MULTIMEDIA EXTENSIONS FOR DLX PROCESSOR

Elham Khorsandi Nia & Omid Fatemi
Department of Electrical and Computer Engineering - University of Tehran
Omid@fatemi.net

ABSTRACT

In recent years, the success of Internet and World Wide Web, and the growing feasibility of image and video compression techniques have pushed multimedia into mainstream computing. These requirements necessitate new and modified hardware architectures enabling real-time multimedia applications. Three methods have been proposed for enhancing multimedia architectures namely dedicated processors, media processors and multimedia extensions for general-purpose processors. Multimedia extended instruction set is an efficient solution for public and widely used computers, because it offers a solution with less expense and high flexibility. In this paper, we propose an enhanced multimedia extended instruction set for the DLX RISC processor. The enhancement is shown by implementing typical multimedia applications. Our synthesis and simulation results show an average speedup of 3.3 for these applications at a expense of 3% growth in chip area.

1. INTRODUCTION

Media processing, or the processing of digital multimedia data requires significant computation power.

Multimedia now defines a significant portion of the computing market, and this is expected to grow considerably. As a result, the processing demands for multimedia applications are rapidly escalating as users desire new and better applications.

Therefore, new architectures for multimedia applications have been proposed. The main methods for supporting multimedia include [1]:

1. Dedicated (application-specific) hardware: Dedicated processors offer an optimized hardware solution for multimedia processing. The major drawback of using dedicated processors is that they provide limited if any flexibility because they are optimized to implement a specific function. An example of a dedicated processor is Analog Devices Inc.’s ADV-JP2000[2]. The ADV-JP2000 is a high performance image co-processor that implements the computationally intensive operations of the JPEG2000 image compression standard in hardware.

2. Media processors: In recent years, several IC vendors have presented processors, generally based on VLIW architectures, which can handle media processing cores for applications ranging from PC multimedia to high-definition digital TV. Examples of media processors are M pact1 and 2 from Chromatic research semiconductor (2-way VLIW), MAP1000A from Equator technologies (4-way VLIW), D30V from Mitsubishi Electronics (2-way VLIW) and Trimedia TM1100 from Philips semiconductor (5-way VLIW) [2][3][4][5][6][7].

3. Multimedia extensions for general-purpose processors: This method has gained more attention in recent years. This has become a very popular method in workstations and personal computer systems as it provides required performance for multimedia applications without a significant extra cost. These instruction set architecture (ISA) extensions operate in a SIMD (Single Input Multiple Data) fashion to exploit data level parallelism (DLP) in multimedia applications.

In this paper, we propose an enhanced multimedia extended instruction set for the DLX RISC processor.

Section 2 describes multimedia extensions for general-purpose processors. In Section 3, we propose the multimedia extensions for DLX processor and section 4 shows the simulation results. Section 5 concludes this paper followed by the references.

2. MULTIMEDIA EXTENSIONS

Multimedia and digital signal processing (DSP) applications typically use small data typed (primarily 8- and 16-bits) and spend a significant portion of the execution time in loops that have a high degree of processing regularity. Packing several small data elements into the wider GPP datapath (typically 32- or 64-bits wide) enables simultaneous processing of separate data elements. This form of SIMD parallelism is commonly known as subword parallelism [1][2].

Subword parallelism was first introduced by Hewlett-Packard in 1994 with the introduction of MAX-1 in PA-RISC 1.0 instruction set [9]. Initial implementation of the SIMD extensions such as Intel’s MMX, Sun’s VIS, Compaq’s MVI, MIPS’s MDMX, and HP’s MAX support integer data types [9][10]. Floating-point support in media extensions was introduced first in 3DNOW! by AMD and was followed by SSE and SSE2 by Intel [10][11]. Motorola’s AltiVec has been introduced with both integer and floating-point capability simultaneously [13].

2.1. Classification of Multimedia Instructions

We propose to categorize multimedia instructions in six groups. These six groups are:

1) Arithmetic (integer, floating-point) instructions:
These instructions include basic functions of arithmetic such as add, sub, multiplication, division, maximum, minimum, reciprocal, square root, average for integer and floating-point operands.

2) Logical instructions:
These instructions include logical operations such as AND, OR, NOT, XOR, … which are bitwise operations.

3) Compare instructions:
These instructions perform the compare operation on operands with integer and floating-point data types.

4) Conversion instructions:
Conversion instructions are used for converting data types. For example, it is used for converting integer data types (signed/unsigned) to floating-point data types with single or double precision.

5) Permutation instructions:
These groups of instructions are employed for rearranging different subwords in one register or between two registers or for packing and unpacking subwords in registers.

6) Others:
There are instructions which are frequently used in multimedia programs like clearing a cache line, loading state/control/condition registers, pre-fetch and pause operations.

3. MULTIMEDIA EXTENSIONS FOR DLX ARCHITECTURE
DLX is a simple RISC-type architecture. It features a minimal instruction set, few addressing modes, and a simple processor architecture [15].

DLX is a 32-bit word-oriented system. The CPU contains a 32-bit ALU, 32 general registers organized in a register file, three buses, six special purpose registers and three registers which provide more orthogonal access to registers in the register file. DLX instruction come in three formats:

\[
\begin{array}{c}
\text{R-type op rs1 rs2 rd fnc} \\
\text{I-type op rs1 rd imm} \\
\text{J-type op Offset}
\end{array}
\]

Where op is six bits, fnc is 11 bits, imm is 16 bits, offset is 26 bits, and all r* fields are five bits.

DLX architecture is a general-purpose processor. Handling multimedia applications with high performance necessitates multimedia extensions method. In this section, we propose our multimedia instructions for DLX processor.

3.1. DLX Multimedia Instruction Set
We note that DLX processor can not support packed floating-point instructions, because it has 32 bits registers and instructions.

Table 1 explains DLX packed integer arithmetic instructions. The subword size is indicated by the suffix “b” for byte(8 bits) and “h” for half-word(16 bits).

Packed add and packed subtract instructions accept signed and unsigned operand. Figure 1 shows an example of packed add.

Figure 1. Packed add instruction on 8-bit subwords

Packed compare instructions perform comparison operation between subwords of two registers. Figure 2 shows the comparing operands in packed 16 bit operand. Packed shift instructions shift operand left or right. The shift amount can be specified either as a constant in an immediate field or as a variable in a register.

Figure 2. Packed compare instruction (16-bit subword)

Table 2 shows the subword permutation instructions.

- Packing instructions (packhh and packlh) are used to create smaller data types from larger ones (Figure 3).

- Unpacking instructions are employed to create larger data types from smaller ones. It is implemented for byte or half-word subword with sign or zero extensions (unpcklb, unpackslb, unpckhh, unpackshb, unpackshh, unpackhh, unpacklh). Figure 4 shows an example of these instructions.

- Mix instructions (mixl, mixh) take subwords from two registers, and interleaves alternate subword from each register in the result register as shown in Figure 5. The suffix “l” or “r” indicates Mix left or Mix right: Mix left collects the odd subwords in the result register, whereas Mix right collects the even numbered subwords.

- The permute instructions (permuteb, permuteh) take one source register, and produces a permutation of the subwords in that register. With 8-bit subwords, this instruction allows all possible permutations, with and
without repetitions of the four subwords in the source register (Figure 6).

Figure 4. Unpack high with zero extension on 16-bit subword

Figure 5. (a) mixl instruction  (b) mixr instruction

Figure 6. Permute operation on byte subwords

3.2- Effects on the Architecture
For designing DLX with above multimedia instruction set, the datapath and controller of architecture has been modified. The datapath of DLX included one ALU unit for arithmetic, shift and compare operations. To enable processing for subword parallelism, ALU is partitioned. This partitioning involves manipulating the carry chains to prevent overflow of the processing of one subword segment into the next. This partitioned ALU of DLX operates on byte, half-word and word operands and executes the instructions in Table 1.

Additional hardware is required to implement methods of rearranging and packing/unpacking the packed register. In modified DLX, we have added a unit for this operation which is called perm unit. This unit performs the permutation instructions which are shown in Table 2.

Subword parallelism does not require any additional ports in the register file, unlike most other forms of parallel processing.

Overall, the typical area overhead for multimedia extensions in DLX is only about 3 % of the entire chip area. We achieved this area overhead by using Leonardo Tools for synthesizing DLX processor in Actel (1200XL) technology.

4. SIMULATION RESULTS
Modelsim has been used for VHDL simulation. Two example algorithms namely block matching and matrix transpose which are used in multimedia applications are then mapped to new DLX processor. Our optimized code has an effect of reducing the number of instructions required which is due to parallelism techniques in DLX architecture[9]. VHDL simulation and assembly code of these two examples have been implemented. We explain briefly these two example.

4.1. Block Matching
In this example, the inner loop accumulates the absolute magnitude of the difference of two corresponding values from two 16*16 blocks of data. This is often used in motion estimation. Its algorithm is as follow:
For \( i=0; i<15; i++ \)
For \( j=0; j<15; j++ \)
\[ \text{Sum} += \text{abs}([a][i][j] – [b][i-m][j-n]) \]

It is clear that we can compute above sum in parallel. Therefore, we use the parallel add and sub in byte instructions (PadduB, PsubuB). At the end of loop, we should use unpacking instructions (UnpacklB, UnpackhB) for computing final results.

4.2. Matrix Transpose (4*4)
We assume 16-bit values of a 4*4 matrix are in memory. For implementing matrix transpose with multimedia instructions, we can use PacklH and PackhH instructions.

Simulation results show a speedup factor of 3.28 for block matching and 3.42 for matrix transpose algorithms for the code with multimedia instructions over the original code.

5. CONCLUSIONS
Multimedia applications are becoming increasingly important and require enhanced hardware architectures. These techniques include: 1) dedicated processors which employ an application specific hardware solution for achieving high performance, 2) media processors which mostly employ VLIW architectures to provide high performance and high flexibility, and 3) multimedia extensions for general-purpose processors which are widely used for today’s multimedia PC and workstations. The last method is a cost-effective solution for PCs and general-purpose processors. In this paper, to enhance DLX RISC processor (the third solution) we first categorized multimedia operations in 6 groups. This helped us to select our add-on instructions. We propose to add 48 instructions to DLX architecture. These instructions are grouped in two classes: 1) arithmetic instructions namely add, subtract, shift and compare (36 instructions), and 2) permutation instructions namely packing/unpacking, mix, and permute (12 instructions).
The achieved speed up is in average 3.3 for two example multimedia algorithm at the expense of adding 3% area to real state of the chip.

6. REFERENCES

[12] SSE, SSE2, MMX instruction set are available at : http://www.cpuid.com

Table 1. Packed integer arithmetic instructions for DLX

<table>
<thead>
<tr>
<th>Instruct</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Paddux</td>
<td>Add 4 or 2 pairs of 8-or 16-bit operands based on x(b, h)(unsigned)</td>
</tr>
<tr>
<td>3 Paddx</td>
<td>Add 4 or 2 pairs of 8- or 16-bit operands based on x(b, h)(signed)</td>
</tr>
<tr>
<td>5 Psabux</td>
<td>Subtract 4 or 2 pairs of 8- or 16-bit operands based on x(b, h)(unsigned)</td>
</tr>
<tr>
<td>7 Psbux</td>
<td>Subtract 4 or 2 pairs of 8- or 16-bit operands based on x(b, h)(signed)</td>
</tr>
<tr>
<td>9 Psllx</td>
<td>Shift left by 0 to 8 or 16 bits(variable) based on x(b, h), with zero shifted in on the left end</td>
</tr>
<tr>
<td>11 Pslr</td>
<td>Shift right by 0 to 8 or 16 bits(variable) based on x(b, h), with zero extensions</td>
</tr>
<tr>
<td>13 Pssr</td>
<td>Shift right by 0 to 8 or 16 bits(variable) based on x(b, h), with sign extensions</td>
</tr>
<tr>
<td>14 Pseqx</td>
<td>Compare (equality) 4 or 2 pairs of 8- or 16-bit operands based on x(b, h)</td>
</tr>
<tr>
<td>15 Pslex</td>
<td>Compare (not equal) 4 or 2 pairs of 8- or 16-bit operands based on x(b, h)</td>
</tr>
<tr>
<td>16 Pseqx</td>
<td>Compare (greater than or equal) 4 or 2 pairs of 8- or 16-bit operands based on x(b, h)</td>
</tr>
<tr>
<td>17 Psllx</td>
<td>Shift left by 0 to 8 or 16 bits(immediate) based on x(b, h), with zero shifted in on the left end</td>
</tr>
<tr>
<td>18 Pssr</td>
<td>Shift right by 0 to 8 or 16 bits(immediate) based on x(b, h), with zero extensions</td>
</tr>
<tr>
<td>19 Pssr</td>
<td>Shift right by 0 to 8 or 16 bits(immediate) based on x(b, h), with sign extensions</td>
</tr>
<tr>
<td>22 Pseqx</td>
<td>Compare (equality) packed byte or half-word operand based on x(b, h) with immediate</td>
</tr>
<tr>
<td>23 Pslex</td>
<td>Compare (less than or equal) packed byte or half-word operand based on x(b, h) with</td>
</tr>
<tr>
<td>24 Pseqx</td>
<td>Compare (greater than or equal) packed byte or half-word operand based on x(b, h)</td>
</tr>
</tbody>
</table>

Table 2. Permutation instructions for DLX

<table>
<thead>
<tr>
<th>Instruct</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Permuth</td>
<td>Rearrange byte subwords from one register, with or without repetition</td>
</tr>
<tr>
<td>2 Permuth</td>
<td>Rearrange half-word subwords from one register, with or without repetition</td>
</tr>
<tr>
<td>3 Mixl</td>
<td>Interleave alternate 8-bit subwords from two source registers starting from leftmost</td>
</tr>
<tr>
<td>4 Mixr</td>
<td>Interleave alternate 8-bit subwords from two source registers starting from rightmost</td>
</tr>
<tr>
<td>5 Packhh</td>
<td>Pack 16-bit high subwords of operands in target register</td>
</tr>
<tr>
<td>6 Packlh</td>
<td>Pack 16-bit low subwords of operands in target register</td>
</tr>
<tr>
<td>7 Unpackh</td>
<td>Unpack high 16-bit subword of operand with zero extension.</td>
</tr>
<tr>
<td>8 Unpacksh</td>
<td>Unpack high 16-bit subword of operand with sign extension.</td>
</tr>
<tr>
<td>9 Unpacklb</td>
<td>Unpack lowest 8-bit subword of operand with zero extension.</td>
</tr>
<tr>
<td>10 Unpacksib</td>
<td>Unpack lowest 8-bit subword of operand with sign extension.</td>
</tr>
<tr>
<td>11 Unpackhh</td>
<td>Unpack highest 8-bit subword of operand with zero extension.</td>
</tr>
<tr>
<td>12 Unpackshb</td>
<td>Unpack highest 8-bit subword of operand with sign extension.</td>
</tr>
</tbody>
</table>