소 시간 지연을 갖는 효율적인 병렬 파이프라인 알고리즘

서 장 원† 송 진 희†† 류 천 열††† 전 문 석††††


## 요 약

본 논문은 VLIW(Very Long Instruction Word) 시스템에 대한 파이프라이닝 알고리즘 문제와 파이프라인 처리에서 발생되는 시간 지연을 최소화할 수 있는 효율적인 파이프라인 처리 방법에 대해 서술하였다. 제안된 알고리즘은 병렬로 수행하면서 병렬 파이프라인 처리되며, 기본 오퍼레이션의 조합으로 응용 목적에 따라 다양한 기능을 수행하는 명령어의 설계가 가능하다. 본 논문에서는 프로세서의 파이프라인 알고리즘 효율성과 제안된 방법에 의해 시간 지연이 최소화됨을 다른 파이프라인 방법과의 비교 분석을 통해 증명해 보인다.


# An Effective Parallel and Pipelined Algorithm with Minimum Delayed Time in VLIW System

Jang Won Suh,† Jin Hee Song ††, Ryou Chun Yeol ††† and Moon Seog Jun††††


## ABSTRACT

This paper describes pipelining algorithm issues for a VLIW(Very Long Instruction Word) System and the effective pipelined processing method by occurrence in pipelined mamegement of processor minimized to timing delay. The proposed algorithm is executed in pipeline and parallel processings, and by combining basic operations variable instruction set can be desinged for various applications. In this paper, we prove and analyze the efficiency of the proposed pipeline algorithm and compare with other processor pipeline algorithm in terms of time minimizing.


## 1. 서 론

최근 반도체 소자 제조 기술의 발달과 컴파일러 최적화 기법의 발전으로 명령어 수준에서 병렬성을 추출하여 한 클럭 주기에 다수의 명령어를 실행함으로써 처리 속도를 고속화하고 강력한 계산 능력을 갖는 대형 병렬 처리 컴퓨터에 대한 관심과 연구 개발이 활발히 진행되고 있다. 초기의 표준 RISC(Reduced Instruction Set Computer) 형태의 프로세서가 명령어의 간소화를 통해 시스템을 단순화하고 명령어의 복합적인 파이프라인 처리를 효율적으로 수행하여 CISC(Complex Instruction Set Computer)와 비교하여 상대적으로 프로세서의 성능을 향상시켰다. 그러나, 한 클럭 당 하나의 명령어만을 실행하기 때문에 명령어를 파이프라인 처리하여도 프로세서가 도달할 수 있는 최대 성능 향상은 CPI(Clock Per Instruction)를 1 이하로 줄일 수 없고, 실제 프로그래밍 환경에서는 데이타 종속 관계, 분기 명령 및 프로세서의 구조적인 문제 등으로 인해 발생하는 파이프라인 시간 지연 때문에 파이프라인 처리가 어렵다. 이러한 구조상의 한계를 극복하고자 프로세서에 여러개의 FU(Functionl Unit)를 구비하고, 각각의 FU들을 효율적으로 이용할 수 있도록 하는 병렬 처리 컴퓨터에 대한 연구가 대두되고 있다[8]. 이러한 관점에서 볼 때, 현재까지 제안된 여러 병렬 처리 컴퓨터들 중, 비교적 이들 조건들을 가장 잘 만족시키는


† 정 회 원:숭실대학교 전자계산학과 석사과정
†† 정 회 원:신흥전문대학 전산정보처리과 조교수
††† 정 회 원:숭실대학교 전자계산학과 부교수
†††† 종신회원:유한전문대학 전자계산학과 부교수

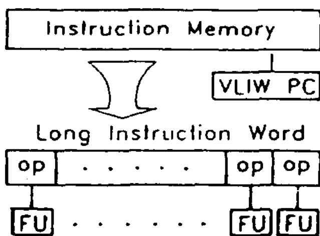
논문접수:1995년 2월 6일, 심사완료 : 1995년 5월 16일

병렬 처리 컴퓨터로는 VLIW(Very Long Instruction Word) 시스템을 들 수 있다.

## 2. VLIW 시스템의 개요

VLIW 시스템은 한 번에 처리되는 수행 단위인 명령 워드의 길이가 수 백 내지 수 천 비트로 일반 프로세서의 명령어 보다 매우 길다. VLIW 시스템의 구성은 (그림 1)과 같다.

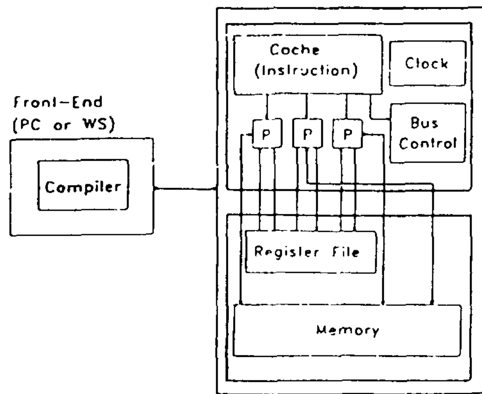
여기서 하나의 명령 워드는 많은 수의 부속 명령어들로 이루어진다. 예를 들면, op1, op2, op3, op4, · · ·, opn 등의 여러 명령어들이 있을때, 이들을 프로세서 수 만큼의 일정 단위로 묶어서 하나의 명령 워드 {op1, op2, op3, op4, · · · opn}를 구성하며, 이 명령 워드 내의 각 부속 명령어는 여러 프로세서에서 분산되어 동시에 수행된다. VLIW 시스템에서의 명령 워드는 단일 프로세서에서의 한 명령어와 동일한 개념이므로 하나의 PC(Program Counter)에 의해 전체 프로그램 수행의 흐름이 제어된다. 또한, 가장 두드러진 특징 중에 하나는 전반적으로 하드웨어의 복잡도가 매우 줄어드는 반면에, 컴파일러의 역할은 매우 증가한다는 점이다. 특히, VLIW 시스템에서 프로그래밍 요구 사항의 최소화란 일반적인 순차 프로그래밍 기법과 언어를 그대로 사용할 수 있다는 것을 의미하며, 이에 대한 병렬화 작업은 전적으로 컴파일러가 수행한다.




(그림 1) 전형적인 VLIW 시스템
(Fig. 1) Typical VLIW system

VLIW 시스템이 다중 프로세서나 벡터 프로세서와 비교되는 특징들은 모든 프로세서는 같은 주소에서 명령어를 읽지만 명령어는 각 프로세서마다 다르며, 매우 긴 명령어를 사용한 폰 노이만 식 제어 흐름을 사용한다는 것이다. 그러므로 일반적으로 이상적인 VLIW 시스템의 구성 형태는 여러개의 프로세서 또는 처리 단위가 대용량의 레지스터 화일을 공유하고, 이 레지스터 화일을 통하여 작업 할당과 통신이 이루어진다[6]. VLIW 시스템의 구조는 (그림 2)와 같다.

VLIW 시스템에서 사용되는 프로세서는 모든 명령 수행 시간이 동일해야 하는 특성 때문에 RISC 프로세서 또는 유사한 특성을 가진 프로세서의 사용을 기본적으로 가정한다. 기존의 VLIW 시스템에서는 제작자가 자체적으로 프로세서를 설계하여 사용하였다. 이는 VLIW시스템의 특성상 레지스터 화일과 메모리 접근 경로 또는 프로세서간 자료 교환 작업 등 모든 프로세서에 관련된 자원 할당이 컴파일 시간에 결정되고 프로세서 외부에서 이를 통제할 수 있어야 하기 때문이다.




(그림 2) VLIW 시스템 구성도
(Fig. 2) The organization of VLIW system

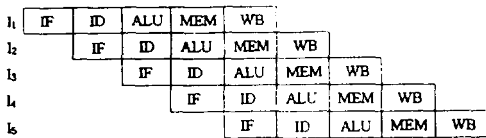
따라서 이상적인 VLIW 시스템을 구현하기 위해서는 VLIW 시스템 전용의 새로운 프로세서를 제작하여 사용하는 것이 가장 바람직하다. 이와 같은 VLIW 시스템에서의 병렬 처리 기법들은 최근에 널리 보급되기 시작한 슈퍼스칼라 프로세서, 대규모 다중 처리 시스템, 병렬 처리 시스템 등의 여러 분야에 활용될 수 있으므로 현재 관련 연구들이 활발하게 진행되고 있다.

## 3. 병렬 파이프라인 프로세서의 구조

### 3.1 파이프라인 구조에서의 처리 지연

CPU에서 파이프라인을 사용하는 목적은 자원을 쉬지않고 사용하여 프로세서의 성능을 향상시키는 것이다. (그림 3)은 RISC 프로세서에서 전형적으로 사용되는 파이프라인 구조를 보여준다.

그러나, 이러한 파이프라인 구조는 명령들의 중첩 수행으로 인한 자원 사용에서의 충돌 현상과 분기 발생시 목적 명령어의 fetch 지연으로 파이프라인의 효율이 감소하는 현상이 발생하게 되는데 이러한 현상을 시간 지연이라고 하며, 이러한 시간 지연을 해결하기 위해 RISC 프로세서는 바이패싱을 이용한다. 여기서는 보다 효율적인 파이프라인 구조의 설계에서 고려해야할 시간 지연과 순서 지연에 대해 서술하기로 한다.




(IF: Instruction Fetch, ID: Instruction Decode, MEM: Memory Access, WB: Write Back)

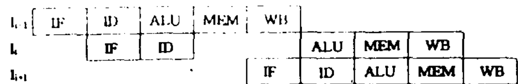
(그림 3) 5단계 파이프라인 구조
(Fig. 3) 5-step for pipeline structure

#### 3.1.1 시간 지연

파이프라인 구조 설계시에 고려해야할 사항 중에 하나가 시간 지연이다. (그림 3)에서, I5 명령어가 I3 와 I4 에서 계산된 두 개의 결과 값을 이용하려고 한다. 하지만, (I3, WB)와 (I4, WB)는 (I5, ALU) 단계 보다 같거나 이후에 수행되기 때문에 I5에서는 엉뚱한 자원을 사용하게 된다. 이와 같은 현상은 어떤 명령어의 한 요소가 아직 유효하지 않은 이전의 결과 값을 이용하려하기 때문에 발생하는 것으로서, 이를 방지하기 위해 필요한 자원이 기록·완료될 때까지 어떤 명령어를 지연시키는 기법인 시간 지연의 몇가지 대표적인 방법들을 소개한다.

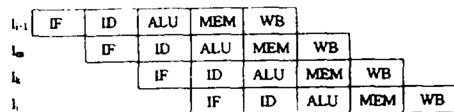
첫번째 방법은, 만일 (Ii-1, WB)의 결과가 (Ii, ALU)단계에서 필요하면 (Ii-1, WB)로 부터 (Ii, ALU)로 연결되는 바이패싱 경로를 두는 것이다. 이 기법은 다수의 파이프라인 시스템에서 사용되고 있는 것으로서 시간 지연 횟수를 줄여 준다. 그러나, 이 기법에서는 (Ii−1, WB)에서 계산된 결과가 (Ii, ALU)로 보내질 수 있도록 하려면 그 값이 빨리 안정을 취해야 한다. 또한, 한 단계에서 다른 단계로 데이타를 전송하려면 일정 시간이 필요하므로 기본 주기 시간을 추가로 부가해야 한다는 단점이 있으며, 이를 위한 부가적인 하드웨어의 구현으로 인한 상당한 부담이 뒤따를 수 있다.

두번째 방법은 준비되지 않은 데이타를 요구하는 명령어의 실행을 단순히 지연시키는 방법을 생각해 볼 수 있다. 아래의 파이프라인 구조는 (Ii−1, WB)와 (Ii, ALU) 사이의 시간 지연을 없애기 위해 한 명령에 종속된 명령의 실행을 단순히 지연시키는 것이다. 여기서 주의해야 할 것은 명령어 Ii의 지연이 명령어 Ii+1 에도 영향을 미친다는 점이며, 명령어 Ii+1은 두 파이프라인 단계 만큼을 지연해야 한다. 그러므로, 이 기법은 명령어 단위 처리량을 감소시키는 단점이 있다.




(그림 4) 명령어들을 지연시키는 파이프라인
(Fig. 4) Delayed instructions for pipeline

세번째 방법은, 두번째 기법의 단점을 없애기 위해 종속되는 명령어만을 지연시키는 방법을 고려하고 있다. 그래서 (Ii-1, WB)와 (Ii, ALU) 사이에 시간 지연이 발생되면 명령어 Ii-1 과 Ii 에 이 두 명령과 종속 관계를 갖지 않는 다른 두 개의 명령어를 삽입하여 파이프라인한다. 따라서 명령어 Im과 Ik는 명령어 Ii-1, Ii에 대해 비종속 관계를 유지한다.




(그림 5) 명령어 순서 변경
(Fig. 5) Modification instructions order

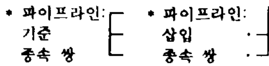
## 4. k차 종속 명령어 집합에 대한 파이프라인

### 4.1 ci=1(0≤i≤k)에 대한 파이프라인 알고리즘

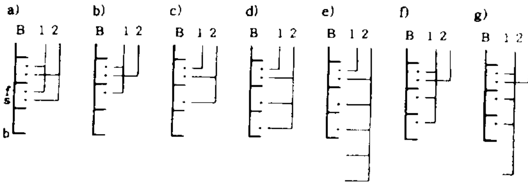
알고리즘 설명을 하기 전에 종속 명령어 쌍에 대한 파이프라인의 수행에서 살펴보아야 할 파이프라인 상태에 대해 먼저 알아보기로 하자.

(1) BASE 기준의 삽입 파이프라인 비종료 상태

기준쌍(BASE)에 대한 삽입 파이프라인을 수행할때, 기준쌍의 최고 종속차수가 b이고, 1삽입 파이프라인을 수행한 종속 명령어 쌍의 최고 종속차수가 f, 2삽입 파이프라인을 수행한 종속 명령어 쌍의 최고 종속차수를 s로 표현하자. 이때, 기준쌍에 대한 1, 2삽입 파이프라인이 완전하게 끝나지 않은 상태는 (그림 6)의 (a)~(g) 중 한 상태가 된다.

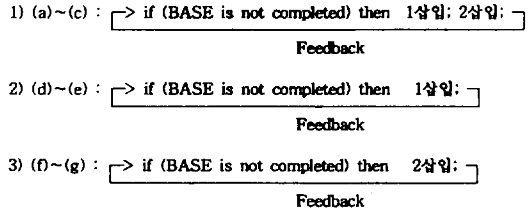



• 파이프라인:(b,B) - BASE의 최고 수행 차수는 b
종속 차수 (f,1) - 1 삽입의 최고 수행 차수는 f
와 종류 (s,2) - 2 삽입의 최고 종속 차수는 s



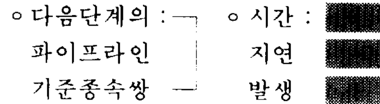

(그림 6) BASE 기준의 삽입 파이프라인 비종료 상태
(Fig. 6) Nonterminate state of insertion pipeline in BASE

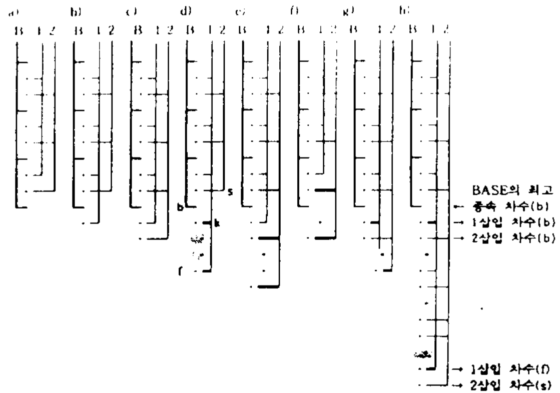
(그림 6)에서 알 수 있듯이 BASE에 대한 삽입 파이프라인이 종료되기 위해서는 다음 단계의 삽입 파이프라인을 반복하여야 한다. 이때 삽입 파이프라인의 반복 수행을 위해 되돌아가기 전, 반드시 삽입 파이프라인을 수행할 수 있도록 처리되지 않고 남아있는 명령어들이 있는지 먼저 확인하여야 한다. (그림 6)에 따른 BASE에 대한 삽입 파이프라인 처리 순서는 다음과 같다.




Feedback 즉, (a)~(c)는 1, 2 삽입이 모두 완료되지 않았으므로 BASE 내의 파이프라인이 종료되지 않았고, 아직 처리하지 않은 명령어가 남아있다면 1, 2 삽입 파이프라인의 두 단계를 번갈아 수행하여야 한다. (d)~(e)는 BASE 내의 2 삽입은 완료되었으므로 1삽입만 반복하면 되고, (f)~(g)는 1삽입은 완료되었으므로 2삽입만 반복 수행하면 된다.

(2) BASE 기준의 삽입 파이프라인 종료 상태




(그림 7) BASE 기준의 삽입 파이프라인 종료 상태
(Fig. 7) Terminate state of insertion pipeline in BASE

(그림 7)은 BASE를 기준으로 하는 파이프라인이 완료되었을 때 발생될 수 있는 완료 형태를 도식화한 것이다.

여기서 (a)~(c)는 BASE에 삽입 파이프라인한 종속 쌍으로부터 발생되는 시간 지연이 없다. 따라서 이 경우는 파이프라인해야 할 다음 종속

쌍의 명령어가 존재하면 그중 종속 차수가 가장 높은 쌍을 선택하여 BASE로 둔 후, 파이프라인의 수행 단계를 처음부터 반복한다. 그러나 (d)~(h)의 경우, 파이프라인 종료 시점에서 BASE에 대한 삽입 파이프라인을 수행하였던 종속 쌍 중에서 1단계 혹은 2단계의 시간 지연이 발생되어 있다. 따라서 삽입 파이프라인에서 발생된 시간 지연을 우선적으로 감소시킨다.

(f)는 (a)~(k)와 같은 방법으로 후속 기준쌍을 결정한 후 후속 기준쌍의 첫 명령어가 f에서 발생된 시간 지연 위치에 오도록 설정하면 된다.

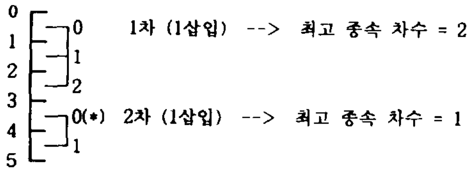
$N=c_0d_0+2c_1d_1+3d_2d_2+\cdots+(k+1)c_kd_k$에서 상수 $C_i=1(0\le i\le k)$인 경우의 파이프라인 수행 알고리즘은 다음과 같다. 단, 알고리즘에서 기준쌍에 대한 파이프라인 처리 함수인 pipeline()과 삽입 파이프라인 처리 함수인 in pipln()은 각 종속 명령어 마다 2단계의 시간 지연이 주어지도록 하여 (그림 7)에서와 같은 형태로 파이프라인 처리한다고 가정한다. 알고리즘에서의 입력 di는 일련의 종속 명령어 쌍들의 집합으로, 최고 종속 차수가 i라는 의미이다.

[알고리즘 4.1]

```
/* 입력: N=c0d0+2c1d1+3d2d2+···+(k+1)
       ckdk에 대한 종속 명령어 쌍의 집합
       (단, N>0, ci=1, 0≤i≤k)
   pipeline():종속 명령어 마다 2단계의 시간 지연
            을 둔 파이프라인 수행 함수
   in pipln():기준 종속쌍에 대한 삽입 파이프라인
            수행 함수
   delete():차수 집합(d Set)에서 파이프라인 완료
            된 종속 차수를 제거하는 함수
   d Set={0, 1, 2, . . . , k };N의 종속 차수집합
   b,f,s:기준 종속 쌍, 1 삽입, 2 삽입의 파이프라
         인 수행 완료된 최고 종속 차수 */
         b=f=s=0;
1단계. 파이프라인 기준 종속쌍과 그 차수를 결정.
      b<-- Max(d SET); /* 파이프라인 미처
      리 명령어 중 최고 종속 차수 결정 */
      BASE <-- db; /* 최고 차수 종속 명령
      어쌍을 BASE로 결정 */
2단계. 기준 종속쌍에 대한 파이프라인 수행.
      pipeline(BASE); /* 파이프라인 수행 */
      delete(d Set, b); /* 파이프라인 수행 완료
      된 종속쌍의 차수(b)를 제거 */
3단계. 기준 종속 쌍에 대한 삽입 파이프라인의 반
      복 수행.
      while (BASE is not completed insert-
      pipeline) do
       {flg <-- call COMPLETE KIND();
        /* 삽입 파이프라인의 수행 종류(그림
        9)*/case (flg)
          0:k <-- call IN PIPLN(1);
             /*1삽입 파이프라인 수행 */
            if (not first insert-pipline in
               BASE) then f=f+k+1
            else f=f+k;/*BASE에 대한 1삽
               입 파이프라인 수행 차수 계산 *
               /
            k <-- call IN PIPLN(2); /*
               2삽입 파이프라인 수행 */
            if (not first insert-pipline in
               BASE) then s= +k+1
            else s=s+k;/*BASE에 대한 2삽
               입 파이프라인 수행 차수 계산 *
               /
          1:k <-- call IN PIPLN(1);
             /* 1삽입 파이프라인 수행 */
            if (not first insert-pipline in
               BASE) then f= +k+1
            else f=f+k;/*BASE에 대한 1삽
               입 파이프라인 수행 차수 계산 *
               /
          2:k <-- call IN PIPLN(1);
             /* 2삽입 파이프라인 수행 */
            if (not first insert-pipline in
               BASE) then s=s+k+1
            else s=s+k;/*BASE에 대한 2삽
               입 파이프라인 수행차수계산 */
        endcase
       }
      endwhile
4단계. 새로운 기준 종속 명령어 쌍의 결정. /* 새
      기준쌍이 삽입 종속쌍에서 결정되지 않은 경
      우에만 BASE에 대한 파이프라인이 수행 */
       call NEW BASE(b,f,s);
5단계. STOP.
```

[알고리즘 4.1]의 3단계에서 삽입 파이프라인 수행 후, 삽입 파이프라인의 수행 차수 계산은 다음을 의미한다. 예를 들어 종속 차수 5인 BASE에 대해 최고 차수가 2인 1삽입 명령어 쌍으로 1차 삽입을 수행하면 f=2가 되며, 2차 삽입 수행한 명령어 쌍의 차수가 1이라 하면 최종 수행 결과는 다음과 같다.

종속 차수




따라서 BASE의 최고 차수 b는 5이지만, BASE에 대한 삽입 파이프라인의 차수는 첫번째 삽입 파이프라인에서는 원래의 종속쌍 차수가 그대로 적용되며 두번째부터는 최종 수행 결과의 종속 차수에 1을 더한 값이 해당 삽입 파이프라인의 최종 차수가 됨을 알 수 있다. 그것은 BASE에 대해 2차 이후의 삽입 파이프라인이 반복될 때마다 종속 명령어 내에 0차 명령어(*)가 존재하여 직전 종속 차수를 1씩 증가시키는 역할을 하기 때문이다. 따라서 1, 2 삽입 파이프라인이 반복될 때마다 해당 삽입 파이프라인이 완료된 상태에서의 최고 종속 차수를 의미하는 변수 f, s는 삽입 파이프라인 종속 차수에 1을 더하도록 하고 있다.

또한 알고리즘에서 사용하고 있는 서브 프로그램은 (1) COMPLETE KIND, (2) IN PIPLN, (3) NEW BASE의 세 개를 사용하고 있다. 알고리즘에서 호출하는 각각의 서브 프로그램 기능은 (1)은 삽입 파이프라인을 하나씩 처리한 후 변화된 상태에 따라 실행해야할 다음 삽입 파이프라인 수행 종류를 알려주며, (2)는 실제 삽입 파이프라인을 실행, (3)은 기준 종속쌍에 대한 파이프라인이 완료되었을 때, 다음 BASE가 될 기준 쌍을 설정하여 계속적인 파이프라인 처리를 수행하도록 하고 있다.

### 4.2 ci ≥ 2 (1≤i≤k)에 대한 파이프라인 알고리즘

수식 ②에서의 상수 ci ≥ 2 (0≤i≤k)인 k차 종속 명령어들에 대한 파이프라인 알고리즘 최고 종속 차수부터 내림차순으로 종속쌍 명령어들에 대한 파이프라인을 처리한다. 종속 차수가 2인 종속쌍이 4개가 있을 때, 동일한 종속 차수 내에서의 상수 표현과 의미는 다음과 같다.

$4d_2 = d_2 + d_2 + d_2 + d_2 +$ /*4개의 종속쌍*/

$= d^1_2 + d^2_2 + d^3_2 + d^4_2$ /*윗첨자:동일 차수 내에서의 종속쌍 표시 순번(상수)
아래첨자:종속차수*/

$= (d^1_{20} + d^1_{21} d^1_{22}) + (d^2_{20} + d^2_{21} d^2_{22}) + (d^3_{20} + d^3_{21} d^3_{22}) + (d^4_{20} + d^4_{21} d^4_{22})$

즉, $d^k_i$의 표현은 해당 명령어가 포함된 종속쌍의 최고 차수가 i이며, 차수가 i인 여러 개의 종속쌍들 중 k번째 종속쌍임을 의미한다. 또한 $d^k_{ij}$는 최고 종속 차수가 i인 종속쌍 내에서 해당 명령어의 차수는 j이며, 차수가 i인 다수 종속쌍 중에서 k번째 종속쌍에 해당됨을 나타낸다. 그 예로 $3d^1_{42}$는 특정 명령어에 대한 종속 차수가 1, 2, 3, 4의 관계를 가지는 종속 명령어 쌍이 3개가 있으며, 자신은 3개의 종속쌍 중 첫번째 종속쌍에 해당되는 2차 종속 명령어임을 의미한다.

$N = 3d_2 + 4 \cdot 2d_3$은 종속 차수가 2인 종속쌍이 3개, 차수 3인 종속쌍이 2개로 구성되어 전체 명령어 수 N=(3×1×3+4×2×4)=41이다. (조건2)에 의해 해당 차수의 상수가 3의 배수이면 완벽한 파이프라인이 되므로 본 알고리즘에서는 완벽한 파이프라인 되는 부분은 처리하지 않기로 한다. 이것은 주어진 전체 명령어에 대해서 최초의 파이프라인시에만 적용되며, 최고 종속 차수의 상수가 3의 배수이면 그 차수에 대한 처리는 완벽한 파이프라인으로 간주하여 그 다음 종속 차수를 대상으로 (조건2)를 적용한다. 그래서 내림차순 종속 차수에 따라 최초로 (조건2)를 만족하지 않는 상수가 나타나면 그 상수의 종속 차수가 최초의 파이프라인 대상으로 선정되어 알고리즘을 수행하는 방식을 취한다. 각 종속 명령어 쌍이 (조건2)에 대해 위배될 때, 그 차수에 있어서 파이프라인을 시작할 상수 순번은 다음 수식에 의해 결정한다.

$$c = c_K/3 \times 3 + 1 \qquad ③$$

그러므로 $2d_2 + 4d_3 + 6d_4$에 대한 첫 파이프라인은 종속 차수가 3인 종속쌍이며, 4개의 해당 종속쌍 중에서 상수 순번 1, 2, 3의 세 쌍까지는 완전 파이프라인이 되므로 네번째 종속쌍이 첫 파이프라인 대상으로 선택되어 [알고리즘 4.2]

에 따라 처리된다. [알고리즘 5.2]의 전체적인 수행 과정은 [알고리즘 4.1]과 동일하며, 동일 차수를 가지는 종속쌍들이 여러개 있으므로 파이프라인 수행은 고차 종속쌍부터 시작하는 동시에 같은 차수 내의 여러 쌍들에 대해서는 상수 순번에 따라 차례로 수행한다. 따라서 고차 종속쌍의 상수 횟수만큼 파이프라인이 먼저 수행된 후, 그 다음 차수의 종속쌍이 앞에서와 동일한 방법으로 파이프라인되는 것이다.

[정리1] 동일한 종속 차수를 가지는 종속 명령어 쌍의 갯수가 ck MOD 3=0이면, 그 종속 차 수의 명령어 쌍은 완전 파이프라인이 된다.

상수 $C_i \geq 2$인 종속쌍들에 대한 파이프라인에서 발생되는 상태에 대해 살펴보자. (조건2)에 따라 수식 ③을 적용하여 최초의 BASE가 선택되어 파이프라인 된 이후부터는 선택된 종속 차수의 상수만큼 파이프라인이 처리된 후에야 다음 종속 차수가 수행 대상으로 선정된다. 이 사항은 삽입 파이프라인을 수행하는 함수인 IN PIPLN ()에서 처리하며, 이것은 [알고리즘 4.1]에 이 부분만을 추가하여 수행하고 있다. 따라서 기준쌍에 대한 파이프라인 처리 함수(pipline())에서도 기준쌍 외에 현재 수행할 상수 순번과 입력된 수식에서의 상수를 매개 변수로 전달받아 처리하게 된다.

먼저 선택된 BASE에 대한 삽입 파이프라인이 완료되어 다음의 후속 기준쌍을 결정할 때에도 직전 파이프라인에서 사용되었던 상수와 상수 순번을 비교하여 해당 차수에 대한 처리가 모두 종료된 경우에만 새로운 종속 차수를 선택하며 그렇지 않은 경우에는 미처리된 직전 차수의 상수 순번이 남았을 경우에는 그 차수를 새로운 BASE로 선택하도록 한다. 삽입 파이프라인의 종류를 결정하는 COMPLETE KIND() 함수는 [알고리즘 4.1]에서 사용하던 것과 동일하다.

$N=c_0d_0+2c_1d_1+3c_2d_2\cdots+kc_kd_k$에 대한 종속명령어 쌍에 대한 파이프라인 수행 알고리즘은 다음과 같다.

[알고리즘 4.2]

```
/* 입력: N=c0d0+2c1d1+3d2d2+…+(k+1)ckdk
      에 대한 종속명령어 쌍의 집합
      (단, N>0, ci≥2, 0≤i≤k)
   d Set={0, 1, 2, . . . , k };종속 차수 집합
   pipeline(BASE, i, cb) BASE, 파이프라인 수행
      상수 순번, BASE의 상수
   i=0;동일 차수 내에서 파이프라인 수행할 상수
      순번
   c=0;현재 파이프라인 수행 중인 종속 차수
   b, f, s:기준 종속 쌍, 1삽입, 2삽입의 파이프라
      인 수행 완료된 최고 종속 차수 */
   b=f=s=0;
1단계. 최초의 파이프라인 기준쌍 선정 /* ci MOD
   3 ≠ 0인 상수를 가지는 최초의 종속
      차수 결정 */
   b <-- Max(d SET);
   if (cb MOD 3 ≠ 0) then BASE <---
   db; /* (조건2) 적용 */
   else goto 1단계;
2단계. BASE에 대한 파이프라인 수행.
   i=(cb/3)×3 + 1;/* 수식 ③ 적용 */
   pipeline(BASE, i, cb);/* 파이프라인 수행
   */
   if(i=cb);then delete(d Set, b);/* 최고 차
   수의 상수만큼 파이프라인
   elsei= i+1;수행된 종속쌍의 차수는 제거*/
3단계. 기준 종속 쌍에 대한 삽입 파이프라인의 반복
   수행.
   while (BASE is not completed insert-
      pipeline) do
    { flg <-- call COMPLETE KIND();
    /* 삽입 파이프라인의 수행 종류(그림 9)
    */case (flg)
      0:(i, k)<-- call IN PIPLN(1); /*
        1삽입 파이프라인 수행 */
        if (not first insert-pipline in
        BASE) then f=f+k+1
        else f=f+k; /* BASE에 대한 1삽
        입 파이프라인 수행 차수 계산 */
        (i, k) <-- call IN PIPLN(2);
        /* 2삽입 파이프라인 수행 */
        if (not first insert-pipline in
        BASE) then s=s+k+1
        else s=s+k; /* BASE에 대한 2삽
        입 파이프라인 수행 차수 계산 */
      1:(i, k) <-- call IN PIPLN(1); /
        * 1삽입 파이프라인 수행 */
        if (not first insert-pipline in
        BASE) then f=f+k+1
        else f=f+k; /* BASE에 대한 1삽
        입 파이프라인 수행 차수 계산 */
      2:(i, k) <-- call IN PIPLN(1);
        /* 2삽입 파이프라인 수행 */
        if (not first insert-pipline in
        BASE) then s=s+k+1
```

```
            else s= s+k;/* BASE에 대한 2삽
            입 파이프라인 수행 차수 계산 */
          endcase
        endwhile
4단계. 후속 기준쌍 결정 /*k:직전에 파이프라인 수
                  행한 종속 차수 */
      call NEW BASE(b, f, s, i, k);
5단계. STOP.
```

현재 삽입할 종속쌍의 상수와 수행할 상수 순번과의 비교에 의해 파이프라인할 대상이 결정되며, 각 종속쌍에 대한 수행은 [알고리즘 4.1]에서와 같이 종속 차수의 내림차순에 의한다.

[알고리즘 4.1]에서 사용되는 서브프로그램 IN PIPLN() 과의 차이점은 각 종속 차수마다 서로 다른 상수를 가지고 있으므로 종속 차수에 대한 상수(kind)를 삽입 파이프라인 종류(i)와 함께 매개변수를 전달하는 것이다. 처리 내용에서는 각 종속 차수의 상수 만큼 파이프라인이 완료되었는지 점검하는 부분이 추가된다.

서브프로그램 NEW BASE()와 [알고리즘 4.1]의 NEW BASE() 가 서로 다른점은 수행 대상 종속 차수(k)와 그 종속 차수 k의 상수(i)를 매개변수로 전달한다는 것이다. 전달된 두 매개변수 k, i는 후속 기준쌍을 결정할때, 직전의 파이프라인시 사용된 종속 차수와 그 상수만큼 처리되었을 경우는 다음 종속 차수를 후속 기준쌍으로 결정하며, 그렇지 않을 경우도 새로운 종속 차수를 지정할 필요가 없게 된다. (그림 7)의 (a)~(c)의 경우에만 이 결정을 하기위한 처리 과정이 추가되면 된다.

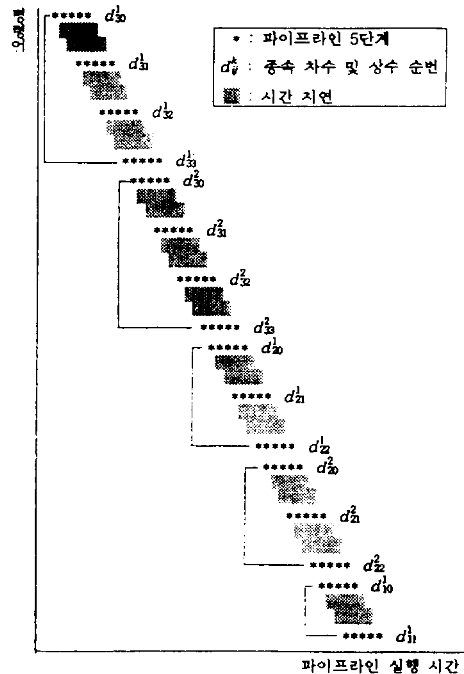
그러면 다음 수식을 [알고리즘 4.1]의 수행 절차에 따라 파이프라인하여 보자. 최초의 파이프라인 대상은 (조건2)에 의해 종속 차수 6은 완전 파이프라인으로 인정되고, 종속 차수 4가 (조건 2)에 위배되므로 수식 ③에 의해 상수 순번 4부터 파이프라인이 시작된다.

## 5. 제안된 알고리즘의 분석

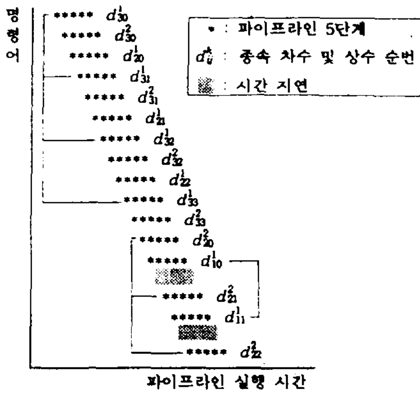
파이프라인에서 발생되는 시간 지연은 각 명령어들의 종속성에 기인한다. 기존의 5 단계 파이프라인 방식에서는 특정 종속 명령어 쌍들에서 발생되는 시간 지연을 감소시킬 수 있는 방법으로 (그림 4, 5)와 같은 방법을 제시하고 있으나 이 방식에 의해서는 시간 지연의 최소화를 도모할 수 없다. 그 이유는 단지 명령어의 실행 시간을 지연하는 것은 전체 프로그램의 실행 소요 시간의 증가를 가져올 뿐이며, 종속된 명령어 사이에 종속성이 없는 독립된 명령을 삽입하는 것은 시간 지연 발생 수만큼 독립 명령어가 없을 경우 그 효과를 기대하기 어렵기 때문이다.

그러나 제안된 알고리즘에 의하면 종속 관계에 있는 명령어 쌍들이 다른 종속쌍과의 파이프라인에서 각 명령어 자체의 파이프라인 일련 순서에 의해서 상호간의 시간 지연을 서로 감소시켜줄 뿐 아니라 종속쌍의 명령어들을 순차적으로 파이프라인할 때보다 훨씬 짧은 실행 시간에 처리될 수 있다. (그림 8)은 전체 명령어가 $N=d_1+2d_2+2d_3$ 일 때 기존의 파이프라인 방식과 제안된 알고리즘과의 성능과 효율성을 비교 및 검토한 것이다.

(그림 8)에서 (a)는 일반 파이프라인에서 종속 관계의 명령어 쌍들을 순차적으로 처리하는




(a) 종속쌍의 일차원 순서 배열에 따른 파이프라인
(a) General pipeline in one-dimensions sequence of subordinary pairs.




(b) 종속 차수를 고려한 파이프라인
(b) Pipeline with subordinary degree

(그림 8) 알고리즘 성능 비교
(Fig. 8) Comparison with algorithm performance

경우이다. 이 경우 발생되는 시간 지연은 최악의 경우로 종속 관계 명령어 마다 2개의 시간 지연이 발생되므로 전체 22개의 시간 지연이 발생된다. 전체 명령어의 실행 시간은 한 명령어가 5단계로 실행되므로 전체 명령어 수가 n일 때, T=5+(n−1)이 된다. 이때의 n은 시간 지연까지 포함시켜야 한다. 각 종속쌍 안에서 발생되는 시간 지연은 (종속차수×2)이다.

(그림 8)의 (a), (b) 비교에 의해 제안한 알고리즘에 의한 시간 지연의 최소화와 이에 따른 실행 시간의 단축은 종속 관계의 명령어들을 순차적으로 파이프라인하였을 때에 비해 훨씬 효율적임을 알 수 있다. 그리고 종속 관계의 명령어 사이에 다른 명령어를 삽입하는 파이프라인 방식에 비해서도 각 명령어의 파이프라인 수행 순서와 삽입에 사용할 독립 명령어의 갯수와의 함수 관계에 의하면 시간 지연과 실행 시간의 감소에 대한 정량적인 차이를 제시하지는 못했으나 파이프라인 처리 방식에 의해 이미 기존의 방식보다 더 나은 결과를 가져올 수 있음을 알 수 있다.

[VLIW 시스템의 병렬 파이프라인의 예]

다음 프로그램을 예로 VLIW 시스템 프로세서에서의 병렬 파이프라인 처리를 기존 알고리즘과 비교하여 보자.

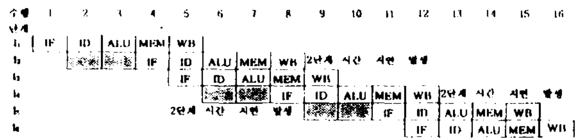
〈예제 프로그램〉

$I_1$:R3:=R1 op R2
$I_2$:R4:=R3+1
$I_3$:R7:=R5 op R6
$I_4$:R8:=R7+1
$I_5$:R9:=R8+1
$I_6$:R12:=R10 op R11

여기서 명령어 $I_2$는 $I_1$에 종속적이고, $I_5$는 $I_4$, $I_3$에 종속적임을 알 수 있다.

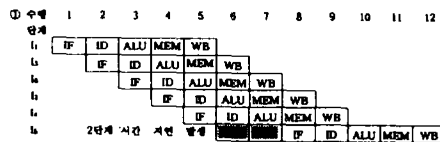
1) 명령어 기술 순서에 따른 파이프라인

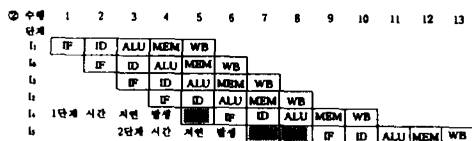
명령어를 기술한 순서대로 파이프라인하면 전체 6단계의 시간 지연이 발생하여 16단계에 이르러 완료된다.




2) 명령어 실행 순서 변경에 의한 파이프라인

(그림 5)의 파이프라인 방식과 같이 명령어 실행 순서를 변경하여 파이프라인 하는 경우에 있어서는 명령어를 처리하는 순서 배열에 따라 시간 지연이 다양하게 발생할 수 있다. 예를 들면, 명령어 처리 순서를 ①과 같이 하면 I4와 I5 사이에 삽입할 독립 명령어가 없으므로 2단계 시간 지연이 발생하여 총 12단계로 처리될 수 있다. 또한 ②의 순서로 명령어를 처리하면 3단계의 시간 지연이 발생하여 총 13단계에 완료된다. 따라서, 이 방식은 파이프라인을 배열하는 순서에 따라 시간 지연이 가변적으로 변한다.

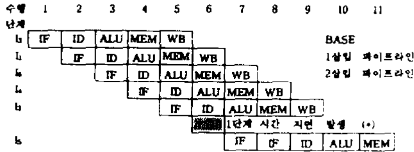



3) 제안한 알고리즘

[알고리즘 4.2]에 의해 종속 명령어 쌍과 차수 집합 및 기준쌍을 먼저 선정하면 다음과 같다.

① 종속 명령어 쌍:(I1, I2), (I3, I4, I5), (I6)
② 차수 집합(d SET):(1, 2, 0)
③ 최초의 BASE = (I3 I4 I5), 1삽입 명령어 쌍=(I1, I2), 2삽입 명령어=(I6)




명령어 기술 순서에 따른 파이프라인에서는 6단계 시간 지연이 발생하여 총 16단계만에 수행이 완료되었으나 제안한 알고리즘에서는 1단계의 시간 지연만이 발생되어 전체 11단계에 파이프라인이 완료될 수 있다.

따라서, 제안한 알고리즘과 같이 종속차수를 고려하여 파이프라인할 경우에는 최소한의 시간 지연으로 파이프라인이 완료될 수 있음을 알 수 있다.

[정리2] $N = c_0 d_0 + 2c_1 d_1 + 3d_2 d_2 + \cdots + (k+1) c_k d_k$일때, 종속 차수의 내림차순에 의한 명령어 순서 배치로 최소 시간 지연으로 파이프라인된다.

〈증 명〉 파이프라인에서 발생되는 시간 지연은 종속 명령어 쌍에 의존한다. 그러므로 $N = c_0 d_0 + 2c_1 d_1 + 3d_2 d_2 + \cdots + (k+1) c_k d_k \cdots$의 최고 종속 차수가 k인 종속쌍 dk에서만 2k개의 시간 지연이 발생된다. 따라서 명령어들의 파이프라인 순서 배치에 따라 나머지 종속쌍에서 발생되는 시간 지연를 포함하면 전체 시간 지연 D=2k+가 된다. 가장 긴 종속쌍을 거느리는 명령어(dk)를 먼저 파이프라인하면 여기서 발생된 2k개의 시간 지연 위치에 i인 종속쌍을 dk와 동일한 파이프라인 방식으로 순서대로 배치하면 시간 지연 $D=(2k+\alpha)-2i$가 된다. 즉 di에서 발생된 시간 지연 위치에는 di의 명령어들이 위치하고, dk에서 발생된 시간 지연 위치에는 di의 각 명령어들이 있으므로 이중의 시간 지연 감소 효과가 오게 된다. 미해결된 dk의 시간 지연도 다른 종속쌍을 선택하여 같은 방법을 적용하면 마지막 단계에는 더 이상 명령어 삽입으로 시간 지연을 줄일 수 없는 부분만이 남게 되며, 이것은 종속 관계를 가지는 명령어들에 의한 최소 시간 지연으로 볼 수 있다.

## 7. 결 론

지금까지 파이프라인 구조와 VLIW 시스템의 프로세서에서 시간 지연 최소화를 포함한 설계와 해결 방안에 대해 기술하였다. 본 논문은 보다 효율적으로 시간 지연을 최소화할 수 있는 병렬 파이프라인 알고리즘을 제안하였으며, 이에 대한 시간 지연 발생 문제에 대해 연구하였다. 물론 기존의 시스템에서 설계된 파이프라인 구조가 효용성이 없는 것은 아니지만 제안한 알고리즘에 의하여 기존의 문제점을 훨씬 간소화시킬 수 있으며, 프로세서의 유휴 시간을 줄일 수 있다.

제안한 병렬 파이프라인 알고리즘은 각 명령어간의 종속 관계를 유지하면서 파이프라인 시에 복잡한 문제를 야기하는 여러 종속 명령어 쌍들의 단점을 역이용하고 있다. 또한 각 명령어 간의 종속성과 종속쌍들의 수에 따라 유휴 시간이 전혀없이 파이프라인할 수도 있으며, 처리 전에 미리 완벽한 파이프라인이 가능한지 여부를 밝힐 수도 있다.

본 논문은 프로그램 상에서 발생될 수 있는 시간 지연에 관한 문제를 최소화할 수 있는 병렬 알고리즘을 제시하여 기존 파이프라인 방식과의 비교·검토를 수행하였다. 이것은 명령어 수준의 시간 지연 최소화 기법이며, 향후 대규모 응용 프로그램의 병렬 처리를 위한 VLIW 프로세서에서의 광역 스케쥴링 알고리즘이나 VLIW 기반의 다중프로세서 시스템의 설계 등에 관한 연구 개발로 고속의 효율적인 파이프라인 구조와 프로세서 최적화에 대한 연구가 요구된다.

## 참 고 문 헌

[1] S. Wallace, N. Bagherzadeh, "Performance Issues of a Superscalar Microproc-

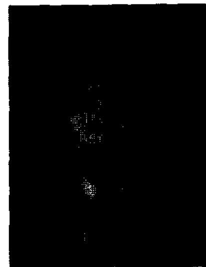
essor", Int. Conf. on Parallel Processing, pp. 293-297, 1994.
[ 2 ] K. Hwang, "Advanced Computer Architecture", McGraw-Hill, 1991.
[ 3 ] M. Annaratone, E. Arnould, T. Gross, H. T. Kung, M. Lam, O. Manzilicíloglu, J. A. Webb, "The Warp Computer: Architecture, Implementation, and Performance", IEEE Trans. on Software Engineering, Vol. 14, No. 5, pp. 1523-1537, December 1987.
[ 4 ] R. Duncan, "A Survey of Parallel Computer Architecture", Computer, pp. 5-16, Feb. 1990.
[ 5 ] M. J. Irwin, "A Digit Pipelined Dynamic Time Warp Processor", IEEE Trans. on ASSP, pp. 1412-1422, September 1988.
[ 6 ] A. Capitanio, N. Dutt, A. Nicolau, "Partitioning of Variables for Multi-Register-File VLIW Architectures", Int. Conf. on Parallel Processing, pp. 298-301, 1994.
[ 7 ] C. L. Su, A. M. Despain, "Branch With Masked Squashing in Superpipeline Processor", in Proc. of the 21st Annual International Symposium on Computer Architecture, pp. 130-140, April 1994.
[ 8 ] S. K. Chen, W. K. Fuchs, W. W. Hwu, "An Analytical Approach to Scheduling Code for Superscalar and VLIW Architectures", Int. Conf. on Parallel Processing, pp. 285-292, 1994.
[ 9 ] 서장원, 방혜자, 전문석, 이철희, "VLIW시스템에서의 파이프라인 구조와 바이패싱 기법", 가을 정보과학회 발표논문집, Vol. 21, No. 2, 883-886, 1994.

**서 장 원**

1992년 서울산업대학교 전산과(학사)
1993년~현재 숭실대학교 대학원 전산과 석사과정
관심분야: 병렬컴퓨터 구조, 프로세서 설계, 병렬처리 이론

**송 진 희**

1988년 서울산업대학교 전산과(학사)
1990년 한국외국어대학교 대학원 전산과(석사)
1994년~현재 숭실대학교 대학원 전산과 박사과정
1992년~현재 신흥전문대학 전산정보처리과 조교수
관심분야: 병렬 알고리즘 계산, 기하학, 그래픽스 이론



**류 천 열**

1979년 숭실대학교 전자계산학과(학사)
1985년 숭실대학교 산업대학교 전자계산학과(석사)
1995년 숭실대학교 대학원 박사과정중
1986년~현재 유한전문대학 전자계산학과 부교수
관심분야: 프로그래밍언어, 시스템소프트웨어, 데이타통신

**전 문 석**

1980년 숭실대학교 전산과(학사)
1986년 Univ. of Maryland 전산과(석사)
1988년 Univ. of Maryland 전산과(박사)
1989년 Morgan State Univ. 전산수학과 조교수
1991년 New Maxico State Univ. 부설 Physical Science Lab. 책임연구원
1991년~현재 숭실대학교 컴퓨터학부 부교수
관심분야: 병렬 알고리즘, 병렬컴퓨터 구조, 대규모 집적회로, 병렬처리 이론, 폴트톨로런스