# Computer Arithmetic for the Processing of Media Signals

Vojin G. Oklobdzija[1], Aamir A. Farooqui[2]

[1]ACSEL
Electrical and Computer Engineering Department
University of California
Davis, CA 95616
USA
(510) 486-8171
vojin@ece.ucdavis.edu

[2]Synopsys® Inc.
Synopsys Module Compiler Group
700 Middlefield Road,
Mountain View
CA 94043-4033
USA
(650) 584-5689
aamirf@synopsys.com

## ABSTRACT

Media signal processing requires high computing power and the algorithms exhibit a great deal of parallelism on low precision data. The basic components of multi-media objects are usually simple integers with 8, 12, or 16 bits of precision. In order to support efficient processing of media signals, Instructions Set Architecture (ISA) of the traditional processors requires modifications. In this paper, we present the quantitative analysis and the computational complexity required to perform media processing. Main classes of instructions that are needed for the required level of performance of the Media Processor are identified. Their efficient implementation and effect on the processor data-path is discussed. The main operations required in media processing are Addition (with or without saturation), Multiplication (with or without rounding), Sum of Products, and Average of two numbers.

**Keywords:** ISA, SIMD, VLSI, Media Processor, MPEG, Media Arithmetic, IDCT, FFT, Addition, Multiplication.

## 1. INTRODUCTION

Media signal processing is the real time processing of audio and video signals. Real-time signal processing of audio/video signals is necessary in consumer electronic products, such as personal digital assistants, cellular phones, video games, digital cameras, and high-end communication products, such as IP-telephony gateways, multi-channel modems, speech-processing systems, echo cancellers, image and video processing systems, internet routers, and virtual private network servers. All real-time signal processing algorithms such as, coding/decoding, and compression/decompression of audio and video signals, are based around certain computation demanding functions such as the Discrete Cosine Transform (DCT), the Inverse Discrete Cosine Transform (IDCT), the Fast Fourier Transform (FFT). These functions are computationally intensive so that special dedicated VLSI circuits or Digital Signal Processors (DSPs) are used in conjunction with the main processor to support these functions. Programmable architectures provide a cost-effective alternative to dedicated VLSI, however, conventional DSP architectures lack the computing power and bandwidth to perform more than one multimedia task at a time. General-purpose microprocessors support media processing by introducing multimedia enhancements in their instruction sets. In these processors a long-word ALU is divided into several small-word ALUs, and several pieces of independent short-word data are processed by a single instruction to perform SIMD operation. This approach was first used for graphics instructions [1, 2] and gave birth to a new category of processors called Media Processor. A Media processor is defined as programmable processor dedicated to simultaneously accelerate the processing of multiple data types, including digital videos, digital audio, computer arithmetic, text and graphics. Media processors may operate as stand-alone or with a main processor incorporating features such as, peripheral communications interconnect (PCI) interface that actually support their integration onto a PC motherboard[3]. They have greater parallelism with lower clock frequencies of 50–200 MHz.

The media processors have an increased processing capability for video, but this multimedia enhancement makes the programming for the microprocessors much more complex. Efficient programming can only be attained if experts tune the software using assembly languages, just as in DSP approaches. There are many MPEG application-specific integrated circuit (ASIC) chips that contain a reduced instruction set computer (RISC) core for control and housekeeping purposes. The most successful approach right now belongs to these classes [4, 5, 6, 7, 8, 9]. Most of the major microprocessor architectures have included multimedia instructions in their Instruction Set Architecture (ISA), which supports MPEG-1 and MPEG-2 media standards. The HP PA-RISC was the first ISA to introduce multimedia extension, MAX-1 (Multimedia Acceleration eXtension), in 1994[10]. SUN followed shortly thereafter with the VIS (Visual Instruction Set) in the SPARC ISA[11, 12, 13]. In 1997, HP developed the second generation of MAX-1, called MAX-2, and incorporated it in its 64-bit PA processor[14]. In the

same year, Intel introduced the MMX (Multi-Media eXtension) in its Pentium processor line[15]. Both SGI and Digital have also announced or introduced multimedia instructions into their respective processor architecture[16]. Motorola's new AltiVec technology expands the capabilities of PowerPC microprocessors by providing a leading edge, general-purpose processing performance, while concurrently addressing high-bandwidth data processing and algorithmic-intensive computations in a single-chip solution[17].

This paper is organized as following: in Section 2 we present the computational complexity and the quantitative estimates to perform block-level MPEG decoding. Section 3 presents the implementation of Add operation and saturated arithmetic for media processing. Overview of different kinds of multiplication operations is presented in Section 4. Finally conclusions are presented in Section 5.

## 2. QUANTITATIVE ANALYSYS OF MPEG DECODING

In this section we present the computational complexity and the quantitative estimates to perform block-level MPEG decoding. Based on these estimates we describe new instructions and their implementation to efficiently support media processing. To perform quantitative analysis we made the following assumptions. We assume a bit rate of 4 Mbits/sec from the input bit stream, and the average symbol size is 4 bits, which corresponds to 1 million symbols per second. The frame size is 720 x 480 with 30 frames per second (MP@ML, main profile at main level) in a 4:1:1 YUV format. Therefore each frame contains 8100 (720 x 480 x 1.5), or 243,000, 8x8 micro blocks per second. In this performance analysis it is assumed that the data is already available in the registers (loaded by the main processor or load store unit) and cache hit rate is 100%, and instruction issue rate is one instruction/cycle.

Block-level image processing constitutes the major portion of existing image processing standards, such as H.261 and MPEG-1/2[18, 3]. Recent MPEG-4 standard offers a much broader range of functionalities and coding modes then the previously defined standards, but still the block-level processing consumes the major portion of MPEG-4 video coding[19]. As shown in Fig. 1 MPEG decoding requires the following six basic steps:

1. Header Decode.
2. Huffman and Run-length decode.
3. Inverse Quantization.
4. Inverse Discrete Cosine transform.
5. Motion compensation.
6. YUB to RGB conversion.

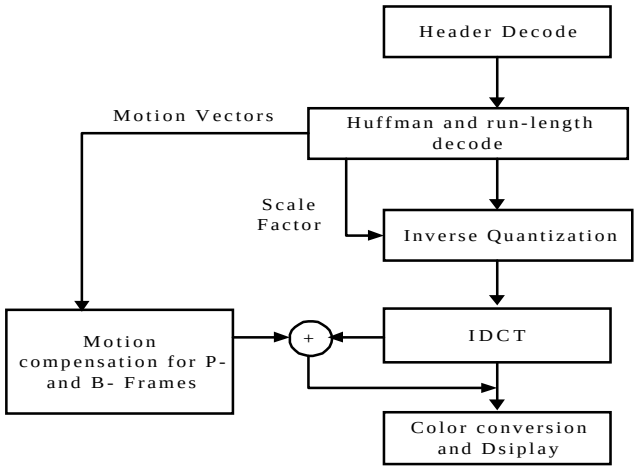


**Fig. 1.** Steps in MPEG Decoding.

### 2.1. Header Decode

**Operation**: Header decode provides information about video sequence parameters such as picture rate, bit rate, macroblock, and image size.

**Computational requirements**: This step takes minimum amount of computational resources due to the small amount of header information.

### 2.2. Huffman Decode

**Operation:** Huffman decoding, decodes variable-length numbers, which represent quantized inverse DCT (IDCT) coefficients, scaling factors, and motion vectors. Huffman decoding is a bit manipulation operation, and extracts code words from the compressed bitstream. This operation involves parsing of a variable-length code obtained from a bitstream buffer and symbol matching using code tables. A table lookup algorithm[20] can be used for this purpose. The lookup table consists of a zero-run (a number of zero coefficients), a level, and a code length. The table is addressed by the first few bits of the bitstream buffer. Then, the code length for the next table lookup shifts the buffer. This buffer operation is simulated in microprocessors that have word boundaries by using the comparison, shift, masking, and branch operations.

**Computational requirements:** The required performance is approximately proportional to the average number of symbols in a bitstream processed per second. If each operation is a single cycle instruction, then average number of arithmetic operations needed to decode one symbol (DCT coefficient) is around 11 (excluding load), it includes the following operations[21]:

- One load, one compare, one subtract, and one shift operation are required to get variable number of bits from the data buffer.
- Two shifts and one mask operation are required to search for a code in a table.
- One compare, one subtract, one shift, one add and one mask operation is required for data buffer update.

Hence, we require 11 MIPS (million instructions per second) to process a 4 Mbits/sec stream with 4 bits/symbol. Due to the sequential nature of the bitstream this operation is difficult to parallelize. Parallel symbol decoding is possible depending on the design of the code tables [22, 23].

Unlike other media operations, Huffman decoding is not speed up by SIMD-type parallel instructions. However, there are instructions, such as double-word shift, which can implement the parsing operation [24] efficiently.

### 2.3. Inverse Quantization

**Operation:** In inverse quantization the non-zero DCT coefficients are multiplied by quantization coefficients to restore them to their original range.

**Computational requirements:** Inverse quantization requires one load operation to read the quantization scale matrix, two integer multiplications and one divide by a constant number (8, 16) for de-quantization. Hence, the average number of arithmetic operations needed for inverse quantization of one DCT coefficient, are three.

If we assume that 80% of all the decoded symbols are DCT coefficients then we need 2.4 (3 x 0.8) MIPS for the processing of 4 Mbits/sec stream. If we assume 4-way parallel SIMD multiply and shift instructions then we require approximately 1 MIPS.

### 2.4. IDCT

**Operation:** Inverse discrete cosine transform changes each 8x8 block of DCT coefficients from frequency domain back to the original spatial domain. This gives the actual pixel values for I-Blocks, but only the differences for each pixel for P- and B- blocks. Inverse Discrete Cosine Transform (IDCT) is the most important function in MPEG and JPEG (Joint Photographic Experts Group) algorithms [25]. In MPEG standard the N x N 2-D IDCT is defined as [26]:

$$f(x,y) = \frac{2}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1}\alpha(u)\alpha(v)F(u,v)\cos\frac{(2x+1)\pi u}{2N}\cos\frac{(2y+1)\pi v}{2N} \qquad \text{Equation-1}$$

*where:*
*f(x,y)* is the pixel data associated with spatial coordinates x,y (x,y = 0,1……, N-1) in the time domain,
*F(u,v)* is the DCT coefficients with respect to coordinates u,v (u, v = 0, 1, …., N-1) in the frequency domain.
*C(u) = C(v) = 1/^2 for u,v = 0, else 1*

Using Equation 2-1 an N point 1D, IDCT can be represented by the following simplified equation 26.:

$$P = \sqrt{\frac{2}{N}}(TF) \qquad \text{Equation-2}$$

*where,*

P is the Nx1 pixel output vector,
F is the Nx1 DCT coefficient input vector, and
T is the NxN IDCT constant matrix.

For MPEG, N is normally 8, therefore we have $P = \frac{1}{2}(TF)$ in matrix form (after some transformations for repeatability and modularity) it can be written as:

$$\begin{bmatrix} p0 \\ p5 \\ p3 \\ p6 \\ p7 \\ p2 \\ p4 \\ p1 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} A & B \\ A & -B \end{bmatrix} \times \begin{bmatrix} F0 \\ F4 \\ F2 \\ F6 \\ F1 \\ F5 \\ F7 \\ F3 \end{bmatrix} \qquad \text{Equation-3}$$

*where:*

$$A = \begin{bmatrix} 0.707 & 0.707 & 0.923 & 0.382 \\ 0.707 & -0.707 & -0.382 & 0.923 \\ 0.707 & 0.707 & -0.923 & -0.382 \\ 0.707 & -0.707 & 0.382 & -0.923 \end{bmatrix} \quad B = \begin{bmatrix} 0.980 & 0.555 & 0.195 & 0.831 \\ -0.555 & -0.195 & -0.831 & 0.980 \\ 0.195 & 0.831 & -0.980 & -0.555 \\ -0.831 & 0.980 & 0.555 & 0.195 \end{bmatrix}$$

By decomposing Equation 2-3 into two 4x1 matrices we have

$$\begin{bmatrix} H0 \\ H1 \\ H2 \\ H3 \end{bmatrix} = \frac{1}{2}[A] \times \begin{bmatrix} F0 \\ F4 \\ F2 \\ F6 \end{bmatrix} \qquad \text{Equation-4}$$

and

$$\begin{bmatrix} K0 \\ K1 \\ K2 \\ K3 \end{bmatrix} = \frac{1}{2}[B] \times \begin{bmatrix} F1 \\ F5 \\ F7 \\ F3 \end{bmatrix} \qquad \text{Equation-5}$$

$$\begin{bmatrix} p0 \\ p5 \\ p3 \\ p6 \\ p7 \\ p2 \\ p4 \\ p1 \end{bmatrix} = \begin{bmatrix} H0+K0 \\ H1+K1 \\ H2+K2 \\ H3+K3 \\ H0-K0 \\ H1-K1 \\ H2-K2 \\ H3-K3 \end{bmatrix} \qquad \text{Equation-6}$$

**Computational requirements:** IDCT is a computationally intensive operation but it contains a great deal of parallelism in the form of matrix (usually 8 by 8) operations making it an interesting operation for SIMD processing.

An 8x8-point two-dimensional IDCT can be realized by using the 8-point one-dimensional (1D) IDCT for eight rows followed by eight independent, 1D IDCT on the columns. Hence, for 8,100, 8x8 blocks at 30 frames/second we need; 8,100x16x30 = 3.888 Million 1D IDCT. An 8-point one-dimensional IDCT that is realized through a single 8x8 matrix-vector multiplication requires 64 multiplications and 64 additions. Hence, 3.888 million 1D IDCTs require 248.8 MIPS. Several fast algorithms to minimize the number of multiplications have been proposed [27, 28, 29, 30, 31].

Multiply-accumulate instruction (MAC) [32, 18] dramatically reduces the number of instructions for IDCT execution. As we can see from Equation 4 - 6 each $H_i$ and $K_i$ requires four 16x16 multiply and accumulate operations as shown below:

$$Hi = A_{i0}F_0 + A_{i1}F_4 + A_{i2}F_2 + A_{i3}F_6 \qquad \text{Equation-7}$$

$$Ki = B_{i0}F_1 + B_{i1}F_5 + B_{i2}F_7 + B_{i3}F_3 \qquad \text{Equation-8}$$

Therefore, the calculation of each $H_i$ and $K_i$ requires four MAC operations. Finally, one ADD/SUB instruction is required to calculate each element of P, using $H_i$ and $K_i$. Hence, 40 operations are required to get a row of one-dimensional IDCT, and a total of 40x8 = 320 operations are required to get eight rows of one-dimensional IDCT. A matrix transposition (normally done through a shuffle instruction[3] ) is needed to start one-dimensional IDCT for eight columns. Hence, an average of 650 operations are required to perform 8-point IDCT and total complexity of 157.95 MOPS (million operations per second) is required to perform MPEG-2 MP@ML. If we assume 4-way parallel SIMD instructions then we require approximately 40 MIPS to perform MPEG-2 MP@ML.

The accuracy of the arithmetic operation is important in IDCT calculations. IEEE 1180 [33, 34] defines the accuracy required for IDCT to avoid degrading the quality of the decoded image. Truncation of the 32-bit multiplication result into 16 bits does not provide sufficient accuracy for IDCT. It has been reported, however, that sufficient accuracy can be obtained by shifting and rounding before truncating to 16 bits[7].

### 2.5. Motion compensation

**Operation:** MPEG video consists of three types of frames: I-frames, which are compressed without motion prediction; P-frames, which are compressed by unidirectional prediction using one previous I- or P-frame, and B-frames, which are compressed by bidirectional prediction using two frames (one past frame and one future frame). Typically, for each I-frame, there are four P-frames and ten B-frames.

Motion compensation is performed for P- and B-frames. In the reconstruction of P-frames the IDCT output is added with the pixels in the past reference block (the reference blocks are decoded in advance and are pointed by the motion vectors). This operation is normally referred as unidirectional motion compensation. In case of B-frames the IDCT output is added with the average of pixels in forward and backward reference blocks. This operation is called bidirectional motion compensation. When the motion vector is located between pixels, then pixel interpolation is needed to produce a prediction signal by interpolating between neighboring pixels. Fig. 2 shows possible locations of interpolation samples, denoted by $p_a$, $p_b$, $p_c$, $p_d$, and $p_e$. Their interpolated values are calculated as shown below:

$$p_a = (p_{(x,y)} + p_{(x+1,y)})/2 \qquad \text{Equation-9}$$

$$p_b = (p_{(x,y)} + p_{(x,y+1)})/2 \qquad \text{Equation-10}$$

$$p_c = ((p_{(x,y)} + p_{(x+1,y)}) + (p_{(x,y+1)} + p_{(x+1,y+1)}))/4 \qquad \text{Equation-11}$$

$$p_d = (p_{(x+1,y)} + p_{(x+1,y+1)})/2 \qquad \text{Equation-12}$$

$$p_e = (p_{(x,y+1)} + p_{(x+1,y+1)})/2 \qquad \text{Equation-13}$$

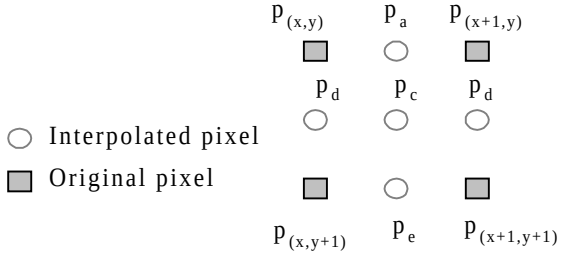


**Fig. 2.** Pixel interpolation in motion compensation.

**Computational requirements:** Unidirectional half-pixel motion compensation requires at most one addition to add a DCT coefficient and with the pixels in the past reference block, a total of three add operations for pixel interpolation, and one shift operation for averaging, and one 8-bit clipping. That means 384 (6x64) operations for one 8x8 block. Bidirectional half-pixel motion compensation requires eight add operations, three shift operations, and one 8-bit clipping, which result in 768

operations for one block. If we have I, P, and B frames in the ratio 1:2:5, then we require maximum of 159.9 ((384x0.285 + 768x0.714) x 24300) MOPS for MPEG-2 MP@ML.

In case of SIMD operation, the addition of two-pixel data for interpolation requires more than 8-bit arithmetic therefore 16-bit operations are used for this purpose. Assume a 4-way parallel SIMD unidirectional motion-compensation for 4-pixel data, it requires two 8- to-16 conversions, four split additions, one split shift, and one packing operation. This results in 128 operations for an 8x8 block, three times faster than the motion compensation without SIMD-type operations. Similarly, bidirectional motion compensation operations for 4-pixel data require four 8-to-16 conversions, eight split additions, three split shifts, and one 8-bit clipping. Which is 256 operations for an 8x8 block, or 50 MIPS for MPEG-2 MP@ML, this is also three times faster than the implementation without SIMD-type instructions. Some multimedia processors [14, 17] have SIMD-type multimedia instructions for this pixel averaging operation. In [24, 52] it has been reported that the split-word-type operation can be simulated using non split-word instructions with a special treatment of the carry propagation between each byte in a word.

### 2.6. YUV to RGB color conversion

**Operation:** YUV to RGB color conversion, converts color from YUV co-ordinates to RGB color co-ordinates, including up-sampling of U and V values, and writing to the frame buffer for displaying the decoded video. The main operation in this conversion is a 3x3 transform as shown below:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.402 \\ 1 & -0.344 & -0.714 \\ 1 & 1.772 & 0 \end{bmatrix} \times \begin{bmatrix} Y \\ U-128 \\ V-128 \end{bmatrix} \qquad \text{Equation-14}$$

**Computational requirements:** YUV to RGB color conversion requires 4 integer multiplications and 6 additions for each pixel. Hence, the average number of arithmetic operations needed for 720x480 pixel YUV to RGB color conversion are 3.256 or 103.68 MOPS for 30 frames per second. If we assume 4-way parallel SIMD multiply and add instructions then we require approximately 41.47 MOPS.

A summary of the quantitative estimates of the arithmetic operations to perform block-level MPEG MP@ML decoding is presented in Fig. 3. It is clear from these estimates, that the software-only approach to MPEG decoder requires a processing power of 424 MIPS on processors without media enhancement, while SIMD media instructions reduce the MPEG execution time by approximately three times (143.3 MIPS). The major time consuming operation in MPEG decoding is motion compensation (49.8 MIPS) followed by the display step (41.47 MIPS), and then IDCT (40 MIPS). Fortunately, the two inherently serial steps, decoding the MPEG headers and the Huffman decoding (12 MIPS), are relatively insignificant in execution time [3, 21]. The actual MPEG decoder implementation requires a higher processing power due to factors such as cache misses, memory access time, and bus width. However, the number of operations for motion compensation processing is reduced if no pixel interpolation (a non-half-pixel bitstream) is required.

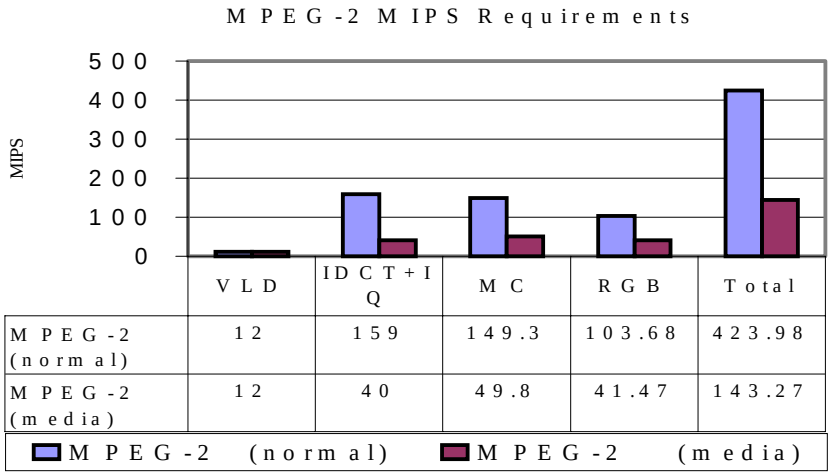


**Fig. 3.** MPEG-2 MP@ML execution profile with and without media instructions.

Since, the major arithmetic operations in MPEG decoding are addition, and multiplication thus, in order to speed-up the IDCT, motion compensation, and color conversion processing, new instructions have been developed to support these operations. Following we explain the implementation of these instructions considering a 64-bit datapath containing two 64-bit SIMD adders, and 32x32 SIMD multiplier. Fig. 4 shows the macro-level implementation of the two cycle datapath, containing four 16x32 multipliers, 3x2 compressors, and two 64-bit adder-subtractors. This datapath can easily be implemented in VLSI using datapath design tools such as, Synopsys Module Compiler™ [35].

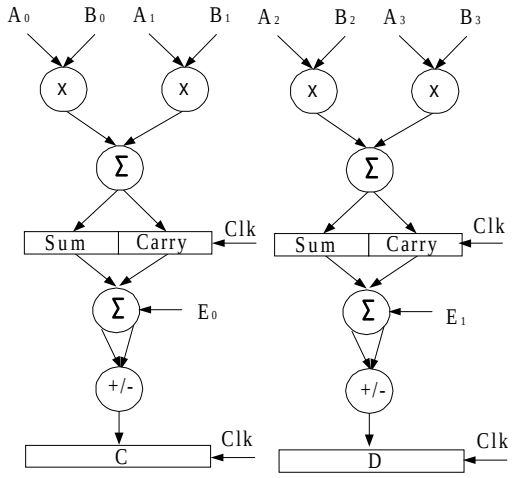


**Fig. 4.** Macro-level implementation of the datapath.

## 3. ADD/SUB INSTRUCTIONS

The Add/Sub instructions are executed in the 64-bit partitioned adders of the datapath in the second pipeline stage. Table-1 shows the operation of these instructions on input operands A and B, the result is produced in C and D. The A, B, C, and D can be a byte, half-word, word, or double word packed as 8-bytes, 4-half-words, 2-words, and a double-word respectively; in a 64-bit register Rx, where x = 0…n. The partition of the 64-bit adder is performed according to the partition control signals as shown in Table-2. Since we require two 64-bit adders for the multipliers, therefore by re-using these adders we can support two 64-bit operations in parallel.

**Table-1.** Instructions supported by the ALU (A, B, are the inputs, and C, D are the outputs).

| Instruction | Operation. | Result |
|---|---|---|
| ADD | A + B | C |
| ADS(U) | A + B | C or SAT(max) |
| ADS(S) | A + B | C or SAT(min) or SAT(max) |
| SUB | A-B | C |
| SUS(U) | A-B | C |
| SUS(S) | A-B | C or SAT(min) or SAT(max) |
| BFS(U) | A+B, A-B | C or SAT(max) , D |
| BFS(S) | A+B, A-B | C, D or SAT(min) or SAT(max) |
| BFD(U) | A+B/2, A-B/2 | C/2 or SAT(max), D/2 |
| BFD(S) | A+B/2, A-B/2 | C/2, D/2 or SAT(min) or SAT(max) |
| AVG2(U) | A+B/2 | C/2 |
| AVG2(S) | A+B/2 | C/2 |

The Add and Subtract operations are performed as normal Addition and Subtraction. The parallel average of two numbers is a very common and useful function in image and video processing: the arithmetic mean as shown in Fig. 5-a). Only few high performance ISAs[10, 17] support this operation. This combined operation involves an addition and a right shift of one bit (divide by two). In the proposed datapath, AVG2 operation is implemented using the 64-bit partitioned adder; performing

normal addition and finally shifting the result one bit right. The datapath shifts in the carry out bit as the most significant bit of the result, so it has the added advantage that no overflow can occur.

**Table-2.** Partition of the ALU, using Partition control signals.

| Part1Part0 | ALU partition |
|---|---|
| 00 | Byte |
| 01 | Half_word |
| 10 | Word |
| 11 | Double_word |

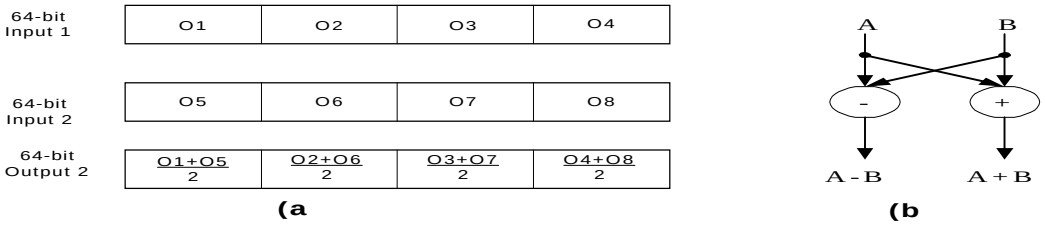


**Fig. 5.** a) Parallel averaging operation, b) Butterfly Operation on A and B.

### 3.1. Butterfly

Butterfly is an important operation, which is frequently required in FFT, and IDCT. Butterfly operation is performed as shown in Fig. 5-b). In the proposed datapath the Butterfly (BFS) and Butterfly Divide by 2 (BFD) instructions are implemented using the two 64-bit partitioned adders. In this operation one 64-bit adder performs the addition while the other one performs the subtraction in parallel, at the end two results are combined to produce the final result.

The BFD instruction performs the addition and subtraction of two operands simultaneously and then divides the result by two (shift right). Since, we are performing AVG2 operation, therefore the BFD operation is performed using the same circuit. There are two types of BFD instructions one is for signed operands; the other is for unsigned operands. The basic operation for both the instructions is the same, but in case of signed operands, the sign of the result is extended during the shift right operation.

### 3.2. Saturated Arithmetic

In multimedia arithmetic we deal mostly with 8 or 16-bit pixel data, the data represent the color intensity and luminous information. To represent this data we cannot use modulo arithmetic in which the overflow is ignored, because a small change in color intensity or luminous may result a huge a change in the resulting information, for e.g., from FFHex to 00Hex (white to black). Therefore, in media signal processing we use saturated arithmetic. In case of saturated arithmetic it is necessary to clamp overflowing results to a high or low value.

There are two modes of saturation asymmetric and symmetric. For unsigned integers, the two modes are the same. While, for signed values, symmetric mode saturates positive and negative numbers to the same absolute value function, for instance signed byte saturation values of -127 and +127. The asymmetric mode will saturate the negative value to a number, which is an absolute value of one greater than the positive value, for instance -128 and +127 for sign bytes. The proposed datapath can support only asymmetric mode of saturation. The conditions for saturation resolution based on overflow/underflow are summarized in Table-3 and Table-4, for Add and Subtract operations respectively. In these tables, A and B are the input operands and As, and Bs, are the sign bits of these operands. In case of unsigned addition an overflow occurs when carry out is '1' and the result is saturated to the maximum value. While, in case of unsigned subtraction an underflow occurs when carry out ($C_{out}$) is '0' and the result is saturated to the minimum value. Similarly, in the case of signed addition an overflow occurs when both the input operands are positive and carry out and carry out-1 ($C_{out-1}$) are '1'. In this case, the result is saturated to the maximum positive value. These rules are summarized in Table-3 and Table-4.

**Table-3.** Overflow and underflow detection for Add operation.

| Data Type | MSB | Operation | Overflow | Underflow |
|---|---|---|---|---|
| Unsigned | | A + B | $C_{out} = 1$ | |
| Signed | As = 0, Bs = 0 | A + B | $C_{out}$ XOR $C_{out-1}$ | |
| | As = 1, Bs = 1 | A + B | | $C_{out}$ XOR $C_{out-1}$ |
| | As = 1, Bs = 0 | A + B | | |
| | As = 0, Bs = 1 | A + B | | |

**Table-4.** Overflow and underflow detection for Subtract operation.

| Data Type | | Operation | Overflow | Underflow |
|---|---|---|---|---|
| Unsign | | A - B | | $C_{out} = 0$ (A<B) |
| Sign | As = 0, Bs = 0 | A - B | | |
| | As = 1, Bs = 1 | A - B | | |
| | As = 1, Bs = 0 | A - B | | $C_{out}$ XOR $C_{out-1}$ |
| | As = 0, Bs = 1 | A - B | $C_{out}$ XOR $C_{out-1}$ | |

## 4. MULTIPLY INSTRUCTIONS

Multiplication is one of the most important operations in digital signal processing. Media processing requires different kinds of multiply, and multiply with accumulate operations on signed and unsigned operands, with or without rounding of the result.

Table-5 shows the multiply and multiply accumulate instructions required for the efficient processing of media signals. These instructions are executed using 32x16 multiplier blocks of the datapath. The input operands are A and B, the result is produced in C and D. A and B can be a byte, half-word, 24-bit or word packed as 8-bytes, 4-half-words, 2-24-bit or 2-words in a 64-bit register Rx, where x = 0…n. The partition of the multipliers is performed according to the partition control signals as shown in Table-2 for the ALU. The only difference is that '11' is used for 24-bit operation instead of 64-bit operation. The multiplier hardware is configured for signed and unsigned operation using the sign control bit. When 'Sign' bit is '1' signed multiplication is performed and when it is '0' unsigned multiplication is performed. All the multiplication instructions require two-cycle latency with a single-cycle throughput.

**Table-5.** Multiply operations supported by the proposed datapath.

| Instruction | Mnemonic | Operation |
|---|---|---|
| Multiplication with full resolution (S / U) | MUL (S/U) | C = A*B |
| Multiplication with accumulate (S / U) | MULA (S/U) | C = A*B + C' |
| Multiplication with deduct (S / U) | MULD (S/U) | C = A*B - C' |
| Multiply upper half result with rounding (S / U) | MULH (S/U) | C = Ah*Bh |
| Sum of two products (S / U) | SUMP (S/U) | $A_1*B_1 + A_0*B_0$ |
| Sum of two products with accumulate (S / U) | SUMPA (S/U) | $A_1*B_1 + A_0*B_0 + C'$ |
| Sum of two products with deduct (S / U) | SUMPD (S/U) | $A_1*B_1 + A_0*B_0 - C'$ |

### 4.1. Multiply

The MUL operations are performed as normal multiplication. In simple multiply operation, the two input operands are multiplied in the first cycle producing Sum, and Carry vectors. In the second cycle sum and carry vectors of AxB are added using the 64-bit partitioned adder to produce the final result as shown in Fig. 6. Since the result of multiplication is twice the width of the input operands, the result of simple multiplication is produced in C and D (C contains the lower half of the result and D contains the upper half of the multiplication). The only difference in MUL (U) and MUL(S), is that MUL(U) is the unsigned multiplication, while MUL(S) is the signed multiplication. Fig. 6-a) shows the datapath for 8x8 and 16x16 multiplication. While Fig. 6-b) shows the datapath for 24x24 and 32x32 multiplication which requires two 16x32 multipliers.

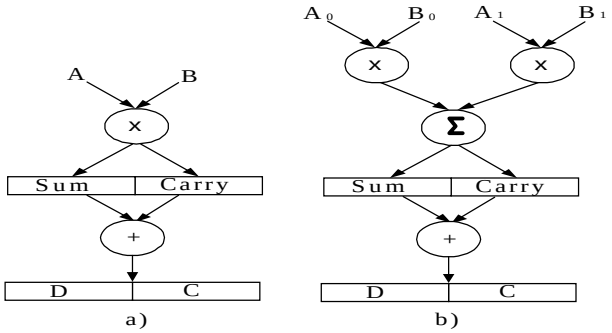


Fig. 6. Two cycle multiply operation a) 8x8 or 16x16, and b) 24x24 or 32x32.

### 4.2. Multiply Accumulate Operation

In the case of the multiply accumulate operation (MULA), the two input operands are multiplied in the first cycle producing Sum and carry vectors, then a third operand F|E (F concatenated to E) is added with the Sum and carry vectors of AxB in the second cycle using the 4x2 compressor. The output of the 4x2 compressor is finally added using the 64-bit partitioned adder to produce the final result as shown in Fig. 7.

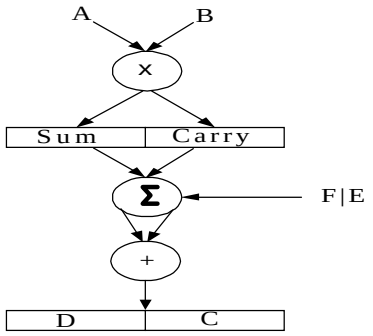


**Fig. 7.** Multiply accumulate operation.

### 4.3. SUMP

Sum of two products is an important operation supported by most of the high performance media processors. This operation performs the addition of two products and it is equivalent to two multiply and one addition as shown below:

$$C = A_0xB_0 + A_1xB_1 \qquad \text{Equation-15}$$

The proposed datapath performs the addition of two products (SUMP instruction) in two cycles with single cycle throughput. The instruction execution at macro level is shown in Fig. 8. This instruction requires two multiplications $A_1xB_1$, and $A_0xB_0$. These multiplications are performed in the first cycle using the MUL 32x16 blocks. In case of byte and half-word operation the summation of two products is performed in the first cycle (Fig. 8-a) using the 4x2 compressor after the multipliers. While in case of word operation, first cycle 4x2 compressor is used for the addition of the partial products of the 32x32 multiplication. Therefore, the summation of two 32x32 products is performed in the second cycle (Fig. 8-b) using the second 4x2 compressor. The sum and carry vectors produced by the summations are finally added using the 64-bit partitioned adder to produce the addition ($A_1xB_1$+ $A_0xB_0$) as shown in Fig. 8. In order to produce the result of the same bit width as the input the result of the addition is rounded and only the upper half of the result is stored in the output.

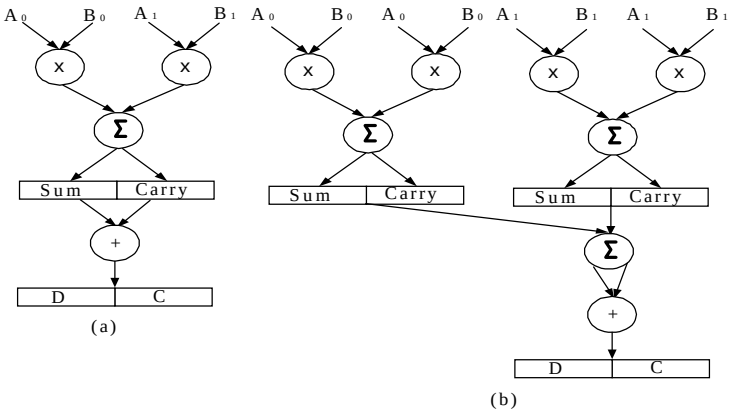


**Fig. 8.** Sum of two products: (a) Byte or half-word, and (b) 32x32.

## 5. CONCLUSION

In this paper, we present the quantitative analysis and the computational complexity required to perform media processing. It is clear from these estimates, that the software-only approach to MPEG decoder requires a processing power of 424 MIPS on processors without media enhancement, while SIMD media instructions reduce the MPEG execution time by approximately three times (143.3 MIPS). The major time consuming operation in MPEG decoding is motion compensation (49.8 MIPS) followed by the display step (41.47 MIPS), and then IDCT (40 MIPS). Fortunately, the two inherently serial steps, decoding the MPEG headers and the Huffman decoding (12 MIPS), are relatively insignificant in execution time. The actual MPEG decoder implementation requires a higher processing power due to factors such as cache misses, memory access time, and bus width. However, the number of operations for motion compensation processing is reduced if no pixel interpolation (a non-half-pixel bitstream) is required. Based on these estimates main classes of instructions that are needed for the required level of performance of the Media Processor are identified. Their efficient implementation and effect on the processor data-path is discussed. The main operations required in media processing are Addition (with or without saturation), Multiplication (with or without rounding), Sum of Products, and Average of two numbers.

## REFERENCES


1. "i860 TM microprocessor family programmers reference manual," Intel corporate literature, 1991.
2. K. Deifendorff and M. Allen, "Organization of the Motorola 88110 superscalar RISC microprocessor", IEEE Micro Mag., vol. 12, pp. 40–63, Apr. 1992.
3. Kuroda, I.; Nishitani, T., " Multimedia processors", Proceedings of the IEEE Volume: 86 6, June 1998, Page(s): 1203 –1221.
4. D. Brinthaupt, D. L. Letham, V. Maheshwari, J. H. Othmer, R. R. Spivak, B. Edwards, C. Terman, and N. Weste, "A video decoder for H.261 video teleconferencing and MPEG stored interactive video applications", in Dig. Tech. Papers ISSCC'93, Feb. 1993, pp. 34-35.
5. D. Galbi, E. Bird, S. Bose, E. Chai, Y.-N. Chang, P. Dermy, N. Fernando, J.-G. Fritsch, E. Hamilton, B. Hu, E. Hua, F. Liao, M. Lin, M. Ma, E. Paluch, S. Purcell, H. Yanagi, S. Yang, M. Chow, T. Fujii, A. Fujiwara, H. Goto, K. Ibara, S. Isozaki, J. Jao, I. Kaneda, M. Koyama, T. Mineo, I. Miyashita, G. Ono, S. Otake, A. Sato, A. Sugiyama, K. Tagami, K. Tsuge, T. Udagawa, K. Yamasaki, S. Yasura, and T. Yoshimura, "An MPEG-1 audio/video decoder with run-length compressed antialiased video overlays," in Dig. Tech. Papers ISSCC'95, Feb. 1995, pp. 286-287.
6. M. Harrand, M. Henry, P. Chaisemartin, P. Mougeat, Y. Du-rand, A. Tournier, R. Wilson, J.-C. Heriuison, J.-C. Longcham-bon, J.-L. Bauer, M. Runts, and J. Bulone, "A single chip video-phone video encoder/decoder", in Dig. Tech. Papers ISSCC'95, Feb. 1994, pp. 292–293.
7. T. Arai, K. Suzuki, and I. Kuroda, "V830R/AV: Embedded multimedia superscalar RISC processor with rambus interface", in Proc. HOTCHIPS IX, Aug. 1997, pp. 177–189.
8. Makino, H.; Suzuki, H.; Morinaka, H.; Nakase, Y.; Mashiko, K.,"A 286 MHz 64-bit floating point multiplier with enhanced CG operation", Symposium on VLSI Circuits, Digest of Technical Papers., 1995 , Page(s): 15 -16.
9. NEC Corporation, " PD77016 user's manual", 1992.
10. Lee. R., "Accelerating multimedia with enhanced microprocessor", IEEE Micro vol.15, No. 2, pp. 22, April 1995.

11. M. Tremblay, J. M. O'Connor, V. Narayanan, and L. He, "VIS speeds new media processing", IEEE Micro Mag., vol. 16, pp. 10-20, Aug. 1996.
12. M. Tremblay, D. Greenley, and K. Normoyle, "The design of the microarchitecture of UltraSparc TM-I", Proc. IEEE, vol. 83, pp. 1653-1663, Dec. 1995.
13. S. Rathnam and G. Slavenburg, "An architectural overview of the programmable multimedia processor, TM-1", in Proceedings of Compcon. New York: IEEE Computer Science Press, 1996, pp. 319-326.
14. R. B. Lee, "Subword parallelism with MAX-2", IEEE Micro Mag., vol. 16, pp. 51-59, Aug. 1996.
15. A. Peleg and U. Weiser, "MMX technology extension to the Intel architecture", IEEE Micro Mag., vol. 16, pp. 42-50, Aug. 1996.
16. L. Gwennap, "Digital, MIPS add multimedia extensions", Microprocessor Rep., vol. 10, no. 15, pp. 24-28, Nov. 1996.
17. Schmookler, M.S., et. al., "A low-power, high-speed implementation of a PowerPC microprocessor vector extension", Proceedings 14th IEEE Symp. on Comp. Arith., 1999 , Page(s): 12 –19.
18. M. Berekovic, H. Stolberg, M. B. Kulaczewski, and P. Pisch, "Instruction set extensions for MPEG-4 video", Journal of VLSI signal processing, vol. 23, pp. 27-49, October 1999.
19. Kuhn, P., Stechele, W., "Complexity Analysis of the Emerging MPEG-4 Standard as a Basis for VLSI Implementation", vol. SPIE 3309 Visual Communications and Image Processing, San Jose, Jan. 1998, pp. 498-509.
20. K. Patel, B. C. Smith, and L. A. Rowe, "Performance of a software MPEG video decoder", in Proc. ACM Multimedia Conf. 1993.
21. Chang-Guo Zhou; Ihtisham Kabir; Leslie Kohn; Aman Jabbi; Rice, D.; Xio-Ping Hu, " MPEG video decoding with the UltraSPARC visual instruction set", Compcon '95.'Technologies for the Information Superhighway', Digest of Papers. , 1995 , Page(s): 470 -477.
22. M. Ikekawa, D. Ishii, E. Murata, K. Numata, Y. Takamizawa, and M. Tanaka, "A real-time software MPEG-2 decoder for multimedia PC's", in ICCE Dig. Tech. Papers, June 1997, pp. 2–3.
23. D. Ishii, M. Ikekawa, and I. Kuroda, "Parallel variable length decoding with inverse quantization for software MPEG-2 de-coders", in Proc. IEEE Workshop Signal Processing Systems (SiPS97), Nov. 1997, pp. 500–509.
24. K. Nadehara, H. J. Stolberg, M. Ikekawa, E. Murata, and I. Kuroda, "Real-time software MPEG-1 video decoder design for low-cost, low-power applications," in Proc. IEEE VLSI Signal Processing IX, Oct. 1996, pp. 438–447.
25. Gregory K. Wallace, "The JPEG still picture compression standard", IEEE Transaction on Consumer Electronics, Vol. 38, No. 1, February 1992.
26. Arup K. Bhattacharya, and Syed S. Haider, "A VLSI architecture of an Inverse Discrete Cosine Transform", 7th International conference on VLSI Design, January 1994.
27. B. G. Lee, "A new algorithm to compute the discrete cosine transform", IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP-32, pp. 1243–1245, Dec. 1984.
28. Y. Arai, T. Agui, and M. Nakajima, "A fast DCT-SQ scheme for images", Trans. IEICE, vol. E71, no. 11, p. 1095, Nov. 1988.
29. W. A. Chen, C. H. Smith, and S. C. Fralick, "A fast computational algorithm for the discrete cosine transform", IEEE Trans. Commun., vol. COM-25, no. 9, pp. 1004–1011, 1977.
30. C. Loef er, A. Ligtenberg, and G. S. Moschytz, "Practical fast 1-D DCT algorithms with 11 multiplications", in Proc. Int. Conf. Acoustics, Speech, and Signal Processing 1989 (ICASSP'89), pp. 988–991.
31. A. C. Hung and T. H.-Y. Meng, "Statistical inverse discrete cosine transforms for image compression", in Proc. SPIE Digi-tal Video Compression on Personal Computers: Algorithms and Technologies, A. A. Rodriguez, Ed., vol. 2187, 1994.
32. M. Yoshida, H. Ohtomo, and I. Kuroda, "A new generation 16-bit general purpose programmable DSP and its video rate application", in Proc. IEEE VLSI Signal Processing VI, Oct. 1993, pp. 93–101.
33. IEEE Standard Specification for the Implementation of 8 by 8 Inverse Discrete Cosine Transform, IEEE Standard 1180–1990, 1990.
34. "Inverse discrete cosine transform", International Standards Organization, ISO/IEC 13818-2:1996/Cor.2 1996(E) in Annex A, 1996.
35 "Module Compiler™ Reference Manual Version 2000.05", Synopsys Corporation, Mountain View, USA, May 2000.