論文98-35C-7-8

스택 기반 객체 지향형 영상 제어기 설계

(Design of Object-Oriented Video Controller based on Stack)

朴 柱 炫 *, 金 榮 民 *

(Ju-Hyun Park and Young-Min Kim)

요 약

본 논문에서는 객체 지향적 특성을 갖는 영상 데이터를 효율적으로 제어하기 위한 제어기를 설계한다. 고품질 영상 서비스에 대한 수요 급증에 따라 객체 단위의 영상 데이터를 처리할 필요성이 대두되었으며, 이러한 영상 객체 정보를 효과적으로 제어하기 위한 제어기가 필요하다. 본 논문에서는 일반 데이터와 리턴 어드레스를 저장하는 스택을 각각 둠으로써 스케줄링에 따른 오버헤드를 줄이는 스택 버퍼를 사용한 프로그래머블 구조를 제안한다. 스택 버퍼에서는 멀티태스킹, 타스크 교환 등이 가능하도록 데이터 간의 이동, 복사 등이 가능하다. 본 논문의 제어기는 32비트 정수형 데이터를 포함한 여러 형태의 데이터를 처리할 수 있는 확장형 명령어를 제공하며, 동작 속도는 44~77MHz이다.

Abstract

In this paper, a controller is designed for efficient controlling of video data with object-oriented feature. The need of controlling video data in an object unit is on the rise for the reason of rapidly growing demand for high-quality video services. We propose a programmable architecture with stack buffers which can minimize a scheduling overhead by having separate buffers for general data and return addresses. The buffers are very useful for move and copy operations for multitasking and task switching. The controller offers extended instructions that process several data types including 32bit integer type. Operation speed of the controller is from 77MHz to 44MHz.

I. 서 론

최근에 기존의 전화망이나 공중망 채널을 통한 음성뿐만 아니라 문자, 그래픽, 동영상 등을 포함한 다양한 멀티미디어 서비스에 대한 수요가 급증하고 있다. 특히 교육망을 비롯 기업체내의 인트라넷, 인터넷 등 다양한 상업망을 근간으로 한 인터넷 산업의 비약적인 발전과 응용 서비스의 출현은 다양한 형태의 데이터 전송을 필요로 하게 되었다. 그러나 지금까지 서비스되고 있는 대부분의 데이터는 문자, 간단한 그래픽, 초보 수준의 멀티미디어 정보이며, 데이터의 품질은 소비자들의 다양한 기호 발달에 따라 고품질의 멀티미디어 데이터로 바뀌어갈 것으로 예측된다. 특히 영상 데이터는 고품질의 서비스를 위해 필수적인 것이라 할 수 있다. 고품질의 영상 데이터를 처리하기 위해서는 영상 신호의 통계적인 동작을 응용한 인간의 시각 시스템을 그 모델로 하는 차세대 코딩 방법이 필요하다. 이 방법은 인간의 시각 시스템에서 사용하는 방법과 유사한 이미지 데이터를 표현하도록 하는 기본적인 변화를 제안하고 있기 때문에 전통적인 디지털 표현에서 사용하는 픽셀 개념과는 다른 속성(Entity)을 표현하

* 正會員, 全南大學校 電子工學科
(Dept. of Electronics, Chonnam National Univ.)
接受日字:1998年3月23日, 수정완료일:1998年6月1日

는 메시지 코딩 문제로 귀결된다. 다양한 차세대 기술은 주로 객체(Object), 구성(Texture), 경계(Contours), 영역(Regions) 등을 코딩 단위로 이용한다[1]. 이 기술은 전통적인 블록 기반 코딩보다 더 복잡하기는 하지만 압축율에 있어 우월하다는 것이 증명되고 있다[2].

이러한 수요와 네트워크 환경에 맞는 64 kbps 이하의 저전송율 전송로를 통한 동영상 부호화 기법들을 ISO/IEC 산하의 MPEG-4 그룹과 ITU-T 산하의 LBC 그룹에서 활발히 연구하고 있다[3].

네크워크를 기반으로 하는 영상 처리 기법들은 대화할 수 있는 양방향성이 필요하며, 이와 같은 환경에 적응이 잘 되로돎 데이터 단위로 객체를 사용한다. 따라서 객체 지향형 영상 부호화 방법은 다양한 형태의 데이터를 처리하기 위한 기본적인 부호화 방식이라고 할 수 있다. 그러나 객체 지향형 부호화는 객체를 추출하기 위한 모델링 작업이 선행되어야 하며, 많은 연산을 필요로 하는 복잡한 방법이다. 따라서 복잡도를 해결하기 위해서는 영상화면에 포함되어 있는 정보를 추출한 후, 효과적인 부호화 작업으로 전환할 수 있는 제어기를 필요로 한다.

지금까지의 제어기는 IC에 명령어를 구현하거나 최적의 컴파일러를 설계함으로써 속도를 올리거나 블록을 기반으로 한 영상 데이터를 제어하도록 설계되었고, 로드해서 저장하는 방식을 지향하는 구조이기 때문에 MPEG-4와 같은 객체 지향형 언어로 기술된 영상 데이터의 제어에는 비효율적일 수밖에 없다. 따라서 본 논문의 객체 지향형 영상 제어기는 네트워크에 기반한 영상 데이터를 제어하는 것 뿐만 아니라 객체 지향형 언어적 특성을 갖는 구조이기 때문에 객체 데이터를 자원으로 사용하는 알고리즘의 경우 프로그램으로 구현할 수 있는 장점을 갖는다.

객체 지향 프로세서는 다양하게 나뉘어지는 객체 데이터에 대한 처리 능력을 높이기 위해 빠른 서브루틴 호출 성능을 제공해야 하고, 인터럽트 핸들링, 타스크 교환 등의 성능을 갖도록 하기 위한 구조는 스택에 기반한 프로세서로 알려져 있다[4]. 대표적인 스택 프로세서로 피코자바를 들 수 있다[5].

그러나 피코자바는 내장형 제어기로 사용할 때 몇 가지의 한계를 가지고 있다. 프로세서를 단일 언어에 대해 최적화하였기 때문에 C 응용과 같은 다른 환경의 성능에 영향을 미칠 수 있다. 또한 씬 클라이언트나 네트워크 컴퓨터형 응용만을 고려했기 때문에 서로 다른 응용, 환경 및 성능 수준에서는 구조의 연속성을 보장할 수 없다[6].

따라서 본 논문에서는 스택의 푸쉬, 팝 등의 빈번한 동작이 가능하도록 스택 버퍼를 내장하고 있으며, 명령어 또한 스택을 사용하는 간단한 구조를 가지고 있기 때문에 성능을 높이기 위한 복잡한 컴파일러 등이 추가로 필요하지 않다. 또한 영상 데이터의 제어 뿐만 아니라 영상 데이터를 저장하기 위한 레지스터 파일을 둠으로써 영상 프로세서의 내장형 제어기로 사용될 수 있도록 하였다.

본 논문의 Ⅱ장에서는 일반적인 스택의 기능을 살펴보고, 본 논문에서 사용하고 있는 스택 구조에 대하여 고찰한다. Ⅲ장에서는 영상 데이터 제어기 구조에 대하여 설명하고, Ⅳ장에서는 시뮬레이션을 통해 확인된 성능을 평가하고 분석한다. 마지막으로 Ⅴ장에서는 본 연구의 결과를 기술하고 향후 연구방향을 제시한다.

Ⅱ. 스택의 기능과 구조

스택 기반 프로세서는 레지스터 기반 프로세서와 비교해서 모듈화된 프로그램과 같은 경우엔 훨씬 더 효율적이다. 또한 단순하면서 적은 하드웨어를 사용하여 높은 계산 능력을 제공한다. 스택 기반 프로세서에서 사용하는 명령어 중 스택 명령어가 차지하는 비율은 약 40% 이상이다[5]. 따라서 스택 명령어의 구현은 프로세서 전체 성능에 큰 영향을 미칠 수 있음을 보여주고 있다. 스택은 프로세서에서 여러 방식으로 구현할 수 있다. 가장 일반적인 방법은 메모리에 어레이를 할당하여 가장 상위 요소의 어레이 인덱스로 변수 값을 나타낸다. 수행 효율을 높이고자 하는 프로그래머들은 메모리 블록의 위치를 할당할 때 위와 같은 방법을 채택하고 있으며, 스택 요소의 실제 어드레스 포인터를 유지함으로써 이것을 실현한다[7]. 그러나 어떠한 경우에도 스택 요소의 푸쉬 기능은 스택에 새로운 어드레스를 할당하여 그곳에 데이터를 저장한다. 또한 스택의 팝 기능은 스택으로부터 최상위 요소를 제거하여 요청한 루틴으로 그 값을 리턴한다.

하드웨어로 스택을 구현하면 소프트웨어 구현 보다 훨씬 더 빠르다는 장점이 있다. 명령어의 상당 부분을 스택 명령어로 사용하는 프로세서에서는 이와 같은 효율을 증가시키는 것이 고성능의 시스템을 얻는 데 중

요하다.

객체 지향형 언어와 같은 멀티태스킹, 서브 루틴 등이 많은 프로그램을 구현하는데는 데이터 요소를 접속하거나 서브루틴 리턴 어드레스를 접속하는데 프로그램 메모리 사이클이 필요치 않는 다중 스택 버퍼가 프로세서 속도를 올릴 수 있는 방법이다. 따라서 본 논문에서는 모듈화 객체 정보를 처리할 때 오버헤드를 줄이기 위해 파라미터, 지역 변수 등의 데이터와 리턴 어드레스를 따로 저장하는 이중 스택 구조를 갖는다. 대부분의 32비트 프로그램에 필요한 스택 버퍼 크기는 총 128개 정도인 것으로 알려져 있으며, 버퍼가 클수록 스택 명령어당 차지하는 사이클 수가 더 완만하게 줄어든다[4].

본 논문에서는 총 64개의 스택을 사용하며, RS (Return Stack)가 32개의 스택을 리턴 어드레스를 저장하는데 사용하며, 연속적인 서브루틴 호출을 할 수 있는 장점이 있다. 또한 지역 변수 및 파라미터 데이터는 DS(Data Stack)에 저장한다. 이와 같은 이중 스택 구조에서는 여러 계층의 서브루틴를 통해 파라미터를 전달할 수 있기 때문에 데이터를 새로운 파라미터 목록에 올리기 위한 오버헤드가 필요 없다. 그림 1은 본 논문에서 사용하는 DSU(Data Stack Unit) 블록도이다. 쉬프트 레지스터 구조를 갖는 스택을 사용하며, 각 쉬프트 레지스터는 긴 채인 레지스터로 구성된다.

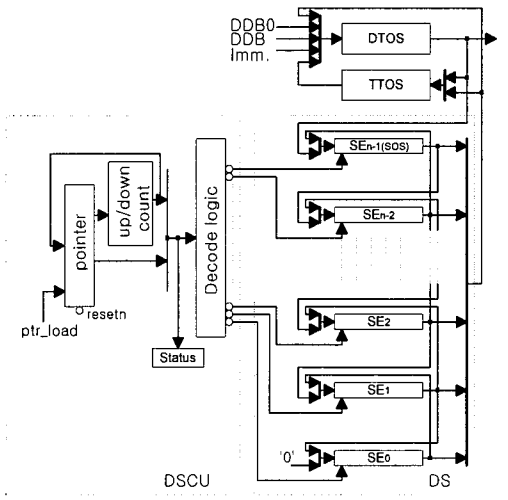


그림 1. DSU의 블록도
Fig. 1. Block diagram of DSU.

DSU는 데이터를 저장하기 위하여 32개 하드웨어 스택을 갖는다. 포인터는 6비트 크기를 가지고 있으며, 5비트 LSB는 SE(Stack Element)를 선택하는데 사용하며, 1비트 MSB는 스택의 오버플로우, 언더플로우을 찾기 위해 사용한다. 또한 스택은 속도를 증가시키기 위해서 최상위 스택 요소를 저장해 두는 DTOS (Top of Data Stack) 버퍼를 둔다. 예를 들어 덧셈의 경우 만약 버퍼가 없다면 두 개의 오퍼랜드를 패취하고, 그 결과를 저장하기 위해 메모리 접속이 3 사이클 추가된다. 그러나 스택 버퍼에 두 요소를 미리 저장해 둘 경우 단지 저장하기 위해 스택에 접속하는데 필요한 1사이클이 추가될 뿐이다. 또한 각 SE에 저장되어 있는 데이터를 상호 교환하거나 복사, 이동 등을 효율적으로 할 수 있도록 DTOS와 각 SE 사이에 TTOS(Temporary TOS)를 둔다. 그림 2는 DTOS와 연산 코어, 스택 버퍼 사이의 동작 관계를 보여주고 있다.

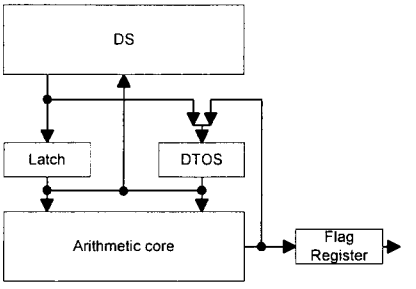


그림 2. DTOS, 연산 코어, DS 간의 동작 블록도
Fig. 2. Block diagram of inter-operation of DTOS, arithmetic core and DS.

DTOS는 프로그래머가 사용하는 데이터 스택의 최상위 요소를 가지고 있다. 따라서 데이터 스택 블록의 최상위는 실제는 두 번째 요소가 되는 것이다. 이와 같은 구조는 최상위 두 개의 스택 요소 값을 연산하고자 할 때 단일 포트만을 이용해서 연산이 가능하도록 해 준다[4]. 또한 연산 코어 출력이 DTOS에 연결되어 있어서 연산 코어 입력단에 있는 래취와 DTOS를 이용하면 어큐뮬레이션 동작도 가능하다.

리셋 상태에서 DSU의 포인터는 '111111'으로 초기화하며, 스택의 오버플로우와 언더플로우를 탐색할 수 있다. 오버플로우와 언더플로우 탐색은 다음과 같은 연산으로 가능하다.

O(오버플로우) = (pointer [5] ='1')
AND (pointer [4:0] ='00000') (1)

U(언더플로우) = (pointer [5] ='1')

AND (pointer [4:0] ='11110') .

초기화 상태의 스택 포인터는 스택이 빈 상태임을 나타내는 '111111' 값을 가지고 있다. 스택 포인터가 '111111'에서 '111110'로 될 때 스택은 언더플로우가 되며, '011111'에서 '100000'로 되면 스택 오버플로우가 된다.

Ⅲ. 제어기 구조

그림 3은 스택을 기반으로 한 제어기 구조를 나타낸다.

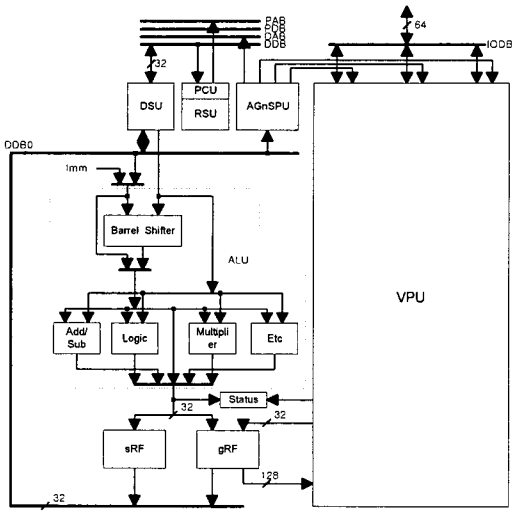


그림 3. 제어기 블록 다이어그램
Fig. 3. Block Diagram of Controller.

그림 3의 DSU는 데이터를 저장하는 스택 블록이며, RSU는 리턴 어드레스를 저장하는 스택 블록이다. PCU(Program Conter Unit)는 외부 프로그램 메모리의 어드레스를 연산하는 블록이고, AGnSPU (Address Generation & Stack Pointer Unit)는 외부 스택 메모리와 DSU 사이의 데이터 송수신을 제어하는 블록이다. VPU(Vector Processing Unit)[8]는 영상 데이터를 처리하고 저장하는 블록이며, gRF (general Register File)는 영상 데이터를 임시 저장하기 위한 레지스터 파일이다.

1. 코어 연산부

코어 연산부는 크게 ALU와 곱셈기, 배럴 쉬프터로 구성된다. ALU는 32비트 덧셈기,뺄셈기, 32비트 배럴 쉬프터, AND, OR, XOR, NAND 기능이 가능한 32비트 논리연산기로 구성된다. 덧셈기, 뺄셈기, 곱셈기는 부호 있는 정수 연산과 부호 없는 연산을 할 수 있도록 설계되었다. 그리고 연산 결과에 따라 N (Negative), Z(Zero), V(oVerflow), C(Carry) 등 4개의 상태를 출력한다.

배럴 쉬프터는 쉬프트할 데이터와 몇 비트를 이동시킬 것인가를 입력한다. 각각 왼쪽, 오른쪽, 산술, 논리 이동이 가능하도록 설계되어 있으며, 왼쪽 쉬프트인 경우 오른쪽 비트들은 0으로 채워지므로 산술적, 논리적 쉬프트의 결과는 같으나 오른쪽 쉬프트인 경우 왼쪽 비트들은 산술적 쉬프트에서는 부호 비트로 채워지고 논리적 쉬프트에서는 0으로 채워지므로 결과값이 다르다. 산술적인 왼쪽 쉬프트인 경우 부호있는 정수형 32비트로 표현할 수 있는 값을 넘어서면 오버플로우가 일어난다. 배럴 쉬프터의 출력은 ALU에 입력되어 또 다른 연산을 만들며, 영상 처리와 같은 비트 조작이 많은 알고리즘에 유용하다.

덧셈, 뺄셈기는 이진 보수화 체계를 가지므로 부호없는 연산을 하는 경우 상태값만 바뀌고 출력값은 같기 때문에 한 블록으로 구현이 가능하지만 곱셈의 경우 부호있는 곱셈과 부호없는 곱셈 방법은 구조가 서로 다르고, 계산 결과 값 또한 다르기 때문에[9], 각각 16비트 부호없는 곱셈기와 부호 있는 곱셈기를 사용한다. 또한 필요에 따라 부호없는 곱셈과 부호 있는 곱셈을 따로 계산할 수 있기 때문에 연산 속도를 높일 수 있는 장점이 있다.

오퍼랜드는 모두 32비트 부호있는 정수이며, 소스 오퍼랜드에서 32비트 정수의 부호 비트(MSB, 즉 비트 31)와 하위 15비트로 16비트 부호있는 정수를 생성한 후 연산을 하며 결과는 32비트 부호있는 정수가 된다. 단 소스 오퍼랜드가 16비트 부호있는 정수의 범위를 벗어나도 오버플로우를 발생하지 않는다. 즉 소스 오퍼랜드의 비트 30-비트16은 무시한다.

ALU의 출력은 배럴 쉬프터, 덧셈기, 뺄셈기, 논리연산기, 곱셈기 출력 중 하나가 된다. ALU의 출력은 N, Z, V, C등의 4개의 상태값을 출력한다. Z는 출력이 있는 모든 연산에 대하여 유효하며 출력이 0인 경우 1이 된다. N은 부호있는 연산에 대해서만 유효하며 출력의 맨 상위 비트가 1인경우 1이 된다. V는 배럴 쉬프트나 덧셈기, 뺄셈기 출력에 대해서만 유효하

며 연산 결과가 오버플로우인 경우 1이 된다. C는 덧셈기, 뺄셈기의 출력에 대해서만 유효하며 연산중 캐리가 발생하는 경우 1이 된다. 부호있는 연산의 경우 V는 맨 상위 비트 캐리와 그 전 비트의 캐리의 XOR 연산에 의해 구해지며, C는 맨 상위 비트의 캐리이다. 부호없는 연산의 경우 V는 맨 상위비트의 캐리가 되며 C와 같다.

2. 레지스터

레지스터는 범용 레지스터 16개, PC(Program Counter), FLR(Flag Register), BsR(Base Register), IxR(Index Register), SP(Stack Pointer), VP(Variable Pointer), FP(Frame Pointer), OP(Opcode top Pointer) 등이 있다[10] [11].

레지스터는 벡터 연산을 할 때 데이터를 VPU내의 메모리에 직접 쓰거나 읽는 경우 이외에 레지스터를 이용한 데이터 처리가 가능하도록 하기 위해 4개씩 묶어 벡터 레지스터로도 사용이 가능하도록 설계하였다. 이는 VPU에서 발생한 벡터 데이터를 단일 사이클에 레지스터에 저장하기 위해서이다.

SP는 외부 데이터 메모리의 스택 영역과 레지스터 사이의 데이터 교환을 위해 스택의 어드레스를 저장하고 있으며, VP는 현재 객체의 지역 변수를 저장하고 있는 곳의 메모리 어드레스를 가지고 있다. 이 레지스터가 필요한 이유는 VP가 가르키는 외부 데이터를 DS로 가져온 후 실행을 하는 동안 DS는 내부 연산 과정에서 계속적인 동작을 통해 DS를 갱신하기 때문에 현재 지역 변수 위치를 저장해 두지 않으면 외부 데이터의 위치를 알 수 없기 때문이다. 또한 현재 수행되고 있는 프레임 포인터 역할을 하는 FP가 있으며, 오퍼랜드 스택의 최상위를 저장하고 있는 OP 등이 있다[11].

3. 버스 구조

DDB0(Data Data Bus 0)는 데이타 버스로, 내부의 모든 레지스터와 연결되어 있어서 읽기, 쓰기가 가능하다. PC는 이 데이터 버스에 접속할 수 없다. 오퍼랜드를 코어 연산부로 가져오거나 레지스터 들간의 데이터 이동 및 스택과 레지스터간, 스택과 스택간의 데이터 이동을 하는데 이용한다. PAB(Program Address Bus), PDB(Program Data Bus)는 제어기에서 수행되는 외부 프로그램 메모리의 어드레스와 데이터를 입, 출력하기 위한 버스이며, DAB(Data Address Bus), DDB(Data Data Bus)는 외부 데이터 메모리의 어드레스와 데이터를 제어기에 입, 출력시키기 위한 버스이다. 또한 IODB(Input/Output Data Bus)는 VPU 내부에 있는 영상 데이터를 DRAM과 같은 외부 프레임 메모리로부터 프로세서 내부로 입, 출력하기 위한 버스이다.

4. 파이프라인

파이프라인은 명령어를 패취하기 위한 F(Instruction Fetch) 단계, 명령어 디코딩을 수행하는 D (Decode) 단계, 오퍼랜드 어드레스를 계산하는 A (Address generate)단계, 명령어를 수행하는 E(Execute) 단계 등 4단계로 나눌 수 있다. F 단계에서 D 단계로 전달되는 정보는 명령어 코드 그 자체이다. D 단계에서 A 단계로 전달되는 정보는 명령어에 따른 동작 신호와 오퍼랜드 어드레스 계산에 필요한 정보를 전달한다. A 단계에서 E 단계로 전달하는 정보는 프로그램 메모리의 어드레스 및 파라미터 값과 RS에 백업을 위한 신호 등을 전달한다.

F 단계는 외부 프로그램 메모리로부터 하나의 명령어를 패취하여 그 값을 명령어 레지스터에 저장한다. 다음에 패취할 명령어 어드레스는 분기나 점프 동작이 일어날 때는 PC 연산 블록에서 새로운 PC값을 연산하여 다음 사이클에 패취를 한다.

D 단계는 F 단계에서 입력받은 명령어 코드를 디코딩한다. 각 명령어 코드는 32비트이다. 이 단계에서는 각 명령어의 PC 연산 블록 제어, 우선분기 프로세싱[12] 등을 수행한다. 우선분기 프로세싱은 조건부 분기 명령어나 무조건 분기 명령어의 리턴 어드레스를 예측한다.

A 단계는 메모리를 접근하는 명령어의 경우 어드레스의 연산 과정이 필요하며, 이때 단일 사이클에 어드레스를 생성한다. 서브루틴 호출 명령어에서는 리턴 어드레스를 연산하여 레지스터에 저장하고, 분기 타켓 명령어를 연산한다. 리턴 명령어에서는 외부 데이터 메모리 내의 스택 영역을 참조하기 위한 어드레스를 연산한다.

E 단계는 명령어 수행을 통해 데이터를 처리하는 단계이다. ALU, 배럴 쉬프터, 곱셈기, DS 등을 사용한다. 메모리를 접근하는 명령어는 단일 사이클 동안 레지스터 값을 메모리에 저장하거나 메모리에서 값을 읽어 레지스터에 저장한다. 분기 명령어는 조건 분기

여부를 판단하고, PC값을 갱신하거나 RS 및 외부 스택 메모리에 리턴 어드레스를 저장한다. DS에서 데이터를 상호 교환 등을 할 때는 두 사이클이 필요하다.

5. 명령어 세트

본 제어기는 동작 특성에 따라 5개 그룹으로 명령어를 나눈다. 본 제어기에서 구현한 대표적인 명령어의 구성은 표 1과 같다[13].

표 1. 스택 기반 프로세서의 명령어 구성
Table 1. Instruction composition in the stack-based processor.

명령어	차지하는 비율
연산 명령어	47%
분기 명령어	4%
메모리 명령어	9%
스택 명령어	23%
벡터 명령어	17%

전체 명령어 개수는 62개이며, 오퍼랜드 데이터 형태에 따라 명령어를 각각 6개까지 확장할 수 있도록 하였다. 메모리 명령어 중에는 벡터 단위 영상 데이터뿐만 아니라 픽셀 단위 데이터 처리가 가능하도록 VPU내 메모리로부터 8비트, 16비트 데이터를 레지스터에 가져올 수 있는 픽셀 명령어가 존재한다. 또한 스택 명령어는 DS내 데이터의 상호 이동, 복사를 하는 명령어가 있으며, 외부 스택 메모리와 DS 사이에 SP, VP, FP, OP에 따라 푸쉬, 팝 동작을 하는 명령어가 존재한다.

오퍼랜드 데이터 형태는 C 코드에서 지원하는 형태를 갖도록 하였으며, 이는 다른 구조에서도 호환이 가능하도록 하기 위함이다. 지원 가능한 데이터 형태는 표 2와 같다[5].

표 2. 데이터 형태
Table 2. Data type.

type	meanings	Description
byte	1-byte signed 2's complement int.	B
short	2-byte signed 2's complement int.	S
int	4-byte signed 2's complement int.	I
long	8-byte signed 2's complement int.	L
char	2-byte unsigned Unicode char.	C
object	4-byte reference to object	O

표 1의 연산 명령어는 다른 명령어 비해 차지하는 비율이 상대적으로 높다. 그러나 오퍼랜드에 스택과 레지스터가 올 수 있기 때문에 스택내의 데이터를 처리할 때는 스택 명령어로 분류할 수 있으므로 차지하는 비율은 최고 70%라고 할 수 있다. 또한 연산 명령어와 메모리 명령어는 명령어 특성에 따라 확장이 가능하며, 확장된 명령어는 총 104개이다. 연산 명령어 중 덧셈, 뺄셈, 곱셈, 비교 등은 문자형 데이터와 같은 부호없는 데이터에 대해서도 처리 가능하도록 설계하였으며, 객체형 데이터는 객체 어드레스로서 정수형 데이터와 같은 형태로 취급한다.

명령어 포맷의 오퍼랜드 수는 하드웨어 스택과 밀접한 관계가 있는 것은 아니다. 그러나 실질적으로 어드레싱 모드 수는 스택을 어떻게 설계했는가 또는 프로그램이 스택을 어떻게 이용하는가에 달려 있다. 본 논문의 제어기는 2-오퍼랜드 명령어 포맷을 가지고 있으며, 명령어가 Bs(Bus Source) 필드와 Bd(Bus Destination) 필드를 포함하도록 한다. 따라서 오퍼랜드가 없거나 한 개 필드를 갖는 명령어보다 유연성이 훨씬 높다. 또한 명령어 수행에 필요한 데이터를 DTOS에 미리 가져다 놓고, Bs, Bd에 따라 연산을 수행한다. 한 쪽 오퍼랜드 값을 미리 가져다 놓기 때문에 명령어 디코딩과 동시에 오퍼랜드 레지스터를 읽을 수 있어서 처리 속도가 증가한다. 이와 같은 기능은 세 개의 오퍼랜드가 있는 효과를 나타내며, 오퍼랜드의 패취와 저장을 위한 파이프라인 동작의 필요성을 제거하는 효과가 있다.

Ⅳ. 성능 평가 및 분석

설계한 객체 지향형 영상 제어기는 코어에 대해 0.6 ㎛ 5-Volt TLM COMPASS 라이브러리를 사용하여 검증하였다. 표 3은 여러 프로세서 내의 제어기 동작을 비교한 것이다.

본 논문의 제어기는 연산부의 곱셈기에서 최장 경로를 형성한다. 데이터 처리는 32비트 단위이지만 32비트 곱셈기를 사용할 경우 단위 사이클당 40ns 이상의 시간이 지연되며, 이는 25MHz 동작 속도를 갖게 되므로써 현저한 성능 저하를 초래한다. 따라서 본 논문에서는 부호있는 곱셈 연산과 부호 없는 연산이 서로 다르기 때문에 16비트 부호 있는 곱셈기와 16비트 부호 없는 곱셈기를 각각 설계함으로써 최소 44MHz 동

작 속도를 얻는다. 시뮬레이션 결과는 44~77MHz의 동작 속도를 갖는다. 정수형 데이터는 44MHz로 동작하며, 바이트 데이터는 77MHz의 동작 속도를 갖는다. 정수형 데이터는 부호 있는 곱셈의 경우 16비트 곱셈을 4 번 실행하면 그 결과를 얻는다. 문자형과 객체형은 부호없는 연산만을 하기 때문에 50MHz로 정수형에 비해 연산 속도가 약 12% 빠르다.

표 3. 여러 프로세서 내의 제어기 동작 비교
Table 3. Comparison of controller operation in several processors.

	Technology	동작주파수	성능	데이터버스
A [14]	0.3㎛CMOS 3LM	70MHz	.	16비트
B [15]	0.25㎛CMOS 4LM	81MHz	81MIPS	16비트
C [16]	0.45㎛CMOS 2LM	66.6MHz	52.4MIPS	32비트
D [17]	0.75㎛CMOS 2LM	45MHz	450MIPS	16비트
Ours	0.6㎛CMOS 2LM	50MHz	200MIPS	32비트

표 3의 A, B, D 예는 영상 프로세서의 제어기로서 데이터 버스는 모두 16비트이다. 이는 모든 영상 정보를 16비트 데이터로 표현할 수 있기 때문이다. C는 RISC 전용 프로세서로 32비트 데이터 버스를 갖는다. 본 논문의 제어기는 영상 데이터 뿐만 아니라 객체 지향 언어적 특성을 갖는 프로그램을 제어하기 위해 외부 데이터 메모리 접속 회수가 많고, 프레임 등 블록 단위 데이터의 보존과 이동이 많기 때문에 32비트 버스가 효율적이라고 판단된다. 또한 MPEG-4의 CAE(Context-based Arithmetic Coding), 스프라이트(Sprite) 코딩[1] 등은 32비트 데이터를 필요로 하기 때문에 본 제어기의 확장성 면에서도 32비트 구조가 더 효율적이다. 표 4는 대표적인 스택 기반 내장형 제어기인 피코자바와의 특성 비교를 보여주고 있다[18].

본 논문의 제어기는 피코자바와 비교하여 더 작은 명령어 수를 가지고 있지만 피코자바 대부분의 명령어를 구현한다. 또한 32비트 명령어 포맷을 가지고 있기 때문에 한 명령어당 한 사이클 수행이 가능하다. 또한 스택을 기반으로 동작하기 때문에 4단의 파이프라인으로 충분하며, 스택 명령어를 제외한 대부분의 명령어가 단일 사이클에 수행이 가능하다. 외부 메모리를 읽고, 쓰는 메모리 명령어는 동시에 같은 어드레스를 접속하지 않으면 충돌이 일어나지 않는다. 스택 명령어 중에서 스택 간의 데이터 상호 교환 등을 하는 명령어는 2 사이클이 필요하지만 이는 피코자바의 3사이클보다 한 사이클이 작다.

표 4. 피코자바와의 특성 비교[5]
Table 4. Feature comparison with picoJava.

	PicoJava	Ours
명령어 수	224개	62개+ 110개(확장형)
한 동작당 사이클 수	3 번	1 번(2번)
파이프라인	4단	4단
스택 버퍼 수	1	2(64 SEs)
레지스터 수	4개	26개 + reserved

또한 범용으로 사용할 수 있는 레지스터 파일이 있으므로 범용 RISC와 같은 동작이 가능하며, 따라서 벡터 연산이 이루어질 경우 피코자바는 레지스터에 저장해야 할 값을 스택에서 가져와야 하기 때문에 패취에 따른 동작 사이클 수가 늘어나야 하지만 본 제어기는 단일 사이클 처리가 여전히 유효하다. 따라서 응용범위가 피코자바에 비해 훨씬 넓다고 할 수 있다. 또한 스택 버퍼 수가 2개이므로 파라미터 값과 어드레스를 따로 저장할 수 있어 스캐줄링에 따른 제어가 따로 필요없다.

V. 결 론

본 논문에서는 네트워크 환경에 적응이 용이한 객체 지향형 영상 프로그램을 효율적으로 처리하기 위한 32비트 제어기를 설계하였다. 설계한 코어는 0.6㎛ 5-Volt TLM COMPASS 라이브러리를 사용하였으며, 처리하는 데이터 형태에 따라 최고 77MHz까지 동작 속도를 갖는다. 다양한 데이터 형태를 처리할 수 있도록 32비트 정수형을 기본으로 바이트형, 문자형 등 확장 명령어를 사용할 수 있으며, 4단 파이프라인을 가지고 있다. 스택 명령어를 제외한 모든 명령어는 한 사이클에 한 동작이 가능하다. 레지스터는 스택 버퍼 이외에 영상 데이터의 벡터 연산 결과를 저장할 수 있도록 32비트 뿐만 아니라 4개의 레지스터를 단일 레지스터로 인식하도록 설계하였다.

또한 고품질의 영상 데이터를 객체화 하여 처리할 수 있도록 일반 변수 데이터와 리턴 어드레스를 저장하는 스택 버퍼를 각각 둠으로써 스케줄링에 따른 오버헤드를 줄인다. 본 논문의 제어기는 벡터 연산부의 추가에 따른 명령어 필드, 레지스터, 파이프라인 등을 고려하여 설계하였기 때문에 앞으로 영상 데이터를 처리하기 위한 벡터 연산부인 VPU 블록을 쉽게 추가할 수 있는 장점이 있다. 앞으로 벡터 연산부와 내부 메모리를 추가하여 객체 지향형 영상 디코더로 활용할 예정이다.

참 고 문 헌

[1] "MPEG-4 Video Verification Model Version 8.0", ISO/IEC/JTC1/SC29/WG11, MPEG97/N1796, Jul. 1997.

[2] K. R. Rao, J. J. Hwang, *Techniques & Standards for Image · Video & Audio Coding*, Prentice Hall, 1996.

[3] "MPEG-4 Project Description", ISO/IEC/JTC1/SC29/WG11 DOC. no. 96/N1177, Jan. 1996.

[4] P. J. Koopman, Jr., "Stack Computers: the new wave", http://www.cs.cmu.edu/~koopman/stack_computers/, 1989.

[5] M. Tremblay, M. O'Connor, "picoJava: A Hardware Implementation of the Java Virtual Machine", http://infopad.eecs.berkeley.edu/HotChips8/4.3/, 1996.

[6] J. S. Baer, "x86 아키텍처를 통한 내장형 자바 플랫폼의 구축", 아시아 전자 엔지니어, pp. 44-49, Feb. 1998

[7] A. V. Aho et. al., *Compilers : Principles, Techniques, and Tools*, Addison-Wesley, pp. 401-410, 1986.

[8] 박주현, 김영민, "데이터패스를 이용한 SA-DCT 구현", 대한 전자공학회 논문지, 35권 제5호, 1998년 5월

[9] K. Hwang, *Computer Arithmetic*, John Wiley & Sons, pp. 161-167, 1979.

[10] A. van Someren, C. Atack, *The ARM RISC Chip : A Programmer's Guide*, ARM Limited, 1993.

[11] T. Lindholm, F. Yellin, *The JavaTM Virtual Machine Specification*, Addison Wesley, 1996.

[12] M. R. Cosgrove, et. al., "Instruction address stack in the data memory of an instruction-pipelined processor", United States Patent, Appl. no. 280, 417, 1983.

[13] *TMS320C4x User's Guide*, TI Inc., 1993.

[14] M. Takahashi, et. al., "A 60mW MPEG4 Video Codec Using Clustered Voltage Scaling with Variable Supply-Voltage Scheme", *ISSCC'98 Digest*, pp. 36-37, Feb. 1998.

[15] E. Miyagoshi, et. al., "A 100㎟ 0.95W Single-chip MPEG2 MP@ML Video Encoder with a 128GOPS Motion Estimator and a Multi-Tasking RISC-Type Controller", *ISSCC'98 Digest*, pp. 30-31, Feb. 1998.

[16] T. Shimizu, et.al., "A Multimedia 32b RISC Microprocessor with 16Mb DRAM", *ISSCC'96 Digest*, pp. 216-217, Feb. 1996.

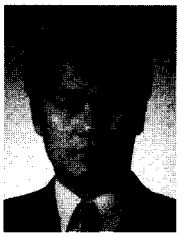
[17] D. Brinthaupt, et.al., "A Video Decoder for H.261 Video Teleconferencing and MPEG Stored Interface Video Applications", *ISSCC'93 Digest*, pp. 34-35, Feb. 1993.

[18] T. Hokugo, "Technology trend : Java Chips Aim at Embedded Control of Java chip", http://www.nikkeibp.com/nea/september/septt.html, Sept. 1996.

저 자 소 개

朴 柱 炫(正會員)

1969년 7월 13일생. 1993년 2월 전남대학교 전자공학과 졸업. 1995년 2월 전남대학교 대학원 전자공학과 졸업(공학석사). 1997년 2월 전남대학교 대학원 전자공학과 수료(공학박사). 1998년 2월 ~ 현재 전남대학교 공과대학 전자통신기술연구소(ETTRC) 전임연구원. 홈페이지 주소 : http://chonnam.chonnam.ac.kr/~pjh. 주관심분야는 MPEG4 코덱 설계, 객체 지향 프로세서, DSP, RISC 프로세서 설계 등임



金 榮 民(正會員)

1954년 4월 18일생. 1976년 2열 서울대학교 전자공학과 졸업(공학사). 1978년 2월 한국과학원 전기및전자공학과 졸업(공학석사). 1978년 3월 ~ 1979년 7월 한국선박해양 연구소(주임연구원). 1986년 오하이로 주립대학교 전기공학과(공학박사). 1988년 6월 ~ 1991년 8월 한국전자통신연구소(실장). 1991년 9월 ~ 현재 전남대학교 전자공학과 교수. 1998년 2월 ~ 현재 전남대학교 공과대학 전자통신기술연구소(ETTRC) 소장. 1997년 12월 ~ 현재 반도체설계교육센터(IDEC) 전남대학교 지역센터장. 주관심분야는 ADSL 모뎀 설계, MPEG2, MPEG4 시스템 설계 등