영상처리용 프로세서를 위한 이차원 어드레스 지정 기법

고윤호, 조경석, 김성대
한국과학기술원 전기및전자공학과
전화 : (042)869-5430 / 팩스 :(042)869-8570

An Efficient 2-dimensional Addressing Mode for Image Processor

Ko Yun-Ho, Cho Kyung Suk, Kim Soeng-Dae
Electronics and Electrical Engineering of KAIST
E-mail : kyh@sdvision.kaist.ac.kr

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. Combining several instructions to load a pixel-data from an external memory to a register, the proposed instruction reduces code size so that satisfy 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.

I. 서론

영상 정보 매체는 음성이나 문자 등과 같은 다른 형태의 정보 매체에 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 2 차원 형태의 신호로 구성되어 있기 때문에 음성이나 텍스트와 같은 다른 종류의 신호들과 그 성격이 다르다고 할 수 있다.

따라서, 영상 정보는 앞서 언급한 바와 같은 데이터 고유의 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 형태의 데이터 처리가 요구됨에도 불구하고 기존의 DSP 나 마이크로프로세서는 메모리에 저장된 데이터를 일차원적으로 처리한다[1][2]. 즉, 실제 발생하는 영상 데이터가 이차원 형태임에도 불구하고, 구현상의 편의와 기존에 DSP 나 마이크로프로세서가 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.

본 논문에서는 기존의 메모리 구조를 바꾸지 않고 메모리에 저장된 영상데이터 값을 이차원적인 개념으로 읽어들이거나 저장하는 명령어를 제시하였다. 아울러 이러한 이차원 메모리 어드레스 지정 기법을 구현하기 위한 하드웨어 구조를 제시한다.

II. 본론

1. 이차원 어드레스 지정 기법의 개념

그림 1 은 ILOAD(image load)라는 새로운 명령어의 동작을 설명하고 있다. 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, 지금까지 사용된 프로세서는 대개 이차원 영상 데이터를 일정한 주사 방법에 따라 일차원으로 나열하고 이를 음성과 같은 일차원 데이터와 동일한 방식으로 처리해 왔다. 여기에 대비해 ILOAD 는 영상 데이터를 이차원적으로 처리한다.

그림 1 에서 x 와 y 는 영상데이터가 저장된 메모리의 위치를 지정하기 위한 두개의 피연산자로서 각각 영상의 수평 방향과 수직 방향으로의 위치를 지정한다. 여기서 x 와 y 는 각각 영상의 폭과 높이를 벗어나서는 안 된다. 그러나 실제 영상처리 프로그램에서는 x 와 y 가 이러한 제약을 벗어나는 경우가 발생하고, 이를 방지하기 위해 기존의 방법들은 부가적인 코드를 사용하고 있다. 따라서, x 와 y 가 영상 크기의 한계를 벗어나지 않도록 적절한 처리를 해야 하는데, x 와 y 가 영상 사이즈를 벗어나는 경우 주로 잘라내거나(clipping) 순환(modulo)시켜 (x) 와 (y) 로 값을 각각 수정한다. 이를 위해서 영상 크기에 대한 정보가 요구되는데, 이를 위해 영상 폭 레지스터(image width register)와 영상 높이 레지스터(image height register)를 이용한다.

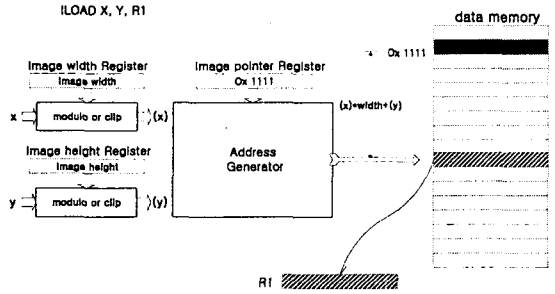


그림 1. ILOAD 의 동작 개념을 위한 블록도

그림 1 에서 주소 발생기(*address generator*)는 변형된 (x)와 (y)값을 이용해 처리하고자 하는 화소에 대응하는 메모리 위치로부터 저장된 값을 읽기 위한 동작을 하게 된다. 이때 (x)와 (y)값 뿐만 아니라 메모리 전체에서 해당 영상 데이터가 시작되는 위치를 알아야 한다. 이를 위해서 별도의 저장장치가 있어야 하고, 그림 1 에서는 영상 지정 레지스터(image pointer register) 가 메모리상에서 영상 데이터가 시작되는 주소를 저장하게 된다.

2. 이차원 어드레스 지정 기법의 필요성

영상은 수직 방향과 수평 방향으로의 성분을 지니는 이차원 데이터라고 볼 수 있다. 그런데, 영상처리 알고리즘은 C 와 같은 상위 단계의 소프트웨어에서는 이차원적으로 구현되지만 하드웨어는 이를 일차원적으로 처리한다. 따라서, 이차원적으로 처리되는 영상 데이터를 하드웨어를 통한 구현에서도 이차원 개념을 도입한다면 그 성능을 극대화 할 수 있을 것이다. 그림 2 는 제안된 메모리 어드레스 지정 기법이 사용될 수 있는 영상처리 알고리즘들 중 영상처리에서 필수적인 이차원 필터링 과정과 동영상 움직임 보상과정을 보여주고 있다.

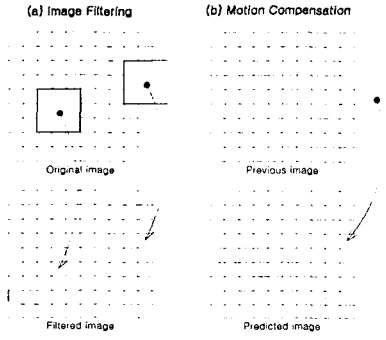


그림 2. 제안된 메모리 어드레스 지정 기법을 적용할 수 있는 영상처리 알고리즘

그림 2 의 (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[3].예를 들어 그림 2 에서와 같이 화면의 경계에 윈도우가 놓여질 수 있는데, 이 경우 윈도우 내에는 있지만 영상 밖에 존재하는 데이터에 대한 처리를 해야 하는데. 이때 이차원 메모리 어드레스 지정 기법을 이용해 효율적으로 이를 수행할 수 있다.

그림 2 의 (b)는 움직임 보상과정을 나타내고 있다. H.263 의 비제한적임 움직임 벡터 모드(unrestricted motion vector mode)에서와 같은 경우 부호화된 이전 영상의 경계에 있는 화소들을 외부로 확장함으로써 외부에 있는 화소들을 참조할 수 있다[4].

<Assembler code containing proposed Image LOAD >

```
%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
%Current Image ptr address 00000H
%R11 : current pixel Memory Address
.
.
.
for( all current image pixel  R3, R4 ){
        ADD R3, R1, R5          %x_pred = x + motion_vector_x
        ADD R4, R2, R6          %y_pred = y + motion_vector_y
        ILOAD R5, R6, A
        STORE A, @R11
}
```

그림 3. 움직임 보상을 수행하기 위한 기존 프로세서의 어셈블리 코드

<Convetional Assembler 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
%Current Image ptr address 00000H
%R7, R8 : Image width & Image height
%R11 : current pixel Memory Address
.
.
.
for( all current image pixel  R3, R4 ){
        ADD R3, R1, R5          %x_pred = x + motion_vector_x
        ADD R4, R2, R6          %y_pred = y + motion_vector_y
        BCND Path1, R5, le. R7          %x_pred <= Image_width
        LOAD R7, R5                     %x_pred = Image_width
Path1 : BCND Path2, R6, le. R8          %y_pred > Image_height
        LOAD R8, R6                     %y_pred = Image_height
Path2 : MUL R9, R5, R7                  %x_pred * Image_width
        ADD R10, R9, R6         %x_pred * Image_width + y_pred
        ADD R9, R10, #0AAAH
        LOAD @R9, A
        STORE A, @R11
}
```

그림 4. 이차원 메모리 어드레스 지정 기법을 이용한 움직임 보상의 어셈블리 코드

그림 3 과 4 는 각각 제안된 기법이 사용되지 않은 경우와 사용된 경우에 대한 움직임 보상 과정의 어셈블리 코드를 나타내고 있다. 그림 3 의 코드에서 제안된 기법을 사용하지 않고 움직임 보상을 수행하기 위해서는 영상 경계 밖에 있는 화소를 처리하기 위해 그림 4 의 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. 한편 그림 3 의 코드에서는 분기 명령어를 통해 움직임 보상 하고자 하는 화소를 결정한 후 그 데이터가 저장된 메모리의 값을 읽기 위해 한번의 덧셈과 곱셈이 추가적으로 사용된다.

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

이차원 메모리 어드레스 지정 기법을 위해서는 두 개의 피연산항을 영상의 폭과 높이에 따라 잘라내거나 순환시켜야 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.

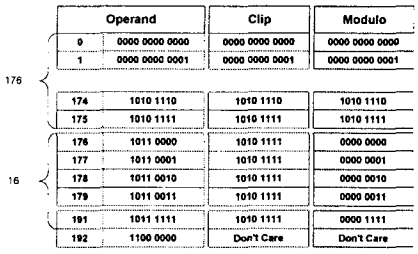
실제로 피연산항이 영상크기 보다 큰지 여부를 결정하기 위해서는 비교기를 쓰지 않을 수 없다. 그러나, 영상 크기가 가지는 특성과 피연산자가 가질 수 있는 값의 변화 범위를 고려하면 비교기를 쓰지 않고 동일한 결과를 발생시킬 수 있다. 먼저, 영상처리에서 처리해야 하는 데이터의 크기는 항상 16 의 배수이다. 실제로 QCIF(176×144), CIF(352×288), SIF, CCIR601 과 같이 표준화되어있는 영상 규격들은 모두 그 폭과 높이가 16 의 배수이다

한편 피연산항으로 입력되는 x 와 y 값이 최대 영상 크기에 16 을 더한 값 보다 작다고 가정할 수 있다. 이는 그림 2 의 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 와 같이 지나치게 큰 윈도우를 사용하지 않으므로 이러한 가정은 타당하다. 또한 그림 2 의 비제한적인 움직임 보상 모드에서도 영상 경계 가장 자리 값을 확장하므로 피연산항 x 와 y 값이 영상 최대 값에서 16 을 초과할 수 없다.

여기서, 다음의 사실을 알 수 있다. 먼저 처리하고자 하는 영상의 크기가 항상 16 의 배수이므로 입력된 피연산항은 0 에서부터 2^4n-1 (여기서 n 은 자연수)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 2^4n 에서 2^4n+16 의 범위에 있는 수는 그 하위 4 비트는 다르지만, 하위 4 비트를 제외한 상위 비트들의 형태가 동일하다. 식 1 은 2^4n 에서 2^4n+15 범위의 값을 이진수로 표현한 것이다.

$$
\begin{aligned}
2^4n &= p^k2^k + p^{k-1}2^{k-1} + \cdots + 0\cdot2^3 + 0\cdot2^2 + 0\cdot2^1 + 0\cdot2^0 \\
2^4n+1 &= p^k2^k + p^{k-1}2^{k-1} + \cdots + 0\cdot2^3 + 0\cdot2^2 + 0\cdot2^1 + 1\cdot2^0 \qquad \text{(식 1)} \\
&\vdots \\
2^4n+15 &= p^k2^k + p^{k-1}2^{k-1} + \cdots + 1\cdot2^3 + 1\cdot2^2 + 1\cdot2^1 + 1\cdot2^0
\end{aligned}
$$

이처럼 부가적인 잘라내기와 순환동작이 요구되는 입력 범위[2^4n, 2^4n+16]에 있는 피연산항은 그 하위 4 비트를 제외한 상위 비트들의 형태가 동일함을 알 수 있다. 그림 5 는 영상의 폭이 176 화소이고 높이가 144 인 QCIF 영상을 처리하는 경우에 발생하는 피연산항 x 를 잘라내거나 순환시키는 동작을 도시하고 있다. 그림에서처럼 잘라내기와 순환동작이 요구되는 입력 범위 [176, 192]에 있는 피연산항은 그 하위 4 비트를 제외한 상위 비트들의 형태가 동일함을 알 수 있다.



	Operand	Clip	Modulo
	0 0000 0000 0000	0000 0000 0000	0000 0000 0000
	1 0000 0000 0001	0000 0000 0001	0000 0000 0001
176			
	174 1010 1110	1010 1110	1010 1110
	175 1010 1111	1010 1111	1010 1111
	176 1011 0000	1010 1111	0000 0000
	177 1011 0001	1010 1111	0000 0001
16	178 1011 0010	1010 1111	0000 0010
	179 1011 0011	1010 1111	0000 0011
	191 1011 1111	1010 1111	0000 1111
	192 1100 0000	Don't Care	Don't Care

그림 5. 영상의 폭이 176 화소인 영상에 대한 잘라내기와 순환동작의 예

한편 피연산항 중 2^4n 에서 2^4n+16 의 범위에 있는 수는 알고리즘에 따라 잘라내거나 순환시켜야 한다. 잘라내는 경우에 있어서는 이 범위에 해당하는 피연산항이 입력될 때 간단히 영상 크기의 최대값으로 바꾸어 주면 된다. 순환시키는 경우에 있어서는 그림 5 에서와 같이 하위 4 비트는 입력되는 피연산항과 동일하고, 이를 제외한 상위 비트들은 입력에 관계없이 0 의 비트열로 만들어 주면 된다. 이처럼 간단한 방법으로 순환을 시킬 수 있는 것은 영상크기와 피연산항의 입력에 대한 앞선 가정으로부터 기인하는 것이다.

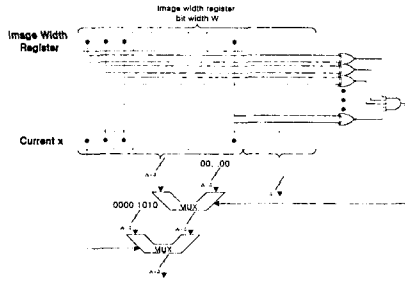


그림 6. 순환을 위한 하드웨어 구조

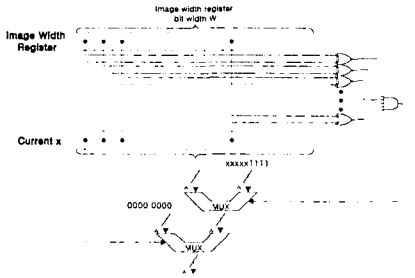


그림 7. 잘라내기를 위한 하드웨어 구조

그림 6 과 7 은 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.

4. 이차원 어드레스 지정을 위한 하드웨어 구조

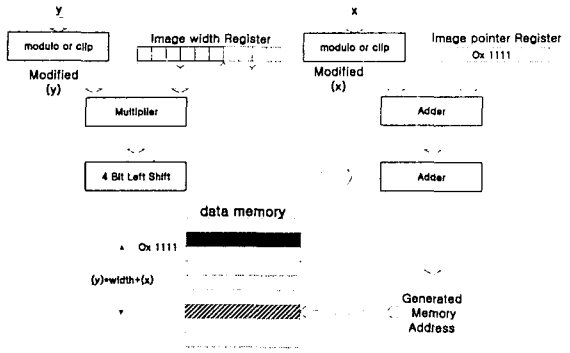


그림 8. 이차원 어드레스 지정을 위한 일반적인 하드웨어 구조

그림 8은 이차원 어드레스 지정을 위한 하드웨어 구조를 도시하고 있다. 입력된 피연산항 x와 y는 각각 영상 크기 정보를 가지고 있는 레지스터 값과 비교되어 영상 크기의 한계를 벗어나지 않도록 (x)와 (y)로 각각 수정된다. 수정된 (x)와 (y)를 이용하여 해당 화소가 저장된 메모리 위치를 구하기 위해서는 $*image\ pointer\ register + (y) \times image\ width + (x)$의 연산을 수행해야 한다. 이때 $(y) \times image\ width$를 계산하는데 있어서 영상 크기가 항상 16의 배수이므로 영상 크기 레지스터의 하위 4비트를 제외한 상위비트들을 (y)값과 곱한 후 이 결과를 4비트 천이함으로써 간단히 구현할 수 있다.

한편 주소 발생기의 구현을 보다 단순화 하기위해 곱셈 연산을 천이기(shifter)로 바꿀 수 있다. 즉 천이기를 사용한다는 것은 영상 폭의 크기 값인 2^4n을 곱하는 대신 2^4n에 가장 가까운 2^m을 곱하는 것과 동일하다. 이처럼 곱셈기를 천이기로 바꾸기 위해서는 메모리에 저장된 영상을 이러한 구조가 수용되도록 바꾸어야 한다. 즉, 영상 데이터가 순차 주사 방식으로 저장된다고 할 때 영상 폭에 해당하는 2^4n까지 영상 데이터를 저장하고 그 이후부터 2^m까지에 해당하는 메모리 영역은 사용하지 않아야 한다. 이러한 방식은 곱셈기를 천이기로 대체함으로써 이차원 주소 발생기의 구조를 간단히 할 수 있지만 일부 메모리를 사용할 수 없는 단점을 가지게 된다.

5. 이차원 어드레스 지정 기법의 성능 검증

그림 9와 10은 이차원 어드레스 지정 기법을 사용하는 경우와 그렇지 않은 경우에 수행되는 명령어의 수를 비교한 결과이다. 그림 9는 이차원 필터링을 수행할 경우에 발생하는 각 RISC 명령어의 빈도를 도시하고 있고 그림 10은 움직임 보상 시의 경우이다. 그림에서 도시된 바와 같이 이차원 어드레스 지정 기법을 사용함으로 해서 약 30% 정도의 수행 명령어 수를 감소 시킬 수 있다.

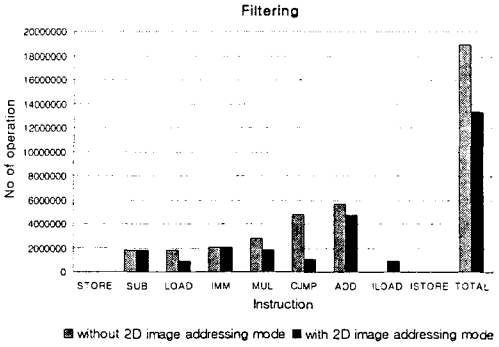


그림 9. 필터링 과정 시 이차원 어드레스 지정 기법의 성능

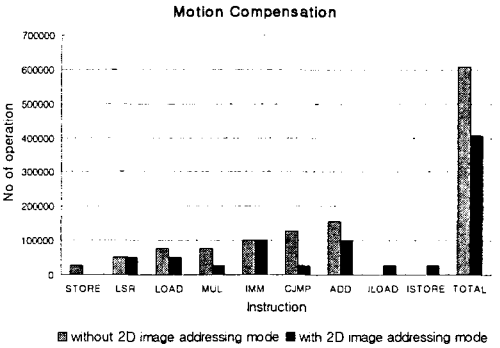


그림 10. 움직임 보상 시 이차원 어드레스 지정 기법의 성능

III. 결론

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

참고문헌

[1] K. Aono, *et al.*, "A video digital signal processor with a vector-pipeline architecture," *IEEE J. Solid-State Circuits*, pp. 1886-1894, Dec. 1992.

[2] H. Igura, *et al.*, "An 800-MOPS 110-mW, 1.5-V, parallel DSP for mobile multimedia processing," *IEEE J. Solid-State Circuits*, pp. 1820-1828, Nov. 19998.

[3] R. Chellappa, *et al.*, "The past, present, and future of image and multidimensional signal processing," *IEEE Signal Processing Magazine*, pp. 21-58, March 1998.

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