논문 2007-44SD-1-8

비동기식 임베디드 프로세서를 위한 적응형 파이프라인 구조

(Adaptive Pipeline Architecture for an Asynchronous Embedded Processor)

이 승 숙*, 이 제 훈**, 임 영 일***, 조 경 록*

(Seung-Sook Lee, Je-Hoon Lee, Young-IL Lim, and Kyoung-Rok Cho)

요 약

본 논문은 비동기식 프로세서에서 동작 상황에 따라 파이프라인 구조가 변경 가능하고 명령어 종류에 따라 병렬처리를 지원하는 적응형 파이프라인 구조를 제안하였다. 제안된 구조는 동작이 불필요한 스테이지를 건너뛰는 스테이지 스키핑(stage-skipping)과 다음 스테이지가 비어 있으면 현재 스테이지와 다음 스테이지를 하나로 통합하는 스테이지 통합(stage-combining) 기법을 지원한다. 이 기법들은 명령어 종류에 따라 서로 다른 데이터패스를 사용하는 명령어들을 병렬로 처리하여 머신 사이클을 단축시켜 프로세서의 동작 속도를 증가시킨다. 본 논문에서는 제안된 파이프라인 구조를 적용한 ARM 명령어 호환 프로세서를 설계하였다. 이 프로세서는 VHDL로 설계한 후 0.35-㎛ CMOS 표준 셀 라이브러리를 이용하여 합성되었다. SPEC2000 벤치마크를 사용하여 성능을 평가한 결과, 타겟 프로세서는 평균 365 MIPS의 속도로 동작하여 영국 맨체스터 대학에서 개발한 비동기 프로세서인 AMULET3i에 비해 2.3배 높은 성능을 보였다. 제안된 파이프라인 기법과 프로세서 구조는 고속 비동기식 프로세서 설계에 적용 가능하다.

Abstract

This paper presented an adaptive pipeline architecture for a high-performance and low-power asynchronous processor. The proposed pipeline architecture employed a stage-skipping and a stage-combining scheme. The stage-skipping scheme can skip the operation of a bubble stage that is not used pipeline stage in an instruction execution. In the stage-combining scheme, two consecutive stages can be joined to form one stage if the latter stage is empty. The proposed pipeline architecture could reduce the processing time and power consumption. The proposed architecture supports multi-processing in the EX stage that executes parallel 4 instructions. We designed an asynchronous microprocessor to estimate the efficiency of the proposed pipeline architecture that was synthesized to a gate level design using a 0.35μm CMOS standard cell library. We evaluated the performance of the target processor using SPEC2000 benchmark programs. The proposed architecture showed about 2.3 times higher speed than the asynchronous counterpart, AMULET3i. As a result, the proposed pipeline schemes and architecture can be used for asynchronous high-speed processor design

Keywords : Asynchronous processor architecture, Asynchronous pipeline, Multi-processing

I. 서 론

VLSI 설계 기술의 발달은 하나의 칩 안에 집적되는 시스템의 크기와 성능을 증가시키고 있다. 대부분의 SoC 칩들은 하드웨어와 소프트웨어의 통합 설계를 지원하기 위해 하나 이상의 임베디드 프로세서를 포함한다. 하지만 임베디드 프로세서는 복잡한 동작을 처리하여 많은 전력을 소모하기 때문에 프로세서의 소비 전력을 줄이는 것은 SoC 설계에서 중요한 항목이다. 이러한 필요성에 의해, 비동기식 프로세서 설계 기법에 관한 연구가 활발히 이루어지고 있다.

* 정회원, *** 학생회원, 충북대학교 정보통신공학과 컴퓨터정보통신연구소
(Dept. of Computer and Communication Engineering and Research Institute for Computer and Information Communication, Chungbuk National University)
** 정회원, 충북대학교 BK21 계약교수
(CBNU BK21 Chungbuk Information Technology Center, Chungbuk National University)
※ 이 논문은 2006년도 교육인적자원부 지방연구중심대학 육성사업의 지원에 의하여 연구되었음.
접수일자: 2006년9월28일, 수정완료일: 2006년12월22일

프로세서를 사용하는 시스템 측면에서 동기식과 비동기식으로 구분해 볼 때, 동기식 시스템은 시스템 클록에 의해 전체 시스템을 구동시키는 반면에, 비동기 시스템은 데이터 의존형(data-dependent) 제어 방식으로 데이터의 흐름에 따라 선택적으로 동작이 필요한 모듈만을 구동시킨다. 따라서 비동기 시스템은 동작이 필요없는 모듈들이 휴지상태에 있을 때 에너지를 소모하지 않는 이점을 갖는다. 기존 연구 결과에서 비동기 설계는 적은 전력 소모로 인해 서브 마이크로 시스템 설계 방법 중의 하나로 인식되고 있다.

현재 ARM프로세서에 대응되는 영국 맨체스터 대학의 AMULET3, 동경 공업 대학의 TITAC-20 32비트 프로세서 등 많은 비동기식 RISC 마이크로프로세서가 구현되어 평가되었다[1-4]. AMULET3i의 경우 동기식 ARM7과 동일한 공정 및 구조를 가짐에도 불구하고 ARM9과 비슷한 성능을 보였다. 최근에는 Lutonium이나 A8051과 같은 CISC (Complex Instruction Set Computer) 타입의 비동기 프로세서도 개발되었고, 동기식 8051에 비해 높은 성능을 보였다[5-6].

대부분의 고성능 프로세서들은 성능 향상을 위해 파이프라인 구조를 채택한다. 이 때 각 파이프라인 스테이지의 동작 시간은 실행중인 명령어의 종류에 따라 가변적이다[7]. 파이프라인 방식이 적용된 동기식 시스템은 가장 긴 지연 시간을 갖는 블록에 의해 성능이 제한되므로 가장 긴 실행 시간을 갖는 명령어는 전체 시스템의 성능을 감소시킨다. 반면에 비동기 시스템은 클록을 사용하지 않고 비동기 핸드셰이킹 프로토콜을 사용하기 때문에 이와 같은 문제가 발생하지 않아 평균 지연 시간으로 동작한다.

프로세서의 파이프라인 구조가 실행중인 명령어의 종류와 프로세서의 동작 상태에 따라 가변적일 경우 프로세서의 성능은 향상될 수 있다. 그러나 동기식 파이프라인 시스템은 고정된 파이프라인의 길이와 위치를 갖기 때문에 가변적인 동작에 대한 대응이 어렵다. 이를 극복하기 위해 Efthymiou와 Garside는 새로운 래치 컨트롤러를 채택한 비동기 파이프라인 구조를 제안하였다. Efthymiou가 제안한 래치 컨트롤러는 데이터 저장이 필요하면 래치를 사용하고 데이터 저장이 필요 없으면 래치가 없는 것처럼 동작시킨다. 래치가 없는 것처럼 동작될 경우 래치 전후의 두 스테이지들은 하나의 스테이지로 통합된다. 따라서 시스템의 동작에 따라 래치의 동작 여부를 제어하여 파이프라인 구조를 유연하게 변화시킬 수 있다[8].

본 논문은 데이터 의존형 제어에 의한 새로운 비동기식 파이프라인 구조를 제안한다. 제안된 파이프라인 구조는 스테이지 스키핑과 스테이지 통합 기법을 지원한다. 스테이지 스키핑은 불필요한 파이프라인 스테이지의 동작을 제거한다. 스테이지 통합은 다음 스테이지가 빈(empty) 스테이지 일 때, 현재 스테이지와 다음 스테이지를 하나의 스테이지로 통합한다. 따라서 두 기법은 프로세서의 실행 데이터 패스의 길이를 줄여 시스템 성능을 향상 시킨다. 또한 본 논문은 제안한 파이프라인 구조의 연산 블록에 대한 비동기 멀티 프로세싱을 제안한다. 만약 데이터 패스가 명령어의 종류에 따라 구분된다면 몇몇의 명령어들은 하드웨어 크기의 증가 없이 병렬로 실행될 수 있다. 제안된 멀티 프로세싱은 프로세서의 전체 데이터 패스를 각각의 독립적인 데이터 패스로 분할하는 구조를 가졌다. 제안된 구조의 성능을 평가하기 위해 ARM9 호환 프로세서를 설계하였다.

본 논문의 Ⅱ장에서는 제안된 적응형 파이프라인 구조와 비동기 멀티 프로세싱에 대해 기술하였으며, Ⅲ장에서는 제안된 컨트롤 기법을 사용한 비동기식 ARM프로세서의 성능 평가 방법에 대해 기술하였다. Ⅳ장은 SPEC2000 벤치마크 프로그램을 이용한 프로세서 성능 평가를 수행하였고, 마지막으로 Ⅴ장에서 결론을 맺는다.

Ⅱ. 제안된 적응형 파이프라인 구조 및 멀티 프로세싱 방법

본 장에서는 파이프라인 스테이지 스키핑(skipping)과 스테이지 통합 방법이 적용된 새로운 비동기식 적응형 파이프라인 구조에 대해 기술한다. 또한 명령어를 종류에 따라 서로 다른 데이터 패스를 사용하도록 하여 병렬로 처리하는 멀티 프로세싱 방법을 설명한다.

1. 스테이지 스키핑 및 스테이지 통합

일반적으로 프로세서는 동작의 규칙성을 유지하기 위해 실행이 불필요한 버블(bubble) 스테이지를 포함한다. 버블 스테이지가 발생되었을 때 이 스테이지 동작을 건너뛰거나, 다음에 실행할 스테이지가 휴지 상태일 때 현재 스테이지 동작 후 데이터를 래치하지 않고 즉시 다음 스테이지로 데이터를 전송한다면 동작 시간을 줄일 수 있다. 그림 1은 제안된 스테이지 스키핑과 스테이지 통합 방식을 채택한 파이프라인 구조를 나타낸다. 이 파이프라인 구조는 각각의 래치 컨트롤러와 파

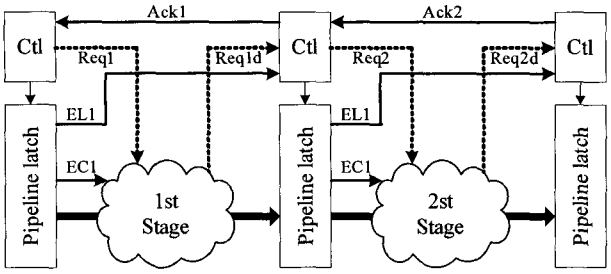

그림 1. 제안된 파이프라인 구조
Fig. 1. Proposed pipeline architecture.

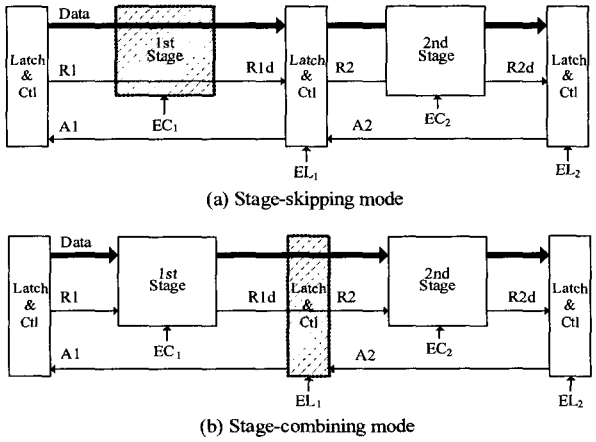

그림 2. 제안된 파이프라인 실행 예
Fig. 2. Examples of the proposed pipeline operation.

이프라인 스테이지에 ECN과 ELN과 같은 제어 신호를 포함하고있다. ECN 신호는 해당하는 스테이지의 동작을 수행할 지 아니면 그대로 통과할 지를 결정한다. ECN 신호가 high인 경우 N번째 스테이지는 동작하지 않고 데이터가 다음 래치로 전달되며, low인 경우 N번째스테이지는 정상적으로 동작한다. ELN 신호는 래치를 정상적으로 동작시킬 지 아니면 존재하지 않는 것처럼 동작시킬 지를 결정한다. ELN 신호가 high이면 N번째 래치는 존재하지 않는 것처럼 동작하고, low이면 정상적인 래치 동작을 수행한다. 이 신호들은 그림 1에서 나타낸 것처럼 데이터가 해당 스테이지와 래치에 도착할 때까지 유효하다.

프로세서는 명령어를 instruction decode(ID)에서 디코딩한 이후에 명령어 수행에 불필요한 버블 스테이지와 현재 사용 가능한 스테이지를 파악한다. 즉, ID 스테이지는 마이크로 명령어와 ELN, ECN 신호를 다음 스테이지로 전달한다. 스테이지 스키핑 모드에서는 버블 스테이지 동작을 제거한다. 각 스테이지는 앞부분에 MUX를 포함하고 있다. 그림 2(a)의 EC1신호는 MUX로 입력되어 첫 번째 스테이지의 동작을 생략할 지 여

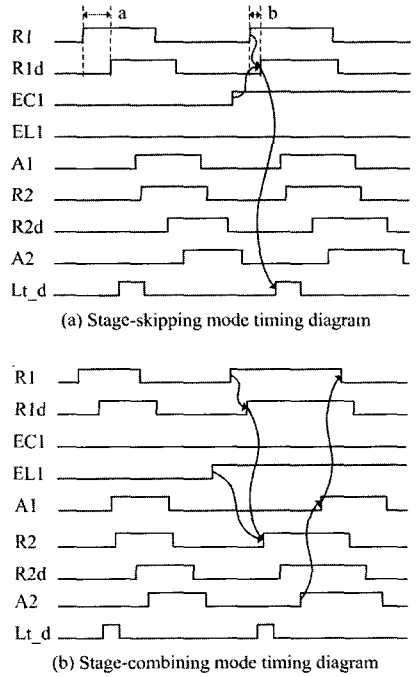

그림 3. 파이프라인 실행 타이밍 다이어그램
Fig. 3. Timing of the pipeline operation.

부를 결정한다. 첫 번째 파이프라인 스테이지의 조합회로는 EC1 신호가 low일 때 동작이 수행된다. 반대로 이 신호가 high일 때에는 조합회로는 연산을 수행하지 않고 입력을 바로 다음 단으로 통과시킨다.

스테이지 통합에서는 두 개 이상의 연속된 파이프라인 스테이지가 하나의 스테이지로 결합된다. 그림 2(b)에서 볼 수 있듯이 첫 번째와 두 번째 스테이지 사이의 래치는 EL1신호가 high일 때 열린 상태(transparent)가 된다. 이때 래치는 데이터 저장을 하지 않는다. 래치가 열린 상태가 되기 위해서는 다음 스테이지의 연산이 모두 끝나있어야 하고, 다음 스테이지를 위해 준비 상태에 있어야 한다. 반대로 EL1 신호가 1일 때에는 래치는 정상적인 동작을 한다. 그림 1과 같이 각 스테이지의 ELN신호는 ID 스테이지에서 발생되고 다음 목적지 래치에 도달할 때까지 각 스테이지에서 다음 스테이지로 전달된다.

그림 3(a)는 스테이지 스키핑 동작의 타이밍 다이어그램을 나타낸다. EC1신호는 첫 번째 스테이지의 동작 여부를 결정한다. 첫 번째 스테이지는 EC1신호가 high일 때 동작하지 않는 즉, 없는 스테이지라고 가정된다. 그림 3(a)의 시간 구간 a와 b는 스테이지 스키핑 모드에서 파이프라인의 실행시간이 스테이지가 실행되지 않는 만큼 감소된다는 것을 나타낸다. 즉, R1↑ → R1d↑가 일어난다는 것은 원래 R1↑ → 1st Stage →

R1d↑에서 1st 스테이지의 실행시간이 생략됨을 나타내고 있다.

그림 3(b)에 나타나듯이 스테이지 통합 모드는 EL1 신호가 high가 되어 두 개의 연속적인 파이프라인 스테이지가 한 개의 스테이지로 통합되어 동작한다. 첫 번째 스테이지의 수행 결과는 래치에 저장되지 않고 다음 스테이지로 전달된다. 즉, 요청 신호인 R1d 신호는 래치를 통과하여 두 번째 스테이지로 바로 전달된다. EL1신호에 의해서 두 스테이지가 통합된 이후에 첫 번째 스테이지는 동작을 끝낸 이후에도 확인(acknowledge) 신호 A1을 받지 못하며, 두 번째 스테이지의 동작이 끝난 이후에 A2↑ → A1↑으로 확인 신호가 전달된다.

2. 데이터패스 분리에 의한 멀티 프로세싱

대부분의 프로세서들은 스테이지마다 한 가지 작업을 하는 파이프라인 구조를 채택한다. 그림 4(a)에서 보는 것과 같이 각 명령어는 파이프라인 스테이지 수에 따라 같은 양의 작업을 병렬로 수행한다. 반대로 몇몇 프로세서들은 슈퍼스칼라나 VLIW 프로세서와 같이 병렬 처리를 위해 기능 블록을 중첩하는 방법을 사용한다. 중첩된 기능 블록들은 데이터 패스의 수만큼 같은 연산을 동시에 실행할 수 있다. 그러나 이러한 프로세서들은 반드시 하드웨어 크기가 증가한다는 단점을 갖는다.

실제로 모든 명령어들이 반드시 같은 데이터 패스로 통과하는 것은 아니다. 예를 들어, ADD R1 R2 R3 명령과 MOV R3 R4와 같은 명령은 각기 다른 데이터 패스를 갖는다. ADD 명령어는 ALU연산을 수행하고, 결과 값을 레지스터 파일에 저장하는 과정을 필요로 하지만, MOV 명령어는 레지스터와 레지스터 사이에서 데이터를 전달한다. 이 두 명령어들은 서로 독립적인 데이터 패스와 자원(resource)을 요구하므로 병렬로 처리될 수 있다. 동기식 프로세서는 병렬처리를 위해 데이터패스를 분리하려면 복잡한 컨트롤 메커니즘을 요구한다. 그러나 비동기식 프로세서에서는 단순하게 핸드셰이킹 제어로 병렬처리가 가능하다. 만약 프로세서가 서로 간에 의존성이 없는 명령어를 동시에 실행시킨다면 그림 4(b)의 기능 블록들의 중첩이 없이 동작할 수 있으므로 프로세서의 성능은 크게 향상될 것이다.

비동기식 설계 방법을 사용하면 데이터 의존 컨트롤에 기반을 둔 고유한 멀티 프로세싱이 가능하다. 이 방법은 실행되고 있는 명령어들이 각기 다른 데이터패스를 가질 경우 데이터 패스 분리에 의한 병렬 처리에 적

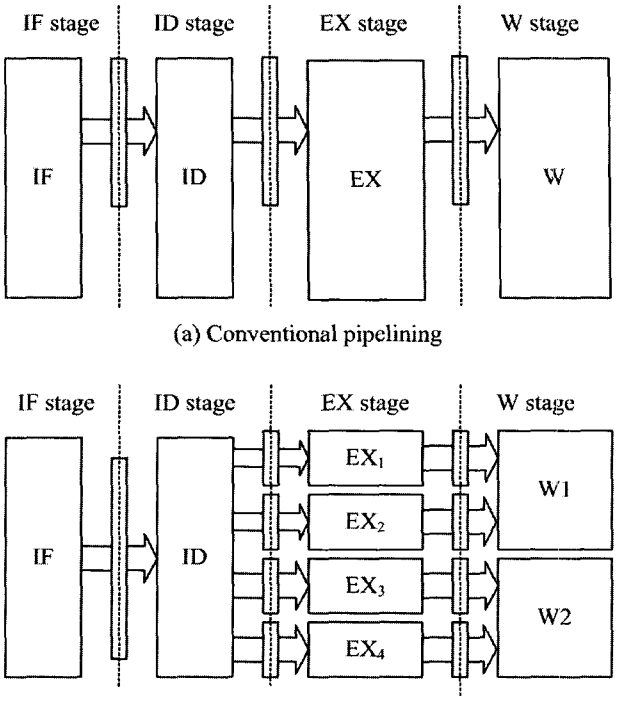

그림 4. 일반적인 파이프라인 구조와 데이터패스 분리 기반의 제안된 파이프라인 구조

Fig. 4. The comparison of the conventional pipeline architecture and the proposed pipeline architecture based on a datapath splitting.

합하다. 이러한 컨트롤 방법은 명령어의 종류에 따라 전체 시스템을 여러 개의 데이터 패스로 나눈다. 그림 4(b)는 데이터패스 분리에 의한 멀티 프로세싱의 예를 보여준다.

데이터패스 분리 방법을 채택한 멀티 프로세싱 구조는 그림 5에서 표시한 복합적인 포크(fork)와 조인(join)을 요구한다. 포크는 해당 스테이지의 입력을 복제하여 분리된 패스로 분배한다. 조인은 여러 패스에서 발생한 명령어들을 하나의 패스로 연결시킨다. ID 스테이지에서 디코딩된 명령어는 포크에 의해서 명령어별로 동시에 해당 EX 스테이지로 전달된다. EX 스테이지의 동작이 완료된 후에는 메모리나 레지스터 파일로의 저장을 위해 조인 동작을 수행한다.

제안된 구조는 그림 5의 페트리 넷(Petri-Net)으로 표현될 수 있다. 페트리 넷에서는 시스템의 동작이 신호 천이의 점화(firing)로 나타내어진다. 점화는 시스템의 동작을 표현하는 넷의 토큰(token) 분포를 변화시킨다. 순차적인 점화(sequentially firing)를 표현하기 위해서 '*'를, 동시 발생(concurrency)을 표현하기 위해서는 '|'를 이용한다[9, 10]. IF 스테이지에서 ID 스테이지까지는 순차적으로 실행된다. ID 스테이지에서 발생하는 포크

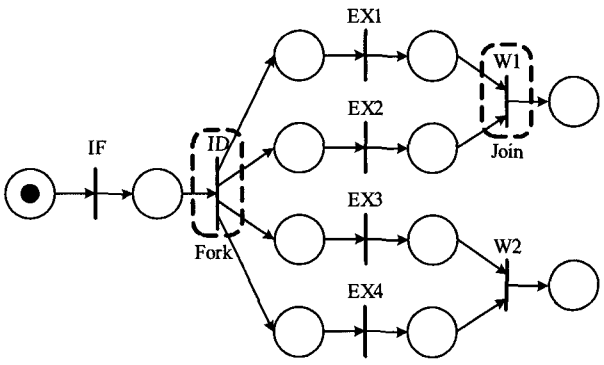

그림 5. 제안된 파이프라인 방식을 위한 페트리 넷
Fig. 5. Petri-Net diagram for the proposed pipelining.

동작은 토큰을 동시에 4개의 EX 스테이지 패스로 전달한다. EX1과 EX2 스테이지의 결과는 W1 스테이지에서 조인되고, EX3와 EX4 스테이지의 결과는 W2 스테이지에서 조인된다. 따라서 이 동작은 아래의 식 (1)과 같이 표현된다.

$$\text{Parallel Processing} = IF * ID * [(EX1 \mid EX2) * W1] \mid [(EX3 \mid EX4) * W2] \quad (1)$$

III. ARM 구조 마이크로프로세서에 응용

제안된 파이프라인 구조와 멀티 프로세싱 방법의 성능 평가를 위해 ARM9 명령어 호환 프로세서를 설계하였다. 프로세서는 그림 6과 같이 IF, thumb decoder, ID1, ID2, operand fetch(OF), execute(EX), reorder buffer(RB) 그리고 write back(WB)의 8개 파이프라인 스테이지로 구성되었다. 추가적으로 데이터 메모리와 프로그램 메모리가 존재하며, 각 스테이지 사이에는 래치 및 래치 컨트롤 블록이 위치한다.

동기식 ARM9 구조와 비교하면 제안된 구조에는 ID2와 RB 두 개의 스테이지가 추가되었다. ID2 스테이지는 데이터 의존성과 명령어 종류를 검사한 후 동시에 실행될 수 있는 명령어들을 선택하여 다음 스테이지로 전달한다. RB 스테이지에서는 병렬처리로 인해 순서가 바뀌어 실행된 명령어 실행 결과를 메모리에서 패치한 순서대로 재배치하고 레지스터에 저장한다.

IF 스테이지는 한 번에 4개의 명령어를 프로그램 메모리에서 패치한다. 패치된 4개의 명령어들이 ID1 블록을 통해 병렬로 디코딩된 후 다음 스테이지인 ID2 블록에서 명령어들 간의 의존성을 검사한다. ID2 스테이지는 크게 인스트럭션 윈도우와 스코어보드 블록으로 구성된다. 인스트럭션 윈도우는 패치된 명령어 중에 병렬처리가 가능한 명령어들을 분리한다. 윈도우 내에서 명령어들의 목적지 레지스터와 소스 레지스터가 동일하면 의존성이 제거될 때까지 명령어의 실행을 미룬다. 스코어보드 블록은 인스트럭션 윈도우에서 출력된 명령어들과 현재 파이프라인에서 실행중인 명령어들의 목적지 레지스터와 소스 레지스터 값을 비교한 후 의존성을 제거한다. 의존성이 발견될 경우 스코어보드는 의존성이 있는 명령어의 실행을 다음 순서로 연기한다.

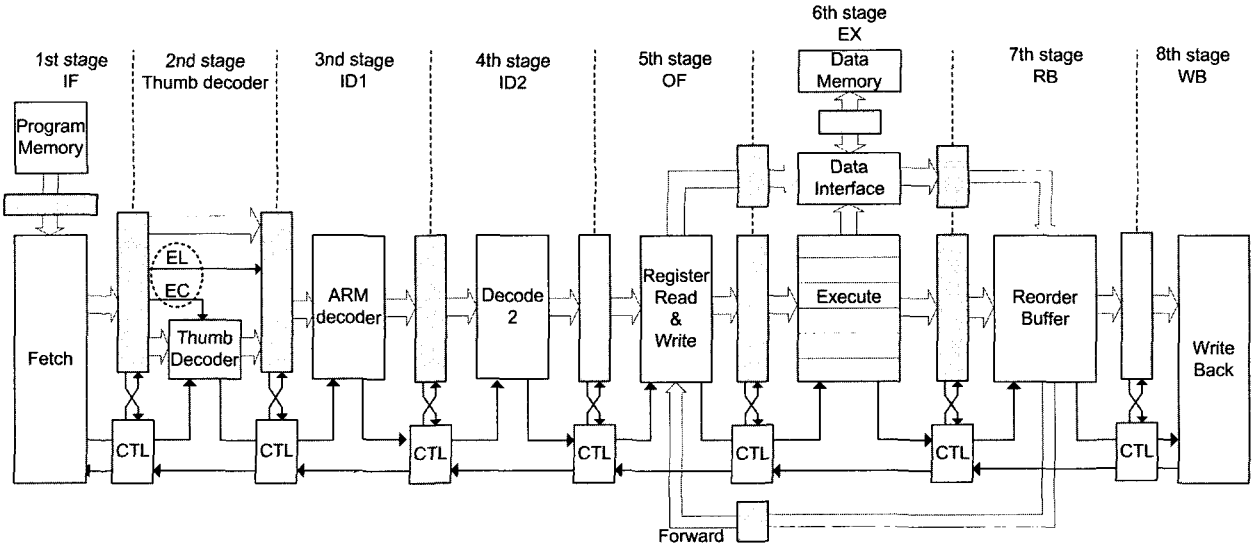

그림 6. 제안된 비동기식 ARM 프로세서의 구조
Fig. 6. The architecture of the proposed asynchronous ARM processor.

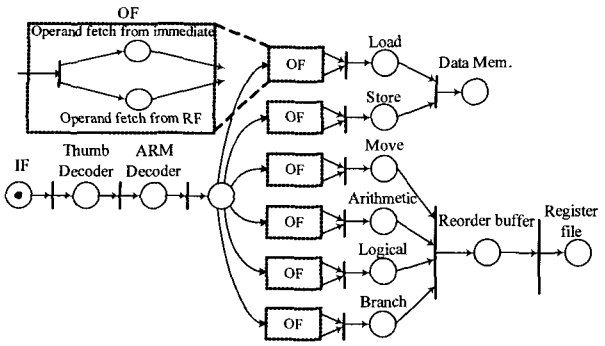

그림 7. 그림 6 파이프라인 구조의 페트리 넷 모델
Fig. 7. The Petri-Net model for the pipeline architecture of Fig 6.

제안된 적응형 파이프라인 구조를 프로세서에 적용하기 위해서 그림 6의 EC와 EL 신호가 스테이지 스키핑 및 스테이지 통합 제어신호로 사용되었다. IF 스테이지에서는 프리디코딩을 수행하여 입력된 명령어가 ARM 명령어인지 thumb 명령어인지 구분한다. 만약 명령어가 thumb 명령어이고 ID1 스테이지가 작업이 완료되어 비어있는 상태라면 EL 신호는 high가 되고 thumb 디코더와 ID1은 통합된다. 명령어가 ARM 명령어라면 EC 신호는 high가 되고 thumb 디코더의 동작은 생략되어 데이터패스가 짧아진다. 다른 예로 배럴 쉬프터(barrel shifter)와 그에 연결되어 있는 ALU가 있다. 쉬프터가 사용되고 ALU는 사용되지 않는 명령어가 수행되어야 할 경우, EL신호는 high가 되고 두 스테이지는 통합된다. 쉬프터가 사용되지 않는 경우에는 EC신호는 high가 되고 스테이지 스키핑이 수행된다.

제안된 프로세서 동작은 그림 7의 페트리 넷으로 표현된다. 데이터패스는 명령어의 종류에 따라 구분되고, 구분된 데이터패스를 갖는 명령어들은 병렬로 실행된다. OF스테이지는 오퍼랜드가 즉치값(immediate value)인지 레지스터 파일로부터 발생되는 값인지에 따라 두 가지 패스로 나뉜다. 예를 들어 MOV A, #immediate 명령어는 레지스터 파일로부터 얻는 오퍼랜드가 없으므로 OF 스테이지의 동작이 불필요하다. 따라서 이 명령어는 IF-ID-MOV-RB-WB의 데이터패스를 갖는다.

IV. 시뮬레이션 결과

제안된 적응형 파이프라인은 IDEC에서 제공된 hynix 0.35-㎛ CMOS 공정을 사용하여 합성되었다. 그림 8은 본 논문에서 제안한 적응형 파이프라인 구조의 시뮬레이션 결과를 보여준다. 스테이지 스키핑은 그림

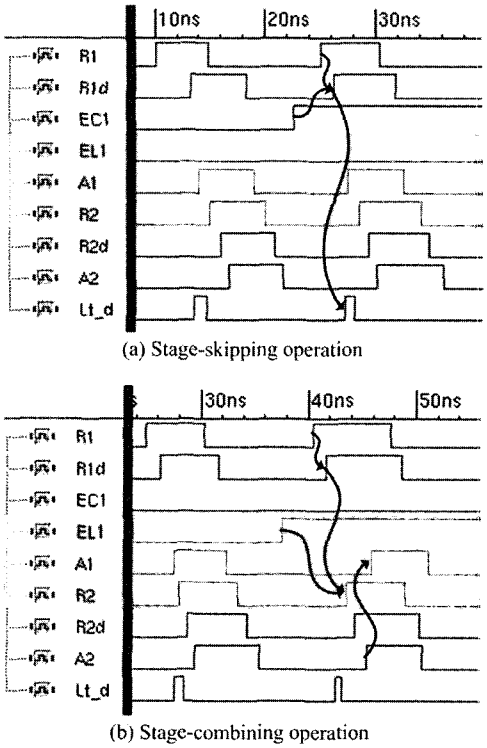

그림 8. 적응형 파이프라인 구조의 시뮬레이션 결과
Fig. 8. Simulation results of adaptive pipeline structure.

표 1. 각 스테이지별 평균 수행 시간
Table 1. Average execution time of each stage.

Stage	IF	ID		OF	EX	RB	RW	WB
		ID1	ID2					
Execution time	4.93ns	6.1ns	5.1ns	3.8ns	6.3ns	3.8ns	4.1ns	0.5ns

8(a)과 같이 EC신호가 high 일 때 수행된다. EC 신호를 전달받은 스테이지의 동작은 수행되지 않는다. 따라서 조합회로로 입력되는 요청신호인 R1과 조합회로에서 출력되는 요청 신호 R1d 사이의 지연이 스테이지 스키핑을 적용했을 때 감소된다. 그림 8(b)는 파이프라인에 스테이지 통합 기법을 적용했을 경우의 시뮬레이션 결과를 나타낸다. EL 신호가 high일 때는 래치가 데이터 저장의 동작을 하지 않고 연속된 두 스테이지가 하나의 스테이지로 통합된다. 이 때 첫 번째 스테이지의 동작 완료 확인 신호인 A1신호 는 두 번째 스테이지의 확인 신호 A2를 따라 천이한다.

표 1은 제안된 비동기 ARM 프로세서의 각 스테이지의 평균 실행 시간을 나타낸다. 성능 평가를 위해 SPEC2000 벤치마크의 8개 정수형 워크로드가 사용되었다. 그림 9는 파이프라인을 적용하지 않은 구조, 일반적인 파이프라인 방식을 적용한 구조 및 데이터패스 분리에 기반을 둔 제안된 파이프라인 구조를 적용하여 시

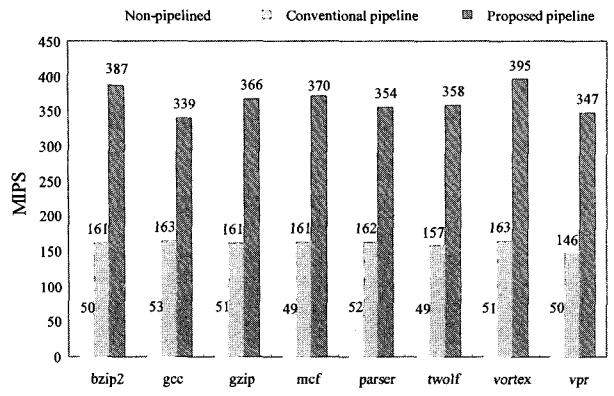

그림 9. 명령어 처리 속도 성능 비교

Fig. 9. Performance comparison by instruction processing speed.

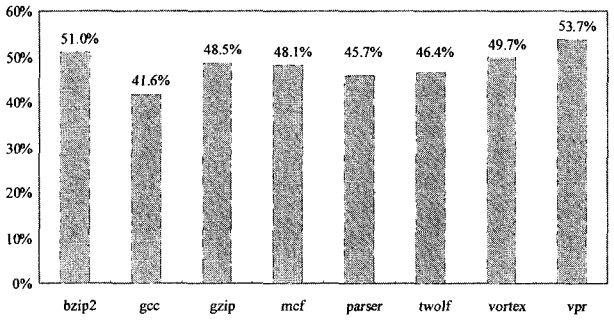

그림 10. 제안된 파이프라인 구조에서 명령어 병렬 처리율

Fig. 10. Parallel processed instruction ratio of the proposed pipeline architecture speed.

뮬레이션한 결과이다. 위의 세 구조에 대한 시뮬레이션 결과 명령어 처리 성능은 각각 평균 50MIPS, 161MIPS, 365MIPS로 나타났다. 이 결과는 프로세서에 일반적인 파이프라인 구조를 적용했을 때보다 제안된 파이프라인 구조와 멀티 프로세싱을 적용했을 때 명령어 처리 속도가 2.3배 더 빠르다는 것을 나타낸다.

그림 10은 SPEC2000의 각 벤치마크 프로그램의 총 명령어에 대한 병렬처리 비율을 나타낸다. 구현된 구조의 병렬 처리율은 평균 48%를 보인다. 이것은 동시에 1.48개의 명령어가 실행된다는 것을 의미한다. 4 way 슈퍼스칼라는 병렬로 2.3개의 명령어를 처리하는 것을 감안하면, 제안된 구조는 슈퍼스칼라와 비교하여 한 사이클 당 64%의 명령을 수행한다. 그러나 제안된 구조가 기능 블록의 중첩을 필요로 하지 않아 전력 당 명령어 처리 능력에서 앞설 것이라 예상된다.

맨체스터 대학에서 개발한 비동기 프로세서인 AMULET3i의 명령어 처리량은 CMOS 0.35um공정으로 설계되었을 경우 110MIPS에서 최대 176MIPS이다. ARM사의 ARM9 코어의 경우 0.13um공정에서 설계되어 253MIPS에서 최대 275MIPS의 성능을 보인다[11, 12]. 제안된 구조에 일반적인 파이프라인을 적용하였을 경우 성능은 AMULET3i 프로세서와 유사하다. 그러나 데이터패스 분리를 적용한 파이프라인 구조를 적용한 결과 명령어 처리량이 증가하여 비동기 방식을 적용한 AMULET3i와 비교하면 처리속도가 2.3배로 빨라진다.

V. 결 론

본 논문은 고성능 고속 비동기식 프로세서를 위한 적응형 파이프라인 아키텍처를 제안하였다. 제안된 아키텍처에는 데이터 패스를 줄이기 위한 방법으로 스테이지 스키핑과 스테이지 통합 방법이 적용되었다. 스테이지 스키핑은 불필요한 파이프라인 스테이지의 동작을 제거한다. 스테이지 통합은 현재 스테이지와 다음 스테이지를 하나의 스테이지로 통합한다. 따라서 두 기법은 프로세서의 실행 데이터 패스의 길이를 줄여 시스템 성능을 향상시킨다.

또한 본 논문은 제안한 파이프라인 구조의 연산 블록에 대한 비동기 멀티 프로세싱을 제안하였다. 제안된 멀티 프로세싱은 프로세서의 전체 데이터 패스를 각각의 독립적인 데이터 패스로 분할한다. 따라서 추가되는 회로 없이 서로 다른 독립된 명령어의 다중 프로세싱 방법으로 프로세서의 성능을 향상시킨다.

제안된 파이프라인 방법과 다중 프로세싱 방법을 적용하여 ARM 명령어 기반의 비동기 ARM 프로세서를 설계하고 그 성능을 평가하였다. 그 결과 구현된 회로는 48%의 명령어 병렬 처리율을 얻었으며, 이것은 1.48개의 명령어가 동시에 실행된다는 것을 나타낸다. 명령어 처리 속도는 약 364MIPS로 제안된 구조가 비동기 프로세서인 AMULET3에 비해 2.3배 빠른 명령어 처리 속도를 보였다.

참 고 문 헌

[1] M. Ozawa, M. Imai, Y. Ueno, H. Nakamura, and T. Nanya, "A cascade ALU architecture for asynchronous super-scalar processor," *IEICE Trans. Electron.*, vol. E84-C, no. 2, pp. 229-237, Feb. 2001.

[2] S. B. Furber, D. A. Edward, and J. D. Garside, "AMULET3: A 100 MIPS asynchronous embedded processor," *Proc. Computer Design*, pp. 329-334, 2000.

[3] A. Takamura, M, Kuwakao, M. Imai, T. Fujii,

M. Ozawa, I. Fukasaku, Y. Ueno, and T. Nanya, "TITAC-2 A 32-bit scalable-delay insensitive microprocessor," *Proc. ICCD,* pp. 288-294, Oct. 1997.

[4] S. B. Furber, J. D. Garside, S. Temple, J. Liu, P. Day, and N. C. Paver, "AMULET2e An asynchronous embedded controller," *Proc ASYNC'1997,* pp. 290-299, Apr. 1997.

[5] A. J. Martin, M. Nystrom, K. Papadantonakis, P. I. Penzes, P. Prakash, C. G. Wong, J. Chang, K. S. Ko, B. Lee, E. Ou, J. Pugh, E. Talvala, J. T. Tong, and A. Tura, "The lutonium sub-nanojoule asynchronous 8051 microcontroller," *Proc ASYNC'03,* pp. 14-23, 2003.

[6] J. H. Lee, Y. H. Kim, and K. R. Cho, "Design of a fast asynchronous embedded CISC microprocessor, A8051," *IEICE Trans. on Electron,* vol. E87-C, no. 4, pp. 527-534. Apr. 2004.

[7] J. M. Colmenar, O. Garnica, S. Lopez, J. I. Hidalgo, J. Lanchares, and R. Hermida, "Empirical characterization of the latency of long asynchronous pipelines with data-dependent module delays," *Proc. 12th EUROMICRO-PDP'04,* pp. 311-321, Feb. 2004.

[8] A. Efthymiou and J. D. Garside, "Adaptive pipeline structures for speculation control," *Proc. ASYNC'03,* pp. 46-55, May 2003.

[9] J. L. Peterson, "Petri nets," *ACM Computing surveys,* vol. 9, no. 3, sept. 1977.

[10] J. L. Peterson, *Petri nets theory and the modeling of systems,* Englewood Cliffs, Prentice Hall 1981.

[11] J. D. Garside, W. J. Bainbridge, A. Bardsley, D. M. Clark, D. A. Edwards, S. B. Furber, J. Liu, D. W. Lloyd, S. Mohammadi, J. S. Pepper, O. Petlin, S. Temple, J. V. Woods, "AMULET3i-an asynchronous system-on-chip," *Proc. ASYNC2000,* pp. 162-175, Apr. 2000.

[12] S. Segars, "The ARM9 family-high performance microprocessors for embedded applications," *Proc. ICCD,* pp. 230 - 235, Oct. 1998.

[13] D. Sima, T. Fountain, and P. Kacsuk, *Advanced Computer Architectures: a Design Space Approach,* Addition-Wesley, 1997.

저 자 소 개

이 승 숙(정회원)
2004년 충북대학교 전기전자공학부 공학사
2004년 9월~현재 충북대학교 정보통신공학과 석사과정
<주관심분야 : 고속 마이크로프로세서 설계, 저전력 시스템 설계>

임 영 일(학생회원)
2005년 충북대학교 정보통신공학과 공학사.
2005년 3월~현재 충북대학교 정보통신공학과 석사과정.
<주관심분야 : 저전력 디지털 회로 설계, 비동기 회로 설계>

이 제 훈(정회원)
1999년 충북대학교 정보통신공학과 공학사
2001년 충북대학교 정보통신공학과 공학석사
2005년 충북대학교 정보통신공학과 공학박사
2005년~2006년 Univ. of Southern California 방문연구원
2006~현재 충북 대학교 BK21 계약교수
<주관심분야 : 고속 마이크로프로세서 설계, 저전력 디자인>

조 경 록(정회원)
1977년 경북대학교 전자공학과 공학사
1989년 일본 동경대학교 전자공학과 공학석사
1992년 일본 동경대학교 전자공학과 공학박사
1979년~1986년 (주)금성사 TV연구소 선임연구원
1999년~2000년 Oregon State University 객원교수
1992년~현재 충북대학교 전기전자공학부 교수
<주관심분야 : 통신시스템 LSI 설계, 저전력 고속 회로 설계, Platform기반의 SoC설계>