Adding Support for Vector Instructions to 8051 Architecture

Akhil Alluri[1], Dhiraj Balakrishnan[1], Manvendra Singh[1], Rohan Verma[1], Pulkit Gairola[1] and Dr. Rajeev Kumar Singh[2]

***Abstract*— The focus of this paper is on adding support for vector instructions to the Intel 8051 architecture. The proposed architecture has a new vector register bank and a unit to decode the vector addresses.**

I. INTRODUCTION

A vector processor is a processor design wherein the instruction set includes operations that can perform mathematical operations on multiple data elements in sequential cycles. A vector processor that has the capability to perform operations on all the elements of a vector at once is termed Array Processor. [1]

Vector processors are in laymen speak could be called mini supercomputers, machines built primarily to handle large scientific and engineering calculations. They derive their performance from a heavily pipe lined architecture which operations on vectors and matrices can efficiently exploit.

We are trying to make changes to an existing scalar processor so that it use a single instruction to operate on one-dimensional arrays of data called vectors i.e. we are trying to implement a Vector Processor on a Scalar Processor. The scalar processor we chose for this purpose is the Intel 8051.

A. Why 8051 ?

The 8051 was a popular micro controller and still is due to its simplicity. It has since been replaced by more powerful and efficient architecture like the AVR and ARM.

The data path of the 8051 is simple enough to be pliable for adding an experimental Vectorization module. The same approach may be possible on any other current analogous architecture. [2]

Some of the features that have made the 8051 popular are:

- 4 KB on chip program memory.
- 128 bytes on chip data memory(RAM)

B. Components of 8051

- 4 reg banks.
- 128 user defined software flags.
- 8-bit data bus
- 16-bit address bus
- 16 bit timers (usually 2, but may have more, or less).
- 3 internal and 2 external interrupts.
- Bit as well as byte addressable RAM area of 16 bytes.
- Four 8-bit ports, (short models have two 8-bit ports).
- 16-bit program counter and data pointer.
- 1 Microsecond instruction cycle with 12 MHz Crystal.

[1]The students are with the Department of Computer Science, Shiv Nadar University. Akhil Alluri aa417@snu.edu.in. Dhiraj Balakrishnan db422@snu.edu.in. Manvendra Singh ms234@snu.edu.in. Rohan Verma rv285@snu.edu.in. Pulkit Gairola pg473@snu.edu.in

[2]Dr Rajeev Kumar Singh is with the Faculty of the Department of Computer Science, Shiv Nadar University, NH91, Tehsil Dadri Gautam Buddha Nagar, Greater Noida, Uttar Pradesh 201314, India rajeev.kumar@snu.edu.in

II. IMPLEMENTATION

A. Components added to enable Vectorization

1) Register Banks - 3 Vector Register Banks (V1,V2,V3). Each bank has eight register cells.
2) Instruction Set - Proposed addition of Instructions to the family of 8051 Instruction set namely: vlw, vst, vadd, 'vsub', 'vmul'.

TABLE I
PROPOSED ADDITIONS TO THE INSTRUCTION SET

Op code	Operand	Operation
VADD	V1, V2	V3 = V1 + V2
VSUB	V1, V2	V3 = V1 - V2
VMUL	V1, V2	V3 = V1 $\times$ V2
VLW	V1, x(R1)	Load into Vector Bank 1 starting from memory address stored in R1
VSW	V1, x(R1)	Store from Vector Bank 1 starting into memory address stored in R1

3) Control Unit for Vector access - A control unit that enables sequential access to the vector bank for loads, stores and processing. The logic is similar to the data path for the *Program Counter*. This is key to the concept of vectorization.

B. Data flow in using Vectorization

The normal data flow of the 8051 is not affected during normal operation. After the Instruction is decoded to be a 'Vector' operation, the main control circuit of the 8051 activates the secondary control system data path as shown in the figure 2.

There is a counter unit analogous to the program counter data path to enable sequential register element indexing in the vector being used. The counter gives the index input to the PLA.

The PLA has control lines (in blue) from the main control engine. These lines will enable the PLA to select which vector bank to access. The PLA also controls the index counter for reset purposes. The PLA's output is given to a standard register address decoder. The aforementioned

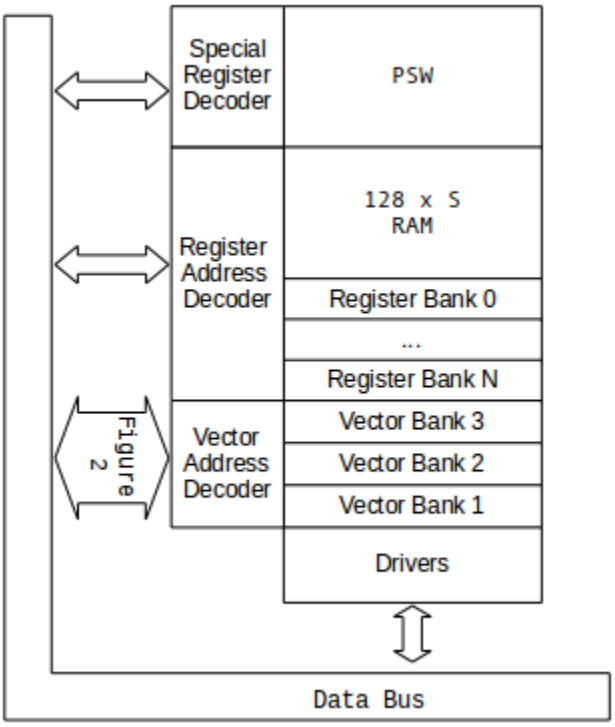

Fig. 1. Layout of the RAM with Vector Register Bank

control lines also allow the PLA to query the memory for the next element.

Data path flow for a program: Consider the first instruction in the *Vectorized assembly code* in Sec III.

1) The instruction decoder encounters a vlw op code.
2) The control engine passes the information to the vector PLA. Sets Index counter to 0.
3) The control engine also begins querying the data from the memory referenced by the instruction operand.
4) The incoming word is stored into the vector referenced by the PLA control lines and the register address decoder.
5) The index counter is automatically incremented.
6) This process is repeated to complete the vector load.

*In case of the **VADD** instruction:*

1) The PLA resets the counter to 0. Register addressing decoder passes the corresponding cell to the ALU.
2) The result from the ALU is stored in the result vector of the **same index**.
3) This is repeated until the end of the vector.

III. ANALYSIS

A. Metrics

Vectorization reduces the number of instructions to process a continuous block of data. An example to support this statement is provided below.
Consider the following case:

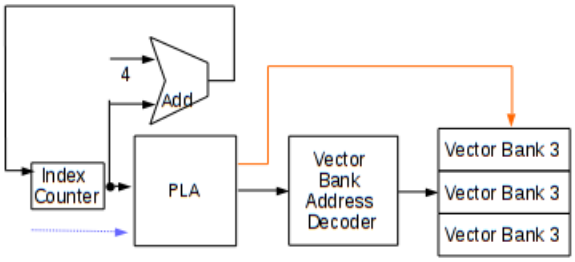

Fig. 2. Decoder for vector bank

Two 4x2 matrices, A and B, are in memory starting from the base 100 and 200 respectively. Add and store result in matrix C starting at 300.

Non-vectorised assembly code:

```
lw  R1, #0
loop: lw R2, 100(R1)
lw  R3, 200(R1)
add R2, R2, R3
st  R2, 300(R1)
add r1, r1, #4
cmp r1, #28
beqz loop
```

Vectorised assembly code for the same

```
vlw  V1, #100
vlw  V2, #200
vadd V3,  V1,  V2
vst  V3, #300
```

Number of instructions:

- *Non Vectorized: 1 + (8 * 7) = 57 Instructions*
- *Vectorized: 4 Instructions*

Reduction in number of Instructions:

$$\text{Decrease Percentage} = \frac{57-4}{57} \times 100 = 92.98\% \text{ reduction}$$

B. Significance

The significant reduction in the Instruction fetch overhead for vectorizable operations is useful in embedded systems thus increasing performance for nominal block operations.

The reduced dependency on memory also means a more **power efficient computation**. Memory writes are power expensive operations.[1]

In recent years, more emphasis is being placed on extremely low footage learning algorithms. Learning tools like back propagation extensively use matrix multiplications to calculate hypotheses. Vectorization on embedded systems will help improve the performance of such applications on embedded systems many times over.

C. Further Research

Adding more ALU's:
There are multiple approaches to pipeling. The more powerful(and costly) of which is **Array processor** method. Adding multiple ALU's will result in a single cycle vector operation.

The limitations of such an approach however may outweigh the performance improvement. The on die 'real-estate' requirements for multiple ALU's are quite significant and can result in a much costlier to manufacture design.

Variable length vector operations:
The current implementation does vector operations on a fixed block size. This will lead to memory inefficiencies analogous to memory overhead in block aligned storage. Additional optional operands that specify the number of elements in the block will need to be implemented.

The hardware overhead includes more registers to store the vector length. More control lines and a more complex logic in the vector PLA, to read from the state registers and reset the Index Counter appropriately.

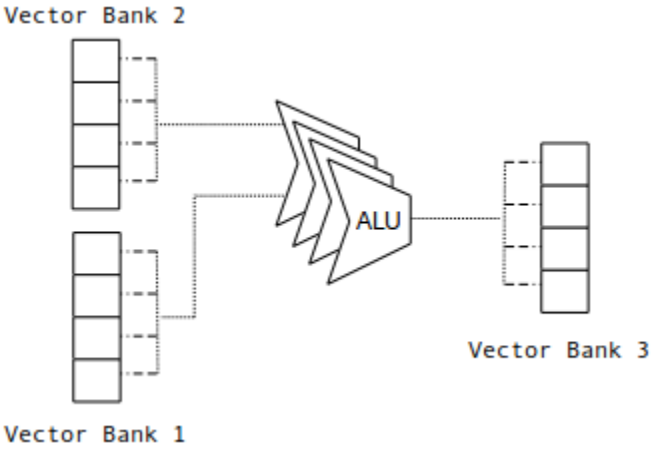

Fig. 3. Multiple ALU's, each with it's own Vector Bank

IV. CONCLUSION

This project is used to demonstrate how to add specialized vectorisation capabilities to architectures found in micro-controllers.

A. Scope for Practical Use

Majority of the IoT (Internet of Things) devices are meant to just collect data and sent it to the cloud for processing. They can be provided with such vectorisation capabilities to carry out very specific computation work and thus reducing latency of output (can especially useful in time critical systems) and the cost of transmitting at each and every step.

ACKNOWLEDGMENT

The authors gratefully acknowledge the guidance of Dr. Rajeev Kumar Singh and Dr. Abhishek Sharma for giving us the opportunity to do research and providing invaluable guidance.

REFERENCES

[1] A.Sharma, Energy Management for Wireless Sensor Network Nodes, May 2011, pp.5
[2] Intel, Atmel AT89S52 Data Sheet, June 2008
[3] https://www.ece.cmu.edu/ ece740/f13/lib/
[4] https://www.kernel.org/pub/linux/kernel/people/geoff/cell/ps3-linux-docs/CellProgrammingTutorial/BasicsOfSIMDProgramming.html