論文2001-38SP-5-4

영상처리용 프로세서를 위한 효율적인 이차원 어드레스 지정 기법

(An Efficient 2-dimensional Addressing Mode for Image Processor)

高 綸 浩 *, 尹 炳 珠 *, 金 聖 大 *

(Yun-Ho Ko, Byoung-Ju Yun, and Seong-Dae Kim)

요 약

본 논문에서는 프로그램 가능한 하드웨어 장치에서 영상 처리를 효율적으로 수행하기 위한 새로운 메모리 어드레스 지정 방법(addressing mode)을 제안한다. 기존의 어드레스 지정 방법은 음성과 같은 일차원적인 형태의 데이터 처리에 적합한 반면, 제안된 메모리 어드레스 지정 기법은 영상 데이터의 이차원적인 특성을 고려한 새로운 메모리 어드레스 지정 기법이다. 제시된 기법은 기존의 메모리 구조를 바꾸지 않으면서도 이차원 데이터의 위치를 표시할 수 있는 두개의 오퍼랜드를 입력으로 메모리에 저장된 영상데이터 값을 처리는 명령어이다.

이차원적인 특성을 지니는 새로운 메모리 어드레스 지정 기법은 다음과 같은 장점을 지닌다. 먼저, 기존 하드웨어에서 여러 명령어에 걸쳐 수행해야 할 작업을 통합함으로써, 수행해야 할 프로그램의 코드 사이즈를 줄여 하드웨어의 성능을 높임과 동시에 근래 무선 응용 분야에서 요구되는 저전력 동작을 가능하게 한다. 아울러, 영상 데이터가 가지는 이차원적인 특성을 그대로 반영하므로, 사용자가 보다 쉽게 어셈블러를 통해 어플리케이션을 프로그램 할 수 있다. 이와 같은 이차원적인 메모리 어드레스 지정 기법은 각종 DSP, media processor, 그래픽 장치 등에 이용될 수 있다. 본 논문에서는 이러한 이차원 메모리 어드레스 지정 기법의 개념을 제안함과 동시에, 이를 효율적으로 구현하기 위한 하드웨어 구조를 제시한다.

Abstract

In this paper, we propose a new addressing mode, which can be used for programmable image processor to perform image-processing algorithms effectively. Conventional addressing modes are suitable for one-dimensional data processing such as voice, but the proposed addressing mode consider two-dimensional characteristics of image data. The proposed instruction for two-dimensional addressing requires two operands to specify a pixel and doesn't require any change of memory architecture.

The proposed two-dimensional addressing mode for image processor has the following advantages. The proposed instruction combines several instructions to load a pixel data from an external memory to a register. Hence, the proposed instruction reduces required code size so that it satisfies high performance and low power requirements of image processor. In addition, it uses inherent two-dimensional characteristics of image data and offers user-friendly instruction to assembler programmer. The proposed two-dimensional addressing mode is applicable to DSP, media processor, graphic device, and so on. In this paper, we propose a new concept of two-dimensional addressing mode and an efficient hardware implementation method of it.

* 正會員, 韓國科學技術院
(KAIST:Korea Advanced Institute of Science & Technology)

接受日字:2000年11月15日, 수정완료일:2001年5月29日

Ⅰ. 서 론

최근 컴퓨터, 직접회로, 통신및신호처리 기술의 발전에 힘입어서, 본격적인 멀티미디어 시대가 열리게 되었다. 특히 개인용 컴퓨터의 대중화, 디지털 전송기술의 발전, 고화질 디스플레이 장치 실현, CD의 보편화 및 메모리 디바이스의 저가격화에 따라 멀티미디어의 개념이 영상이라는 정보매체를 중심으로 급속히 변화되고 있다. 그런데 영상 정보매체는 음성이나 문자 등과 같은 다른 형태의 정보매체에 비해서 훨씬 많은 양의 정보를 가지고 있다는 점에서 그들과 대비된다. 특히, 영상 신호에는 색 성분이 추가로 포함될 수 있기 때문에 보다 많은 정보량은 물론 그 전달 효과가 뛰어나다. 이러한 양적인 차이 외에도 영상신호는 인간 고유의 시각체계에 의해서 처리되는 2차원 형태의 신호로 구성되어 있기 때문에 음성이나 텍스트와 같은 다른 종류의 신호들과 그 성격이 다르다고 할 수 있다.

이처럼 영상 정보매체는 멀티미디어 시대에 정보 전달 수단으로 중요한 역할을 할 것이며, 현재도 각종 비디오 폰 서비스, 의료 영상, 감시 카메라, 3차원 그래픽 시스템 등에 널리 이용되고 있다. 그런데, 영상 정보는 앞서 언급한 바와 같은 특성들로 인하여 시스템을 구현하고자 하는 경우 많은 계산량과 메모리 소자 등을 필요로 하므로 실시간 구현에 많은 어려움을 가지고 있다. 특히, 영상 데이터가 가지는 이차원적인 특성을 충분히 고려하지 않으므로 인해 이의 처리를 위한 하드웨어 부담이 가중된다.

이처럼 이차원 데이터 처리가 요구되는 경우는 이차원 필터링 과정과 같은 간단한 예를 통해 살펴볼 수 있다. 영상 처리에서 자주 발생하는 필터링 과정은 각 화소에 대해 일정한 크기의 윈도우를 설정하고, 윈도우 내의 각 화소들을 해당 필터링 과정이 요구하는 형태로 처리한다. 즉, 하나의 전체 화면을 화소별로 이동하면서 윈도우 내의 데이터를 읽어 들이고 이를 처리하게 되는데, 이때 이러한 사각형 형태로 데이터를 읽기 위해서는 많은 부가적인 연산이 요구된다. 즉, 대부분의 필터링 과정에서 요구되는 필터 계수에 각각의 데이터를 곱해서 누적하는 연산 외에 이차원 형태로 데이터를 읽기 위한 부가적 연산이 요구된다는 것이다. 또한, 화면의 경계에 이러한 윈도우가 놓여지게 되면 보다 많은 부가 연산과 처리가 요구된다.

이처럼 이차원 형태의 데이터 처리가 요구됨에도 불구하고 기존의 DSP나 영상처리 프로세서는 메모리에 저장된 데이터를 일차원적으로 처리한다[1~3]. 즉, 실제 발생하는 영상 데이터가 이차원 형태임에도 불구하고, 구현상의 편의와 기존에 DSP나 영상처리 프로세서가 고려하는 데이터가 주로 일차원 형태의 음성데이터였기 때문에 메모리에 저장된 데이터를 일차원적으로 처리하는 하드웨어 구조만이 제시되었다. 물론, 일차원 데이터 처리를 통해서 이차원 데이터의 처리가 불가능한 것은 아니지만, 이차원상에서 보다 효율적으로 구현되는 과정을 일차원적으로 이행함으로써 추가적인 비용이 초래된다.

이차원적인 데이터 처리를 지원하는 가장 간단한 방법으로 이차원 메모리의 사용이 있을 수 있다. 그런데, 이러한 이차원 메모리를 구현하는데 있어서 다음과 같은 문제들이 발생한다. 첫째, 사용하고자 하는 이차원의 크기에 따라 다른 메모리를 사용해야 한다는 것이다. 예를 들어, QCIF 영상을 사용하고자 할 때와 SIF 영상을 사용하고자 하는 경우, 영상 크기에 따라 각각 다른 이차원 구조의 메모리를 사용해야 한다. 둘째, 이차원적인 데이터 접근을 위해서는 메모리 버스의 크기와 데이터 버스의 크기가 커지게 된다. 끝으로, 음성과 같은 일차원 데이터를 영상데이터와 동시에 사용하고자 하는 경우 이차원적인 메모리 구조는 일차원 데이터 처리에 있어서 역효과를 가져올 수 있다.

이차원적인 데이터 처리를 지원하는 또 다른 방법으로는 기존 메모리 구조를 바꾸지 않고 메모리에 저장된 영상데이터 값을 이차원적인 개념으로 읽어 들이거나 저장하는 어드레스 지정 기법을 사용하는 것이다. 이러한 방법은 어드레스 지정 기법이 다소 복잡해지는 단점이 있지만 기존의 메모리 구조는 변형시키지 않으면서도 이차원적인 데이터 처리를 지원할 수 있게 된다.

본 논문에서는 기존 메모리 구조를 바꾸지 않고 메모리에 저장된 영상데이터 값을 이차원적인 개념으로 읽어들이거나 저장하는 명령어를 제시한다. 이러한 이차원 명령어의 사용은 전체적인 코드 길이를 줄여 하드웨어의 성능을 개선할 뿐 아니라 이차원 데이터를 읽거나 저장하는데 필요한 여러 명령어를 통합함으로써 하드웨어 구현 시 전력감소의 효과를 제공한다. 또한 중요 영상처리 알고리즘을 어셈블리로 구현하고자 할 때 이차원 명령어는 사용자에게 보다 친숙한 명령어를 제공함으로써 부호화 작업을 용이하게 한다.

또한 본 논문에서는 이러한 이차원 메모리 어드레스 지정 기법을 구현하기 위한 하드웨어 구조를 제시한다. 구현 과정에서 영상 크기가 항상 16의 배수라는 것과 피연산항의 입력을 일정 범위로 제한함으로써 메모리 어드레스 지정을 위한 피연산항의 처리를 최소한의 하드웨어 비용으로 구현하게 된다.

본 논문의 구성은 다음과 같다. 먼저 Ⅱ장에서는 이차원 메모리 어드레스 지정 기법의 개념에 대하여 기술하고, 영상처리 알고리즘을 통해 새로운 개념의 이차원 메모리 어드레스 지정 기법의 필요성을 제시한다. Ⅲ장에서는 이러한 이차원 어드레스 지정 기법을 구현하기 위해 요구되는 하드웨어 구조를 살펴 보고, Ⅳ장에서는 이러한 어드레스 지정 기법을 통한 이득을 정량적으로 살펴 본다.

Ⅱ. 이차원 어드레스 지정 기법의 개념 및 필요성

1. 이차원 어드레스 지정 기법의 개념

그림 1은 ILOAD(image load)라는 새로운 명령어의 동작을 설명하고 있다. ILOAD 명령어는 본 논문에서 제안하고자 하는 이차원 어드레스 지정 기법을 설명하기 위해 새롭게 고안된 명령어이다.

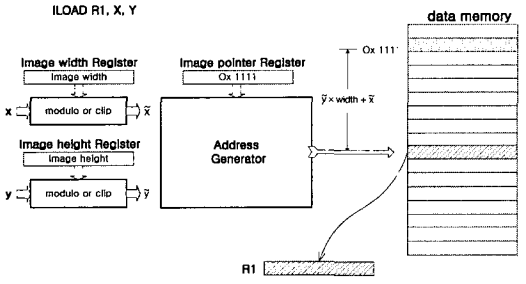


그림 1. ILOAD의 동작 절차
Fig. 1. Operation procedure of ILOAD.

ILOAD 명령어는 기존의 일차원 어드레스 지정 기법을 사용하는 명령어와는 그 성격이 다르다. ILOAD 명령어는 메모리에 저장된 영상 데이터를 이차원적인 개념으로 지정된 레지스터(register)로 읽기 위한 명령어이다. 실제 영상 데이터는 이차원적으로 구성되어 있다. 즉, 하나의 화소를 지정하기 위해서는 수평 방향으로의 성분과 동시에 수직 방향으로의 성분을 알아야 한다. 그런데, 지금까지의 프로세서는 대개 이차원 영상 데이터를 일정한 주사 방법에 따라 일차원으로 나열하고 이를 음성과 같은 일차원 데이터와 동일한 방식으로 처리해 왔다. 여기에 비해 ILOAD는 영상 데이터를 이차원적으로 처리한다.

그림 1에서 x와 y는 영상데이터가 저장된 메모리의 위치를 지정하기 위한 두개의 피연산자이고, 각각 영상의 수평 방향과 수직 방향으로의 위치를 지정한다. 여기서 x와 y는 각각 유효한 화소만을 지정하기 위해 영상의 폭과 높이를 벗어나서는 안 된다. 따라서, x와 y값이 영상 크기의 한계를 벗어나지 않도록 적절한 처리를 해야 하는데, x와 y가 영상 사이즈를 벗어나는 경우 영상처리 알고리즘에 따라 잘라내거나(clipping) 순환(modulo)시키는 연산이 주로 사용된다. 이를 위해서 영상 크기에 대한 정보를 저장하기 위한 별도의 자원이 요구되는데, 이를 위해 영상 폭 레지스터(image width register)와 영상 높이 레지스터(image height register)가 이용된다. 다시 말해서 그림 1에서와 같이 입력된 피연산항 와 는 각각 영상 크기 정보를 가지고 있는 레지스터 값과 비교되어 잘라내어지거나 순환되어지고, 영상 크기의 한계를 벗어나지 않는 값인 $\tilde{x}$와 $\tilde{y}$로 각각 수정된다.

변형된 $\tilde{x}$와 $\tilde{y}$값은 이차원 영상에서 접근하고자 하는 화소의 위치를 나타내고 있지만 이것이 메모리의 특정 위치를 표현하지는 못한다. 그림 1에서 주소 발생기(address generator)는 변형된 $\tilde{x}$와 $\tilde{y}$값을 이용해 처리하고자 하는 화소에 대응하는 메모리 위치로부터 저장된 값을 읽기 위한 동작을 하게 된다. 이때 $\tilde{x}$와 $\tilde{y}$ 값 뿐만 아니라 메모리 전체에서 해당 영상 데이터가 시작되는 위치를 알아야 한다. 이를 위해서 별도의 저장장치가 있어야 하고, 그림 1에서는 영상 지정 레지스터(image pointer register) 가 메모리상에서 영상 데이터가 시작되는 주소를 저장하게 된다. 주소 발생기는 영상 지정 레지스터 값, 영상 폭 레지스터의 값과 변형된 와 값을 이용해 접근하고자 하는 화소의 메모리 위치를 결정하게 된다. 예를 들어, 그림 1에서와 같이 영상 데이터가 ox1111번지부터 메모리에 순차적으로 저장되어있다고 할 때 $\tilde{x}$와 $\tilde{y}$값을 이용해 접근해야할 화소의 메모리 위치를 구하기 위해서는 주소 발생기가 $ox1111 + \tilde{y} \times width + \tilde{x}$의 연산을 해야 한다.

제안된 이차원 어드레스 지정 기법은 기존의 인덱스드 어드레싱 모드(indexed addressing mode)와 비교되

어 질 수 있다. 기존의 인덱스드 어드레싱 모드에서는 하나의 피연산자를 사용해 접근하고자 하는 메모리 위치의 기저 주소(base address)를 저장하는 공간으로 사용하고 또 다른 하나의 피연산자를 사용해 인덱스(index)를 지정하게 된다. 예를 들어, 인덱스드 어드레싱 모드를 사용하고 있는 "$LOAD_{indexed}$ R3, R1, R2"라는 명령어는 R1 레지스터에 저장되어 있는 값을 접근하고자 하는 메모리 위치의 기저 주소로 사용하고 R2에 저장된 값을 인덱스로 사용해 최종적으로 접근하고자 하는 메모리 위치를 결정하고 해당 메모리에 저장되어 있는 값을 R3 레지스터에 저장하게 된다. 즉, R3 레지스터와 R1 레지스터에 저장되어 있는 두 값을 더한 것에 해당하는 번지의 메모리 위치로부터 값을 읽어 들이게 된다.

이러한 이차원 어드레스 지정 기법은 기존의 인덱스드 어드레싱 모드에 비하여, 하나의 기저 주소로부터 일정한 간격 만큼 떨어진 메모리 위치를 접근한다는 공통점이 있지만 다음과 같은 차이점을 지닌다. 첫째 이차원 어드레스 지정 기법은 기저 주소로부터의 간격을 이차원적으로 표현한다는 것이다. 인덱스드 어드레싱 모드에서는 앞선 예($LOAD_{indexed}$ R3, R1, R2)에서와 같이 R1이라는 기저 주소로부터 일차원적으로 R2 만큼 떨어진 위치의 메모리 값을 처리한다. 반면 이차원 어드레스 지정 기법에서는 "ILOAD R3, x, y"라는 명령어가 있을 때 영상 지정 레지스터(image pointer register)라는 특수 목적 레지스터에 저장된 기저 주소로부터 $y \times width + x$에 위치한 메모리 값을 처리하게 된다. 즉, 인덱스드 어드레싱 모드에서는 "Mem[R1]"에서와 같이 메모리를 일차원적으로 다루는 것이고, 이차원 어드레스 지정 기법은 "Mem[y][x]"에서와 같이 메모리를 이차원적으로 다루게 된다.

둘째, 인덱스드 어드레싱 모드에서는 기저 주소로부터의 간격을 표시하는 오퍼랜드에 대하여 부가적인 처리를 하지 않는다. 반면 이차원 어드레스 지정 기법에서는 영상 폭 레지스터(image width register)와 영상 높이 레지스터(image height register)와 같은 별도의 특수 목적 레지스터를 두고 이 레지스터 값과 오퍼랜드의 값을 비교해, 해당 오퍼랜드가 각각 영상의 폭과 높이에 해당하는 범위를 벗어나지 않도록 부가적인 처리를 하게 된다. 이처럼 이차원 어드레스 지정 기법은 인덱스드 어드레싱 모드와는 다른 동작을 수행하게 된다.

이후의 II장 2절에서는 이러한 이차원 어드레스 지정 기법이 영상 처리 응용프로그램을 수행하는데 있어서 어떠한 장점을 가지는지를 살펴 본다. 한편 이러한 이차원 어드레스 지정 기법은 인덱스드 어드레싱 모드에 비해 다소 복잡한 동작을 수행하므로 부가적인 하드웨어가 요구되는데, III 장에서는 이러한 이차원 어드레스 지정기법을 효율적으로 구현하기 위한 하드웨어 구조에 대하여 살펴본다.

2. 이차원 어드레스 지정 기법의 필요성

이차원 어드레스 지정 기법의 개념을 II장 1절에서 살펴 보았다. 이러한 이차원 어드레스 지정 기법은 기존의 어드레스 지정 기법에 비해 다소 복잡한 구조를 가진다. 본 절에서는 이러한 이차원 어드레스 지정 기법이 영상 처리를 수행하는데 있어서 어떠한 장점을 가지는지를 설명한다. 이를 위해 이차원 어드레스 지정 기법이 간단한 RISC 프로세서에 적용되었을 때 기존의 인덱스드 어드레싱을 포함하는 RISC 프로세서에 비해 어떠한 장점을 가지는지를 비교한다.

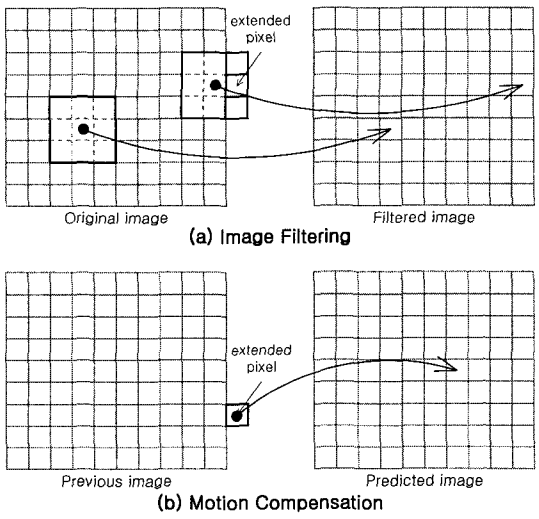


그림 2. 이차원 메모리 어드레스 지정 기법을 적용할 수 있는 영상처리 알고리즘의 예
(a)영상 필터링
(b)움직임 보상

Fig. 2. Examples of image processing algorithm applicable to 2-dimensional addressing mode
(a)Image filtering
(b)Motion compensation

그림이나 사진과 같은 영상은 수직 방향과 수평 방향으로의 성분을 지니는 이차원 데이터라고 볼 수 있다. 실제로 영상 처리 프로그램에서 "image[x][y]" 형태

의 데이터 선언을 자주 접할 수 있는데 이것은 대부분의 알고리즘이 이차원적인 데이터 처리를 요구하고 있음을 의미한다. 이처럼 소프트웨어적인 처리에서 이차원적으로 처리되는 영상 데이터를 하드웨어를 통한 처리에서도 이차원 개념을 도입한다면 그 성능을 극대화할 수 있을 것이다. 앞서 설명한 이차원 메모리 어드레스 지정 기법은 이차원 데이터 처리가 요구되는 대부분의 영상처리 알고리즘에 적용할 수 있다. 그림 2는 제안된 메모리 어드레스 지정 기법이 사용될 수 있는 영상처리 알고리즘들 중 영상처리에서 필수적인 이차원 필터링 과정과 동영상 움직임 보상과정을 보여주고 있다.

그림 2의 (a)는 이차원 메모리 어드레스 지정 기법이 영상 필터링 과정에서 사용될 수 있음을 설명하기 위한 그림이다. 영상 처리에서 필터링은 잡음제거, 영상강화(image enhancement), 영상복원(image restoration), 웨이브릿 변환, 전처리 과정 등 그 응용이 매우 다양하다[4]. 이러한 필터링 과정은 그림 2의 (a)에서와 같이 하나의 전체 화면을 화소별로 이동하면서 윈도우 내의 데이터를 읽어 들이고 이를 처리하게 되는데, 이때 이러한 사각형 형태로 데이터를 읽어 들이기 위해서는 많은 부가적인 연산이 요구된다. 즉, 대부분의 필터링 과정에서 요구되는 필터 계수에 각각의 데이터를 곱해서 누적하는 연산 외에 이차원 형태로 데이터를 읽기 위해 다수의 부가적인 연산이 요구된다는 것이다.

예를 들어 그림 2의 (a)에서와 같이 화면의 경계에 윈도우가 놓여질 수 있는데, 이 경우 윈도우 내에는 있지만 영상 밖에 존재하는 데이터에 대한 처리를 해야만 한다. 필터링 과정에서 이러한 데이터의 처리는 영상이 이차원 공간상에서 주기적으로 반복된다고 가정하고 처리하는 방법과 영상 경계의 가장 자리에 있는 화소가 확장된다고 가정하고 처리하는 방법을 사용한다. 만약 II장 1절에서 설명한 순환되어지는 이차원 메모리 어드레스 지정 기법을 사용할 수 있다면 영상이 주기적으로 반복된다고 가정하고 처리하는 방법과 동일하게 될 것이다. 한편, 잘라내는 이차원 어드레스 지정 기법을 사용하는 것은 영상 데이터의 경계에 있는 화소값이 확장된다고 가정하는 것과 동일한 개념이 된다.

그림 2의 (b)는 움직임 보상과정을 나타내고 있다. H.263의 비제한적 움직임 벡터 모드(unrestricted motion vector mode)에서와 같은 경우 부호화된 이전 영상의 경계에 있는 화소들을 외부로 확장함으로써 외부에 있는 화소들을 참조할 수 있다[5]. 이 모드는 영상의 경계를 따라 움직임이 있거나, 카메라의 움직임과 같은 전역 움직임에서 두드러진 이득을 얻을 수 있다. 그림 2의 (b)에서와 같이 움직임 벡터가 부호화된 이전 영상 외부에 있는 화소를 참조할 경우 이 외부 화소는 영상 경계에 있는 화소로 대체 된다. 이러한 움직임 보상 과정은 그림 3과 같은 의사코드로 표현할 수 있다.

<C Code >

```
int motion_vetor_x, motion_vector_y;
int x, y, x_pred, y_pred;
int prev_image[height][width];
int compensated_image[height][width];
.
.
.
for( all current image pixel  x, y ){
  x_pred = x + motion_vector_x;
  y_pred = y + motion_vector_y;
  if( x_pred > image_width )
     x_pred = image_width;
  if( y_pred > image_height )
     y_pred = image_height;
  compensated_image[y][x] = prev_image[y_pred][x_pred];
}
```

그림 3. 움직임 보상 과정에 대한 의사코드
Fig. 3. Pseudo code for motion compensation procedure.

그림 3의 의사코드는 (x, y)인 현재 화소에 대한 움직임 보상을 위해, 먼저 현재 화소의 위치 값에 움직임 벡터 값(motion_vector_x, motion_vector_y)을 더해 움직임 보상 시 참조해야 하는 화소의 위치를 구한다. 이때 참조 위치가 영상 경계를 벗어나는 경우 영상 경계에 있는 값을 확장 시켜서 만들어진 영상을 이용해야 한다. 이러한 동작을 위해 그림 3의 의사코드는 조건문을 사용하여 참조 위치가 영상 경계를 벗어나는 경우 영상 크기의 최대값으로 제한 시키게 된다. 이러한 동작은 이차원 메모리 어드레스 지정 기법의 개념과 일치하는 것이다. 이차원 메모리 어드레스 지정 기법의 장점을 설명하기 위해 그림 4와 5는 각각 제안된 기법이 사용되지 않은 경우와 사용된 경우에 대한 RISC 프로세서의 어셈블리 코드를 나타내고 있다. 여기서 그림 5에서 사용된 ISTORE 명령어는 ILOAD와 대응하는 데이터 저장 명령어이다.

그림 4와 5에서 움직임 보상 시 화소별로 화면 전체를 검색하는 루프에 대한 부분은 이해를 돕기 위해 어셈블리 코드로 기술하지 않고 루프내에 각 화소별로 움직임 보상을 하는 부분만 어셈블리 코드로 기술하였

<Convetional Assembly Code >

```
%R1, R2 : motion_vetor_x & motion_vector_y
%R3, R4 : current pixel x & y
%R5, R6 : predicted pixel x_pred & y_pred
%Previous Image ptr address 0AAAAH
%Compensated Image ptr address 01000H
%R7, R8 : Image width & Image height
%R9 : compensated pixel value(prev_image[y_pred][x_pred])
%R10, R11 : temporary storage
.
.
.
for( all current image pixel  R3, R4 ){
        ADD R5, R3, R1          %x_pred = x + motion_vector_x
        ADD R6, R4, R2          %y_pred = y + motion_vector_y
        BCND Path1, R5, le. R7          %x_pred <= image_width
        LOAD R5, R7                      %x_pred = image_width
Path1 : BCND Path2, R6, le. R8          %y_pred > image_height
        LOAD R6, R8                     %y_pred = image_height
Path2 : MUL R10, R5, R7                  %x_pred * image_width
        ADD R11, R10, R6      %x_pred * image_width + y_pred
        LOAD_Indexed  R9, #0AAAH, R11
        MUL R10, R3, R7                          %x * image_width
        ADD R11, R10, R6            %x * image_width + y_pred
        STORE_Indexed  #01000H, R11, R9
}
```

그림 4. 움직임 보상을 수행하기 위한 기존 프로세서의 어셈블리 코드

Fig. 4. Assembly code to perform motion compensation in the conventional processor.

<Assembly code using proposed instruction (ILOAD)>

```
%R1, R2 : motion_vetor_x & motion_vector_y
%R3, R4 : current pixel x & y
%R5, R6 : predicted pixel x_pred & y_pred
%Previous Image ptr address 0AAAAH
%Compensated Image ptr address 01000H
%R9 : compensated pixel value(prev_image[y_pred][x_pred])
.
.
.
for( all current image pixel  R3, R4 ){
        ADD R5, R3, R1          %x_pred = x + motion_vector_x
        ADD R6, R4, R2          %y_pred = y + motion_vector_y
        ILOAD R9, R5, R6
        ISTORE R3, R4, R9
}
```

그림 5. 이차원 메모리 어드레스 지정 기법을 이용한 어셈블리 코드

Fig. 5. Assembly code using 2-dimensional addressing mode.

다. 그림 4의 코드에서 제안된 기법을 사용하지 않고 움직임 보상을 수행하기 위해서는 보다 많은 부가적인 코드가 요구됨을 알 수 있다. 첫째, 보상하고자 하는 화소가 영상 경계 밖에 있는 화소일 수도 있으므로 이를 처리하기 위해 그림 4의 코드는 그림 5의 코드에 비해 두개의 분기 명령어가 추가로 사용되었다. 분기 명령어의 추가 사용은 그 자체로 수행해야 할 명령어의 숫자를 증가시킬 뿐만 아니라 파이프라인 시 추가적인 제어-해저드(control hazard)를 야기 시키므로 프로세서의 성능을 크게 저하시키게 된다. 둘째 그림 4의 코드에서는 움직임 보상 하고자 하는 화소를 결정한 후 그 데이터가 저장된 메모리의 값을 읽기 위해 한번의 덧셈과 곱셈이 추가적으로 사용된다. 이에 반해 그림 5의 코드에서는 단 한 번의 이차원 메모리 어드레스 지정 기법을 사용해 접근하고자 하는 메모리의 위치가 영상 경계 밖으로 나가지 않도록 할 뿐만 아니라 메모리의 값을 읽기 위한 부가적인 덧셈이나 곱셈 연산도 요구하지 않는다.

이차원 메모리 어드레스 지정 기법은 메모리에 저장된 영상데이터 값을 이차원적인 개념으로 읽어 들이거나 저장하는 어드레스 지정 기법이다. 이러한 방법은 어드레스 지정 기법이 다소 복잡해져 하드웨어적인 부담이 다소 부가되지만 기존의 메모리 구조는 변형시키지 않으면서도 이차원적인 데이터 처리를 지원할 수 있게 한다. 이러한 이차원 어드레스 지정 기법의 장점은 앞선 두 가지의 예를 통해서 살펴본 바와 같이 전체적인 코드 길이를 줄여 하드웨어의 성능을 개선할 뿐 아니라 이차원 데이터를 읽거나 저장하는데 필요한 여러 명령어를 통합함으로써 하드웨어 구현 시 전력감소의 효과를 제공한다. 또한 중요 영상처리 알고리즘을 어셈블리로 구현하고자 할 때 이차원 명령어는 사용자에게 보다 친숙한 명령어를 제공함으로써 부호화 작업을 용이하게 한다.

Ⅲ. 이차원 어드레스 지정을 위한 하드웨어 구조

1. 잘라냄(clipping) 또는 순환(modulo)을 위한 하드웨어 구조

이차원 메모리 어드레스 지정 기법을 위해서는 두개의 피연산항을 영상의 폭과 높이에 따라 잘라내거나 순환시켜야 한다. 즉, 두개의 피연산항을 각각 영상의 폭 또는 높이와 비교하여, 피연산항이 영상 크기를 벗어나는 경우 잘라내거나 순환시키는 동작을 해야 한다. 이를 하드웨어적으로 구현하기 위해서 일반적인 비교기(comparator)를 사용하면 하드웨어적인 부담이 가중된다. 따라서, 본 논문에서는 이러한 하드웨어적인 부담을 줄이면서 두개의 피연산항을 잘라내거나 순환시키는 구조를 제시하고자 한다.

실제로 피연산항이 영상크기 보다 큰지 여부를 결정하기 위해서는 비교기를 쓰지 않을 수 없다. 그러나, 영상 크기가 가지는 특성과 피연산자가 가질 수 있는 값의 변화 범위를 고려하면 비교기를 쓰지 않고 동일한 결과를 발생시킬 수 있다. 먼저, 영상처리에서 처리해야 하는 데이터의 크기는 항상 16의 배수이다. 실제로 QCIF(176×144), CIF(352×288), SIF, CCIR601과 같이 표준화되어있는 영상 규격들은 모두 그 폭과 높이가 16의 배수이다. 이처럼 영상 규격이 16의 배수인 것은 영상 부호화 시 매크로 블록(macro block)의 크기가 16×16인 것과도 관계된다.

한편 그림 1에서 피연산항으로 입력되는 x와 y 값이 최대 영상 크기에 16을 더한 값 보다 작다고 가정할 수 있다. 이는 그림 2의 필터링 과정에서 윈도우 크기가 33×33 이하인 것을 의미 한다. 실제 필터링 과정은 모든 화소를 검색하며 윈도우 내부 값을 처리해야 하므로 연산량을 고려하여 33×33 이상의 크기를 가지는 윈도우를 사용하지 않으므로 이러한 가정은 타당하다. 또한 그림 2의 비제한적인 움직임 보상 모드에서도 매크로 블록의 크기가 16×16이고 영상경계 가장자리 값을 확장하게 됨으로 피연산항 x와 y 값이 영상 최대 값에서 16을 초과할 수 없다. 요약하면, 모든 영상 크기가 항상 16의 배수라는 것과 대부분의 영상 처리 알고리즘에서 피연산항으로 입력되는 x와 y값이 영상 크기의 최대값으로부터 16을 초과하지 않는다고 가정할 수 있다.

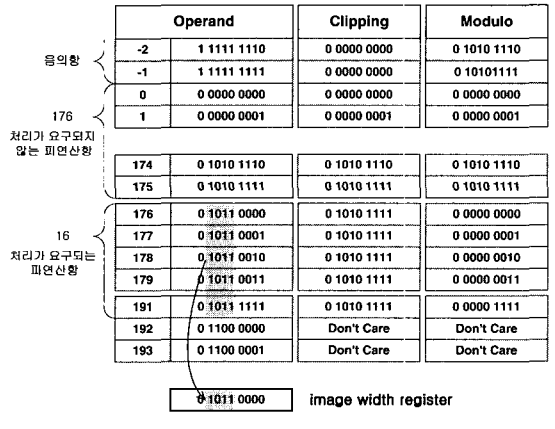


그림 6. 영상의 폭이 176화소인 영상에 대한 잘라내기와 순환동작의 예

Fig. 6. Examples of clipping and modulo operation in the case of image having 176 horizontal pixels.

그림 6은 영상의 폭이 176 화소이고 높이가 144인 QCIF 영상을 처리하는 경우에 발생하는 피연산항 x의 입력에 따라 잘라내거나 순환시키는 예를 도시하고 있다. 그림 6에서 첫번째 열은 피연산항으로 입력된 값을 나타내고 있으며, 두번째 세번째 열은 각각 대응하는 입력에 대하여 잘라내거나 순환하는 동작 후 발생한 값을 나타내고 있다. QCIF 영상은 앞서 기술된 영상 크기가 항상 16의 배수라는 가정을 만족시킨다. 따라서 수평 방향의 피연산항은 영상 폭이 176이므로 175이상의 값을 가져서는 안 된다. 그리고 앞선 가정에서와 같이 영상크기의 최대값으로부터 피연산항이 16을 초과하지 않으므로 처리해야 할 피연산항은 그림 6에서와 같이 176에서부터 191이며 192 이상의 입력 값은 무시할 수 있다.

여기서, 다음의 사실을 알 수 있다. 먼저 처리하고자 하는 영상의 크기가 항상 16의 배수이므로 영상의 크기를 2^4n(여기서 n은 자연수)이라 일반화 할 수 있다. 따라서, 입력된 피연산항은 0에서부터 2^4n-1 사이의 값으로 바뀌어져야 한다. 즉, 영상 내에 존재하는 화소를 표현하기 위해서는 2^4n 이상의 피연산항은 잘라내기와 순환동작에 따라 처리되어져야 한다. 그런데, 여기서 입력으로 가능한 피연산항이지만 처리가 요구되는 2^4n에서 2^4n+16의 범위에 있는 수는 그 하위 4비트는 다르지만, 하위 4비트를 제외한 상위 비트들의 형태가 동일함을 알 수 있다. 식 1은 임의의 자연수 n에 대하여 2^4n에서 2^4n+15 범위의 값을 이진수로 표현한 것이다.

$$
\begin{aligned}
2^4n &= p^k2^k + p^{k-1}2^{k-1} + \cdots + P^52^5 + P^42^4 + 0\cdot2^3 + 0\cdot2^2 + 0\cdot2^1 + 0\cdot2^0 \\
2^4n+1 &= p^k2^k + p^{k-1}2^{k-1} + \cdots + P^52^5 + P^42^4 + 0\cdot2^3 + 0\cdot2^2 + 0\cdot2^1 + 1\cdot2^0 \\
&\vdots \\
2^4n+14 &= p^k2^k + p^{k-1}2^{k-1} + \cdots + P^52^5 + P^42^4 + 1\cdot2^3 + 1\cdot2^2 + 1\cdot2^1 + 0\cdot2^0 \\
2^4n+15 &= p^k2^k + p^{k-1}2^{k-1} + \cdots + P^52^5 + P^42^4 + 1\cdot2^3 + 1\cdot2^2 + 1\cdot2^1 + 1\cdot2^0
\end{aligned}
\tag{1}
$$

식 1로부터 부가적인 잘라내기와 순환동작이 요구되는 입력 범위[2^4n, 2^4n+16]에 있는 피연산항은 그 하위 4비트를 제외한 상위 비트들의 형태가 동일함을 알 수 있다. 이것은 영상 크기가 16의 배수라는 것과 입력 피연산항이 영상 크기의 최대값으로부터 16을 초과 하지 않는다는 가정으로부터 성립하는 것이다.

한편 피연산항 중 2^4n에서 2^4n+16의 범위에 있는 수는 알고리즘에 따라 잘라내거나 순환시켜야 한다. 잘

라내는 경우에 있어서는 이 범위에 해당하는 피연산항이 입력될 때 간단히 영상 크기보다 1작은 값으로 바꾸어 주면 된다. 순환시키는 경우에 있어서는 그림 6에서와 같이 하위 4비트는 입력되는 피연산항과 동일하고, 이를 제외한 상위 비트들은 입력에 관계없이 0의 비트열을 가진다. 즉, 순환동작을 위해서는 하위 4비트를 입력 피연산항과 동일하게 하고 이외의 상위 비트들을 0으로 만들면 된다. 이처럼 간단한 방법으로 순환을 시킬 수 있는 것은 영상크기와 피연산항의 입력에 대한 앞선 가정으로부터 기인하는 것이다.

그림 7과 8은 각각 순환동작과 잘라내기를 위한 하드웨어 구조이다. 그림에서처럼 입력되는 피연산항은 일반적인 비교기에 의해서가 아니라 간단한 xor 로직에 의해 영상 범위를 초과 했는지 여부를 결정 받게 된다. 한편 xor 로직의 결과는 선택기(multiplexer)의 제어신호로 사용되어 피연산항이 영상 범위 내의 화소를 지정할 수 있도록 한다.

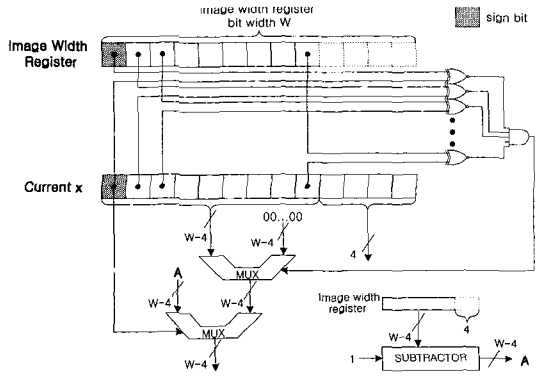


그림 7. 순환을 위한 하드웨어 구조
Fig. 7. Hardware architecture for modulo operation.

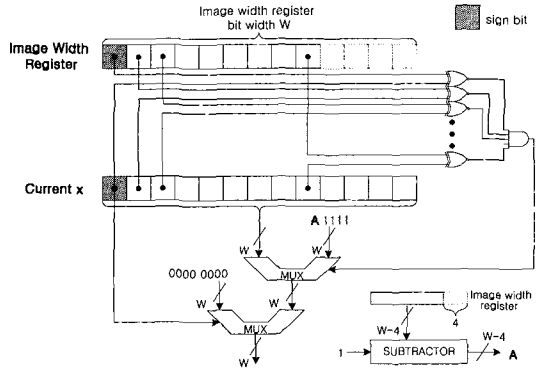


그림 8. 잘라내기를 위한 하드웨어 구조
Fig. 8. Hardware architecture for clipping operation.

먼저 피연산항이 가능한 영상 범위를 초과했는지 여부를 결정하는 로직의 원리는 다음과 같다. 앞선 가정에서와 같이 영상의 크기가 16의 배수이고 입력되는 피연산항이 최대값으로부터 최대 16을 초과하지 않는다고 하자. 이러한 가정하에서 변형이 요구되는 입력 범위 $[2^4n, 2^4n+16]$에 있는 값들은 피연산항의 하위 4비트를 제외한 상위 비트들의 형태가 동일함을 식 1로부터 알 수 있다. 이때 하위 4비트를 제외한 상위 비트들의 형태는 영상의 크기에 해당하는 2^4n의 형태와 동일하다. 따라서, 입력되는 피연산항은 영상 폭 레지스터(image width register) 또는 영상 높이 레지스터(image height register)에 저장되어 있는 영상 크기 정보와 xor로직을 통해 간단히 비교되어지고 피연산항이 영상 크기를 초과했는지 여부가 결정된다. 영상 크기가 16의 배수인 어떠한 경우에라도 그림에서와 같이 해당 크기를 레지스터에 저장하고 하위 4비트를 제외한 상위 비트들의 형태를 비교하는 간단한 xor 로직으로써 처리가 요구되는 피연산항인지를 결정할 수 있다.

그림 7의 순환동작을 위한 하드웨어 구조에서 xor 로직을 통해 해당 피연산자가 영상 범위를 벗어났는지 여부가 결정되고, 이를 제어 신호로 선택기가 동작하게 된다. 선택기는 피연산항이 영상 최대 범위를 벗어나는 경우 피연산항을 순환 시켜 출력하고 그렇지 않은 경우 현재의 입력 피연산항을 그대로 출력한다. 이때 피연산항을 순환시키는 경우에 있어서 하위 4비트는 입력 피연산항과 동일하게 하고 이를 제외한 상위 비트들은 그림 7 에서와 같이 모두 0으로 출력 된다.

한편 현재의 피연산항은 음수가 되어서는 안된다. 왜냐하면 영상에서 음의 위치에 해당하는 화소는 존재하지 않기 때문이다. 따라서, 피연산항의 입력이 음일 경우 부호 비트(sign bit)를 제어 신호로 사용하여 현재의 입력을 순환동작에 따라 적절하게 처리하는 선택기가 추가로 요구된다. 그림 6에는 음의 입력들에 대한 순환동작의 예가 도시되어 있다. 즉, 음의 입력일 경우 하위 4비트는 입력 피연산항과 동일하게 하고, 이를 제외한 상위 비트들은 영상 크기 정보를 저장하고 있는 레지스터의 하위 4비트를 제외한 상위 비트들에서 1을 뺀 비트형태를 출력해야 한다. 그림 7의 A는 영상 크기 레지스터의 하위 4비트를 제외한 상위 비트들에서 1을 뺀 값을 표시하고 있다. A값을 출력하기 위한 회로는 영상 크기 레지스터에 저장된 값이 변할 때만 동작하게 되는 정적인 회로이며, 일반적인 뺄셈기가 아니라

영상 크기 레지스터의 일부 비트열에서 고정된 1의 값을 빼는 단순한 회로를 통해 구현이 가능하다.

그림 8은 잘라내기를 위한 하드웨어 구조이다. 잘라내기를 위한 구조에서도 피연산항이 가능한 입력 범위를 초과 했는지 결정하기 위해 xor 로직이 사용된다. 만약 피연산항이 가능한 입력 범위를 초과한 경우는 피연산항이 가질 수 있는 최대값을 출력하고 그렇지 않은 경우 피연산항이 그대로 출력된다. 이때 피연산항이 가질 수 있는 최대값을 출력하기 위해서는 별도의 저장 공간을 사용하거나 영상 크기 레지스터에 저장된 값을 이용할 수 있다. 영상 크기 레지스터에 저장된 값을 이용해야 할 경우 피연산항이 가질 수 있는 최대값은 영상 크기에서 1을 뺀 값이므로 이를 위한 별도의 뺄셈기가 요구 된다. 이때 영상 크기 레지스터에 저장된 값은 앞선 가정에 따라 하위 4비트가 모두 0이므로 그림 8에서와 같이 이를 제외한 상위 비트에서 1을 뺀 A 비트열에 "1111"의 비트열을 연결시킴으로써 구현 가능해 진다.

한편 잘라내기를 위한 동작에서 입력 피연산항이 음일 경우 부호비트를 제어 신호로 사용하여 음의 입력일 경우 영상 크기의 최소값인 0을 출력하게 된다. 그림 7과 8에서 순환 동작과 잘라내기를 위한 구조는 각각 따로 설명되었지만, 순환 동작과 잘라내기가 모두 요구되는 프로세서에 있어서 두 구조는 간단히 결합되어 질 수 있다.

2. 이차원 어드레스 지정을 위한 하드웨어 구조

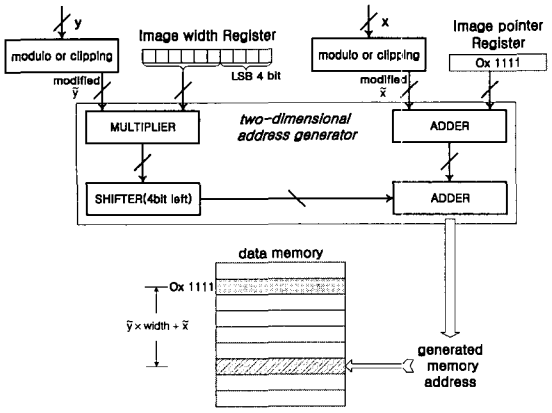


그림 9. 이차원 어드레스 지정을 위한 하드웨어 구조
Fig. 9. Hardware architecture for 2-dimensional addressing mode.

그림 9는 이차원 어드레스 지정을 위한 하드웨어 구조를 도시하고 있다. II장 1절에서 기술된 바와 같이 입력된 피연산항 x와 y는 각각 영상 크기 정보를 가지고 있는 레지스터 값과 비교되어 잘라내어지거나 순환되어지고, 영상 크기의 한계를 벗어나지 않도록 $\tilde{x}$와 $\tilde{y}$로 각각 수정된다. 수정된 $\tilde{x}$와 $\tilde{y}$를 이용하여 해당 화소가 저장된 메모리 위치를 구하기 위해서는 영상이 저장되기 시작한 메모리의 위치 뿐만 아니라 별도의 부가적인 연산이 필요하다. 그림 9에서 영상이 저장되기 시작하는 메모리의 위치는 영상 지정 레지스터(image pointer register)에 저장되어 있으며 주소 발생기는 화소가 저장된 위치를 계산하게 된다.

영상이 순차주사 방식으로 메모리에 저장되어 있는 일반적인 경우에 있어서 수평방향으로 $\tilde{x}$, 수직 방향으로 $\tilde{y}$에 해당하는 메모리 위치를 구하기 위해서는 *image pointe register*+ $\tilde{y}\times$*image width*+ $\tilde{x}$의 연산을 수행해야 한다. 그림 9의 이차원 주소 발생기는 잘라내기 또는 순환동작에 이어서 이러한 연산을 수행하여 해당 화소가 저장된 메모리의 주소를 구하게 된다. 이때 $\tilde{y}\times$*image width*를 계산하는데 있어서 영상 크기가 항상 16의 배수이므로 영상 크기 레지스터의 하위 4비트를 제외한 상위비트들을 $\tilde{y}$값과 곱한 후 이 결과를 4비트 천이함으로써 간단히 구현할 수 있다. 이를 통해 적은 전력과 하드웨어적인 비용으로 곱셈기를 구현할 수 있게 된다.

예를 들어 HDTV(High Definition Television)를 수용하는 MPEG-2의 메인 프로파일(main profile)과 하이 레벨(high level)의 영상을 처리하기 위한 프로세서는 1920x1152의 해상도를 가지는 영상을 처리해야 한다. 이러한 해상도를 수용하기 위하여 영상 크기 레지스터는 최소 11비트 저장 공간이어야 하므로, HDTV에서와 같은 큰 영상을 처리해야 하는 응용에 있어서도 그림 9의 곱셈기는 7×11의 크기를 가지면 된다. 한편 이동통신 단말기 등에서 화상통신의 응용에 QCIF 영상이 사용될 경우 해당 곱셈기는 4×8의 크기를 가지면 된다.

요약하면 이차원 어드레스 지정 기법을 사용하기 위해서는 다음과 같은 부가적인 하드웨어가 요구된다. 먼저 잘라내기와 순환 동작을 위해 xor 로직과 선택기가 요구되고 주소 발생을 위한 하나의 곱셈기와 두개의 덧셈기가 추가로 요구된다. 그림 9에 도시되어있는 4비트 천이기는 항상 고정된 비트를 천이시키는 동작을

수행하므로 실제 천이기를 사용하지 않고 버스 라인의 배열을 통해서 동일한 동작을 수행하게 할 수 있다.

Ⅳ. 이차원 메모리 어드레스 지정 기법의 성능

본 절에서는 이차원 메모리 어드레스 지정 기법을 간단한 RISC 프로세서에 적용했을 때 얻을 수 있는 장점을 분석하고자 한다. 가정된 RISC 프로세서는 참고 문헌 [6]에 기술된 DLX라고 명명된 아키텍쳐를 가진 프로세서 이다. DLX-RISC 프로세서는 간단한 load-store 명령어 집합을 가지고 있으며 파이프라인 시 효율성을 극대화하기 위한 구조를 가지고 있다. 특히, DLX와 유사한 형태의 RISC 프로세서가 매우 대중적이고, 비교적 이해하기 쉬운 아키텍쳐를 가지고 있기 때문에 본 논문에서는 DLX 프로세서에 이차원 메모리 어드레스 지정 기법을 적용했을 때 얻을 수 있는 성능 향상에 대하여 살펴 본다.

제안된 기법을 DLX-RISC 프로세서에 사용함으로써 얻을 수 있는 장점을 분석하기 위해 영상 처리에서 빈번히 발생하는 저대역 필터링 과정과 영상 복호기에 필수적인 움직임 보상 과정을 이용하였다. 먼저 Verilog HDL을 이용하여 이차원 어드레스 지정 기법이 포함되도록 DLX-RISC 프로세서를 레지스터 천이 레벨(RTL:register transfer level)로 기술하였다. 그리고, 각각의 영상 처리 알고리즘을 제안된 어드레스 지정 기법을 사용하지 않고 어셈블리 수준에서 최적화 코딩한 후 각각의 명령어에 대한 빈도수와 이의 수행 시간을 시스템 클럭으로 측정하였다. 명령어에 대한 빈도수를 측정하기 위해서 카운터의 역할을 수행하는 변수를 정의하고 각 명령어의 실행 파이프라인 단계에서 이를 증가시키는 부가 코드를 삽입하였다. 각 알고리즘에 대한 어셈블리 코딩 결과를 기계어로 수동적으로 변환한 후 Verilog HDL 시뮬레이터를 통해 각 명령어에 대한 빈도수와 이의 수행에 걸린 시스템 클럭을 출력하였다. 이때 시스템 클럭은 DLX-RISC 프로세서에서 발생하는 각종 해저드에 의한 영향이 반영된 것이다. 동일한 과정을 이차원 어드레스 지정 기법을 사용한 경우에 대해서도 적용하여 명령어 빈도수와 수행 시간을 측정한 후 이를 비교하였다.

그림 10과 11은 제안된 기법을 DLX-RISC 프로세서에 사용함으로써 얻을 수 있는 수행 시간의 이점을 도시하고 있다. 그림 10은 필터링 과정을 그림 11은 움직임 보상 시 발생하는 각 명령어의 발생 횟수를 이차원 메모리 어드레스 지정 기법을 사용한 경우와 그렇지 않은 경우에 대하여 비교하고 있다. 필터링과 움직임 보상 각 과정은 DLX-RISC 프로세서의 명령어 집합을 사용하여 어셈블리 레벨에서 명령어 수행 횟수가 최소화 될 수 있도록 작성하였다. 그림에서 도시된 바와 같이 필터링 과정과 움직임 보상 시 이차원 어드레스 지정 기법을 사용함으로써 각각 29%와 33% 정도의 수행 명령어 수를 감소 시킬 수 있다. 표 1은 그림 10과 11에 표시된 각 명령어에 대한 동작을 간략히 설명하고 있다. 여기서 각 명령어는 비교 분석의 용이함을 위해 어드레싱 모드 또는 오퍼랜드의 데이터 크기등에 따라서는 분류되지 않은 것이다.

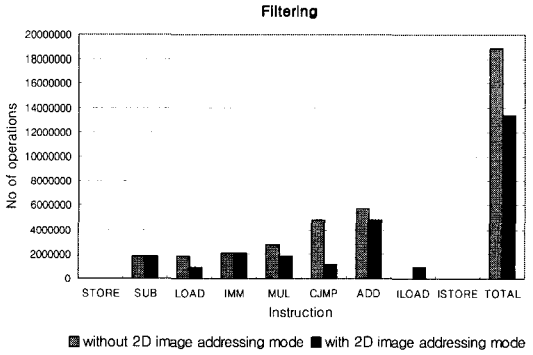


그림 10. 이차원 필터링 시 수행 명령어 수 비교
Fig. 10. Comparison of the number of operation in the case of 2-D filtering.

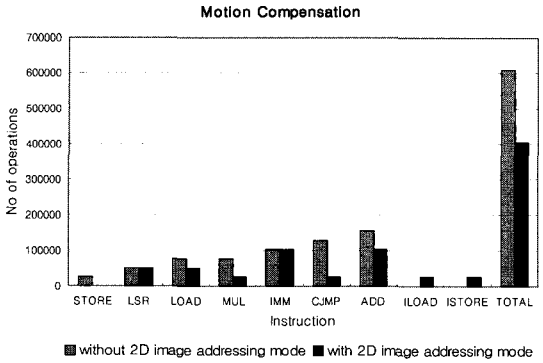


그림 11. 움직임 보상 시 수행 명령어 수 비교
Fig. 11. Comparison of the number of operation in the case of motion compensation.

표 1. DLX-RISC 프로세서의 명령어 일부
Table 1. Partial list of the instruction in DLX-RISC.

	명령어	기술
데이터 이동	LOAD	load byte, half word, or word
	STORE	store byte, half word, or word
	ILOAD	proposed two-dimensional load
	ISTORE	proposed two-dimensional store
	IMM	load immediate value
산술/논리 연산	ADD	add; signed or unsigned
	SUB	subtract; signed or unsigned
	MUL	multiply; signed or unsigned
	LSR	logically shift right
분기	CJUMP	conditional branch

그림 10에서 분석된 필터링 과정은 QCIF(176×144) 영상에 5×5의 크기에 해당하는 윈도우를 가진 저대역 통과 필터이다. 이러한 필터링 과정은 하나의 전체 화면을 화소별로 이동하면서 윈도우 내의 데이터를 읽어 들이고 이를 처리하게 되는데, 이때 이러한 사각형 형태로 데이터를 읽어 들이기 위해서는 많은 부가적인 연산이 요구된다. 즉, 대부분의 필터링 과정에서 요구되는 필터 계수에 각각의 데이터를 곱해서 누적하는 연산 외에 이차원 형태로 데이터를 읽기 위해 부가적으로 연산이 요구된다는 것이다. 또한, 화면의 경계에 이러한 윈도우가 놓여지게 되면 보다 많은 부가 연산과 처리가 요구된다. 그림 10의 결과를 통해서 알 수 있는 바와 같이 이차원 메모리 어드레스 지정 기법을 사용하지 않음으로써 이차원 형태로 데이터를 읽기 위해서 상대적으로 많은 분기 명령어를 사용해야 함을 알 수 있다. 아울러 해당 화소가 저장된 메모리의 주소를 계산하기 위하여 곱셈과 덧셈 등의 부가적인 산술연산이 추가로 요구됨을 알 수 있다.

그림 11의 움직임 보상 과정은 QCIF 영상의 각 매크로 블럭(16×16)에 대하여 움직임 보상을 수행하기 위해 요구되는 명령어 수를 도시하고 있다. 이때 보상하고자 하는 화소가 영상 경계 밖에 있는 화소일 수도 있으므로 이를 처리하기 위해 분기 명령어가 추가로 요구 된다. 이차원 메모리 어드레스 지정 기법을 사용함으로써 이러한 분기 명령어의 수를 줄이고 아울러 메모리 상에 화소의 위치를 구하기 위한 부가적인 연산을 줄일 수 있다.

분기 명령어의 추가 사용은 그 자체로 수행해야 할 명령어 숫자를 증가시킬 뿐만 아니라 파이프라인 시 추가적인 제어-해저드(control hazard)를 야기시키므로 프로세서의 성능을 크게 저하시키게 된다. DLX-RISC 프로세서는 5단계의 파이프라인으로 구성되며 분기 명령어의 경우 제어-해저드에 의해 하나의 클럭 사이클이 추가로 요구 된다[6]. 표 2는 DLX-RISC 프로세서에 앞서 설명한 필터링 과정과 움직임 보상 과정을 수행할 때 요구되는 클럭 사이클을 제안된 방법을 사용했을 경우와 그렇지 않을 경우에 대하여 각각 비교 하고 있다. 표 2의 결과로부터 이차원 메모리 어드레스 지정 기법을 사용함으로써 필터링시 약 23%, 움직임 보상시 약 41%의 수행 시간 감소 효과를 얻을 수 있음을 알 수 있다.

표 2. 수행시간 비교
Table 2. Comparison of execution time.

	이차원 메모리 어드레스 지정 기법을 사용하지 않는 경우	이차원 메모리 어드레스 지정 기법을 사용하는 경우	수행 시간 감소 효과
필터링	23,663,809 (clock)	18,121,537 (clock)	23%
움직임 보상	738,553 (clock)	431,425 (clock)	41%

V. 결 론

본 논문에서는 기존 메모리 구조를 바꾸지 않고 메모리에 저장된 영상데이터 값을 이차원적인 개념으로 읽어들이거나 저장하는 새로운 어드레싱 기법을 제안하였다. 이러한 이차원 명령어는 전체적인 코드 길이를 줄여 하드웨어의 수행 시간을 감축 시킬 뿐 아니라, 이차원 데이터를 읽거나 저장하는데 필요한 여러 명령어를 통합함으로써 하드웨어 구현 시 전력감소의 효과를 제공한다. 한편, 본 논문에서는 이러한 이차원 메모리 어드레싱을 효율적으로 구현하기 위한 방안을 제안하였다. 즉, 영상 폭과 높이가 저장되는 별도의 레지스터를 사용하고, 영상 크기가 항상 16의 배수라는 것과 피연산자의 입력을 일정 범위로 제한 시키는 구조를 제안함으로써 이차원 어드레싱 기법이 적은 하드웨어 비용으로 구현 가능함을 제시하였다. 끝으로 제안된 어드레싱 기법을 RISC 프로세서에 적용하는 모의 실험을 통해 영상 처리에서 빈번히 발생하는 필터링과 움직임

보상 과정을 수행하는데 요구되는 시간이 각각 23%, 41% 줄어듦을 보였다.

이와 같은 이차원 메모리 어드레스 지정 기법은 각종 DSP, media processor, 그래픽 장치 등에 응용되어 고성능 영상처리시스템의 수행시간을 감축 시키고, 무선 멀티미디어 응용에서 저전력 요구를 만족시킬 수 있을 것이다.

참 고 문 헌

[1] K. Aono, M. Toyokura, T. Araki, A. Ohtani, H. Kodoma, and K. Okamoto, "A Video Digital Signal Processor with a Vector-Pipeline Architecture," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 12, pp. 1886-1894, December 1992.

[2] H. Igura, Y. Naito, K. Kazama, I. Kuroda, M. Motomura, and M. Yamashina, "An 800-MOPS 110-mW, 1.5-V, Parallel DSP for Mobile Multimedia Processing," *IEEE Journal of Solid-State Circuits*, vol. 33, no. 11, pp. 1820-1828, November 1998.

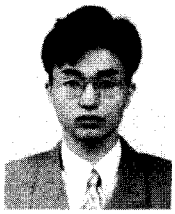
[3] K. Okamoto, T. Jinbo, T. Araki, Y. Iizuka, H. Nakajima, M. Takahata, H.Inoue, S. Kurohmaru, T. Yonezawa, and K. Aono, "A DSP for DCT-Based and Wavelet-Based Video Codecs for Consumer Applications," *IEEE J. Solid-State Circuits*, vol. 32, no. 3, pp. 460-467, March 1997.

[4] R. Chellappa, B. Girod, D. C Munson Jr., A. Murant Tekalp, and M. Vetterli, "The Past, Present, and Future of Image and Multidimensional Signal Processing," *IEEE Signal Processing Magazine*, pp. 21-58, March 1998.

[5] ITU-T Telecom. Standardization Sector of ITU, "Video Coding for Low Bit Rate Communication," *ITU-T Recommendation H.263*, March 1996.

[6] J. L. Hennessy, and D. A. Patterson, *Computer Architecture and Quantitative Approach*, Morgan Kaufmann, pp, 96-360, 1996.

저 자 소 개



高 綸 浩(正會員)

1973년 1월 4일 생. 1995년 부산대학교 전자공학과 졸업(공학사). 1997년 한국과학기술원 전기 및 전자공학과 졸업(공학석사). 1997년~현재 한국과학기술원 전자전산학과(전기 및 전자공학 전공) 박사과정 재학. 주 관심분야는 영상신호처리, MPEG-4, 영상시스템, 워터마킹 등

尹 炳 珠(正會員)

1970년 6월 30일 생. 1993년 경북대학교 전자공학과 졸업(공학사). 1996년 한국과학기술원 전기 및 전자공학과 졸업(공학석사). 1996년~현재 한국과학기술원 전자전산학과(전기 및 전자공학 전공) 박사과정 재학. 주 관심분야는 영상신호처리, MPEG-4, 객체기반 영상부호화, 객체 모양정보 부호화

金 聖 大(正會員)

1977년 서울대학교 전자공학과 졸업(공학사). 1979년 한국과학기술원 전기 및 전자공학과 졸업(공학석사). 1983년 프랑스 INPT ENSEEIHT 졸업(공학박사). 1984년~현재 한국과학기술원 전자전산학과(전기 및 전자공학 전공) 교수. 주 관심분야는 영상처리, 영상통신, 컴퓨터 비젼, VLSI 구현 등