Modify the µCS-51 with Vector Instructions

Assem Badr [1], Ph.D: Abdelmoneim M. Fouda [2], and Prof: Abdelsamie kodb [3]

**[1] Computer Eng. Department, Modern Academy for Engineering & Technology
Cairo, Egypt**

**[2] Computer Eng. Department, Modern Academy for Engineering & Technology
Cairo, Egypt**

**[3] Electrical Eng. Department, Faculty of Engineering, Al-Azhar University
Cairo, Egypt**

Abstract

Computer architects have always strived to increase the overall speed of processing for the CPUs. utilizing a reserved unused machine code "A5H", we can expands the tradition instruction set architecture (ISA) for the µCs-51 family, in this paper we introduce modification of internal architecture for the µCs-51 and their ISA to improve the overall µC performance, specifically we will introduce two innovated vector instructions for the µCs-51 family. The two vector instructions exploiting the data manipulation. The first instruction is to transfer a block of data from specific memory locations to any other memory locations simultaneously, while the other vector instruction is to obtain the minimum data byte value within a block of data bytes. Also we will supply the modified µC with pipeline technique for decreasing the total execution time. Such development improves the total performance of the µC including execution time, and storage ratio.

Keywords: *µCS-51, ISA, Vector instruction, VHDL, pipeline, Amdahl's law, Iron law***.**

1. Introduction

Embedded applications are becoming ever more diverse and complex; processors targeting such applications have an increased tendency to attain a desirable performance via highly specialized instructions tailored for the needs of their targets[1, 2].

Recently, Parallelism is one of the best solutions to achieve high speed of processing, and lowest power consumption for overall speeding application. Parallel processing reduces the execution time taken by any program. The execution time taken by any program is determined by three factors: First, the number of instructions executed, second, number of clock cycles needed to execute each instruction and the third is the length of each clock cycle [3].

There are two forms of parallelism that can be exploited by modern computing machinery to achieve higher performance, the first form of parallelism is called Thread Level Parallelism (TLP) is means the capability to execute independent programs simultaneously using different flows of execution, called threads. The second form is Instructional Level Parallelism (ILP), it mean that executes many instructions in same machine cycle[2].

The conventional general purpose µCs are insufficient to achieve the high performance/cost ratio for advanced communication systems, control system, and digital signal processing (DSP). To satisfy these requirements instruction set design is one of the important issues at which an instruction can be customized for specific applications to make better performance. However the limited encoding space doesn't allow for adding specific complex instructions to the conventional ISA. So it is required to develop the conventional µC in such a way to satisfy a trade-off between reaching the specified application and costs [4]. The developed µC would be dedicated for the specific application, as data manipulation or DSP handling. Therefore this work presents a synthesizable VHDL µC core (and it can be later on implemented on the FPGA chip) for data manipulations.

All conventional µCs-51 family such as "AT89C52" (from Atmel), and more than 1000 advanced µCs-51 such as "TAS3108" (from Texas Instruments, which perform five simultaneous DSP operations per clock cycle) using the traditional instruction set[5]. The conventional µCs-51 executes their instructions in between 12 clock pulses to 48 clock pulses, while the advanced versions of µCs-51 executes their instructions in between few clock pulses to one clock pulse.

From our researches in the advanced versions of µCs-51 family also in the soft-cores of µCs-51, we observed that all corporations concentrated its modification in the hardware architecture; they have been modified the internal structure of µC-51 to improve and enhance the µC enhancement. All modified versions of µCS-51 don't consider or discuss any modification for its instruction set architecture (ISA). This point will be highly investigated in our paper.

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

We developed and modified the internal architecture for the µCS-51 family and their instruction set architecture (ISA) by supplying it with vector instructions to improve their overall performance.
The paper is organized as follows; the next section-2 describes the overall design steps of the developed µC, including the ISA instruction set modification in section-2-1, and the processor architecture modification in section 2-2. While as section-3 discuss the simulation results and processor performance. Finally, section 4 presents an overall conclusion.

2. Design Processor

To develop a novel µC based on a conventional one, different modifications are required including instruction set modification, and architecture modification. This section introduces the necessary basic principles to design modified VHDL code for the internal architecture of the conventional µC-51. The developed VHDL code is obtained by adding the so called "modified instruction decoder" (MID) in addition to the conventional instruction decoder (CID). We can alternate between either MID or CID using the so called "selected instruction decoder flag (SIDF)". Another modification is obtained by inserting VHDL codes representing two added functional units "FU-MBK" and "FU-GMn" respectively. The first one is responsible for transferring block of data bytes simultaneously; while the second on is used to get the minimum byte value among block of data bytes as will be explained in section 2.1. Finally the organization and architecture for the conventional µC must be modified to match these requirements as will be explained in section 2.2.

2.1 Instruction set modification

This section is to modify processor's ISA, as a design methodology, the ISA can be adapted or extended to meet the modern application requirements [6]. ISA modification is obtained by adding two vector instructions, each one of them is a group of individual conventional instructions; these two developed instruction can be used in the field of data manipulations.
The two proposed instructions are associated with the main memory RAM. The first instruction will transfer the contents of 8 successive memory locations to anther 8 successive memory locations through 8 concurrent data buses at same clock pulse. The second one will transfer the contents of 8 successive memory locations to comparator circuit through 8 concurrent data buses, the logical comparator circuit will getting the minimum value among the data bytes which delivered to it simultaneously.

The Direct Memory Access (DMA) is a capability provided by some computer bus architectures that allows data to be sent directly from an attached device (such as a disk drive) to the memory on the computer's motherboard. The microprocessor is freed from involvement with the data transfer, thus speeding up overall computer operation [7]. Our modification similar to the simple DMA architecture, but not completely, it has 8 data channels among addressable locations, and it has a single command to read/write for the main memory, no error detect, parity check, handshaking, …etc, just 8 parallel data bus to transfer 8 bytes concurrently.
From literature survey, and guided with Intel µCS-51 family's data sheets and its manual of instructions set, it's found that the machine code "A5H" is a reserved code which is not used for any operations or tasks, as shown in fig.1.

Hex Code	Mnemonic	Number of Bytes
A4	MUL AB	1
A5	Reserved	1
A6	MOV @R0,direct	2
A7	MOV @R1,direct	2

Fig.1 Partial of conventional µCS-51 instruction set.

Our trend is to utilize the reserved op-code (machine code "A5h") to expand the number of µCS-51family's ISA and adding sophisticated instructions customized for specific multiple data applications. The suggested assembly syntax code "InovConv" uses the reserved machine code "A5H"; this code has toggle alternative action (TAA). Namely, when the "InovConv" code is written by the user it is compiled into "A5h", and alternate as toggling action between either CID (when SIDF = "0") or MID (when SIDF = "1") as shown in fig.2.

Mnemonic	Hex code	SIDF	Enabled instruction decoder
InovConv	A5	0	Conventional Instruction Decoder (CID)
		1	Modified Instruction Decoder (MID)

Fig.2 Toggling stages of A5h.

The developed µC can be used in two states either in its conventional mode of operation (256 instructions) or in the modified mode of operation (255 instructions) therefore the number of instructions for the conventional µC-51 will be expanded from 255 to 511 instructions.
The modified instruction set includes in-order instructions which execute during the original CPU machine cycle (fetching-decoding-executing) according to recommended operations.
A portion of traditional and modified instructions for µC-51 is shown in fig.3; it is illustrates that the machine codes share for both instruction modes. The SIDF specify which decoder is activated. The modified instruction "MOVBK Adr1, Adr2" is a vector instruction which is executed in modified functional unit "FU-

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

MBK". It has double operands, the operand "Adr1" (source operand), it points to the starting location of 8-bytes data block in the RAM. The operand "Adr2" (destination operand), it points to the first location of 8-bytes data block in the RAM which will be receipt data from "FU-MBK". After "MOVBK" has been executed, 8 data bytes transferred from 8 RAM locations which addressed by "Adr1" to others 8 RAM locations which addressed by "Adr2" simultaneously.

The modified instruction "GETMIN Adr1, Adr2" is also a vector instruction which is executed in modified functional unit "FU-GMn", it has double operands, the operand "Adr1" (source operand) points to the starting location of the block of data (8 bytes) in the RAM which will be transfer to a logical comparator (in the "FU-GMn"), the operand "Adr2" (destination operand), it points to the a single memory location in the RAM which receipt the obtained minimum data byte. All operations of "GETMIN" performed simultaneously.

HEX code	conventional (ISA)		modified (ISA)		
	No. of Bytes	Assembly code	No. of Bytes	Assembly code	SIDF
A5	1	Reserved code	1	InovConv ***	0*
A5	1	Reserved code	1	InovConv ***	1**
A6	2	MOV @R0,Direct	6	OUTSeqB N,B,P,S	N.C
A7	2	MOV @R1,Direct	6	INSeq N,B,P,S	N.C
C5	2	XCH A, Direct	2	MovBK Adr1, Adr2	N.C
C9	1	XCH A,R1	2	GetMIN Adr1, Adr2	N.C

*** Boolean alternative action (BAA) ** Causes innovation codes
N.C ..... Not Care * Causes conventional codes

Fig.3 Partial of modified µCS-51 instruction set (ISA).

2.2 Processor's Architecture modification

Adapting the old architecture to a more flexible implementation empowering further developing[2]. This section is to modify the organization and architecture for the conventional µC. Based on Harvard architecture, program and data are accessed on separate buses, having two separate memory spaces (one for instructions, the other is for data), which offer big chance to improve and extend the system architecture by adding more modified blocks[4]. High-level design tools and field-programmable gate arrays (FPGAs) significantly reduce the effort, cost and risk of hardware implementation. These technologies can be incorporated into a manageable and affordable prototyping framework a VLSI-scale “breadboard” for exploring and evaluating new microprocessor designs [8,9]. For this reasons a new architecture will be delivered to modify the conventional µCs-51 by the VHDL over the FPGA technique. As shown in fig.4 there are different modifications for the basic µCs-51architecture, the shaded blocks represent the conventional units while as the other un-shaded blocks represent the added modified blocks. All design was made from scratch using the block diagrams from data sheets for µC-51 family[10].

The 1st modification is obtained by adding MID-unit and modified instruction register (MIR-unit) beside conventional instruction register (CIR-unit) and the CID-unit, all of them is controlled by SIDF.

Initially the SIDF is set to '0', the CID is activated, so the traditional µC-51 instruction set will be fetched, decode, and then executed. When using the code "InovConv" (denominated by user's program) the SIDF is set to '1', the MID is activated and the modified 8051 instruction set will be fetched, decode, and then executed. Furthermore, when the "InovConv" code is written again by user, it toggles the SIDF to logic '0' thus enabling for CID again, and so on.

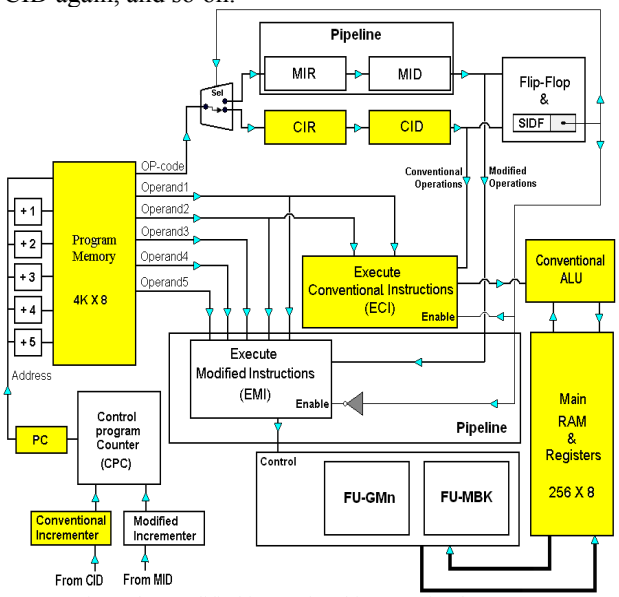

Fig.4 The Modified internal architecture for the µC-51

The 2nd modification is particular for the program memory's interface; it is performed by increasing the number of data bus associated with program memory (from single bus to 6 data bus). The instructions of the conventional µC-51 are represented by up to maximum 3 bytes in the program memory. Each byte is fetched one by one through single bus. While the instructions of the modified version fetches up to maximum 6 sequencing bytes stored in the program memory as show in fig.4. Such developed construction is capable for fetching all adequate bytes simultaneously. The recommended memory location fetched starting from the value stored in the program counter (PC). The desired number of fetched bytes specified by the modified unit designated as control program counter (CPC). The increment in the program counter related to the absolute values stored in the CID or MID.

The 3rd modification is particular for the original RAM's interface; by adding 256 output data buses for transferring any adequate number of data bytes from any

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

successive memory locations to modified functional units concurrently. Also by adding 256 input data buses for transferring any adequate number of data bytes, it transfer from modified functional units to any successive memory locations concurrently.

Moreover the 4th modification supply the modified µC by 3-state pipeline to execute each modified instruction in single clock pulse. The 1st step for fetching up to maximum 6 bytes from the program memory, the 2nd step specifies the recommended operation and transfers the adequate operands to the proper functional unit, the 3rd step for executing instruction.

The 5th modification includes the creation of so the called "execute modified instruction EMI" which is responsible for controlling the additional modified functional units such as "FU-MBK" and "FU-GMn".

Furthermore 6th modification creates several functional units to execute the modified instructions; each functional unit is specified for a particular instruction to eliminate the structure hazards. Each In-order functional unit is designed to execute its instruction in only single clock pulse. So each modified In-order instruction has instruction cycle with 3 clock pulses. Thus the pipeline stages will be overlapped the instructions to execute one instruction per clock. Now we will describe first functional unit "FU-MBK".

Pipeline_ROM_1.vhd

```
621  when x"C5"=>          --Vector instruction <MovBK Rn,Rm>
622        for n in 0 to 7 loop --3 bytes
623           RAM_location(conv_integer(Genrl_Oprd_Byte_2)+n)<=
624           RAM_location(conv_integer(Genrl_Oprd_Byte_1)+n);
625        end loop;
```

Fig.5 VHDL code represent the functional unit "FU-MBK"

The modified µC designed based on VHDL, the vector instruction "MOVBK Adr1, Adr2" represented in the VHDL as shown in fig.5. When the machine code "C5h" (see fig.3) has been fetched, it decoded at line "621", the two lines "623" and "624" represent the data bus connected the RAM location which pointed by the 1st operand "Genrl_Oprd_Byte_1" to the RAM location which pointed by the 2nd operand "Genrl_Oprd_Byte_2". Both pointers are added by "n", the FOR-LOOP in line "622" increment "n" from 0 to 7, it means that 8 parallel data buses connected between 8 successive memory locations (start from initial value of 1st operand) and others 8 memory locations (start from initial value of 2nd operand).

The functional unit "FU-MBK" includes 8 Multiplexers (256X1) and 8 Demultiplexers (1X256) as shown in fig.6, the 8 MUXs connected with the 8 DEMUXs through 8 parallel data buses, the 1st data bus connected between "MUX0" and "DEMUX0", the 2nd data bus connected between "MUX1" and "DEMUX1" and so on. Each MUX has 256 inputs connected with the outputs of the 256 RAM locations, the first operand "Adr1" specifies the first source memory location "Rs", the subsequent incremented for the operand "Adr1" specify the 7 consequence RAM locations "Rs+1","Rs+2" up to "Rs+7". Also each DEMUX has 256 output connected with inputs of 256 RAM locations. The second operand "Adr2" specifies the first destination memory location "Rd", the subsequent incremented for the operand "Adr2" specify 7 consequence RAM locations "Rs+1","Rs+2" up to "Rs+7". If the "MOVBK" activated by using vector instruction "MOVBK Adr1, Adr2", 8 data bytes will be transferred from 8 successive memory locations to others 8 successive memory locations through the 8 parallel data buses simultaneously.

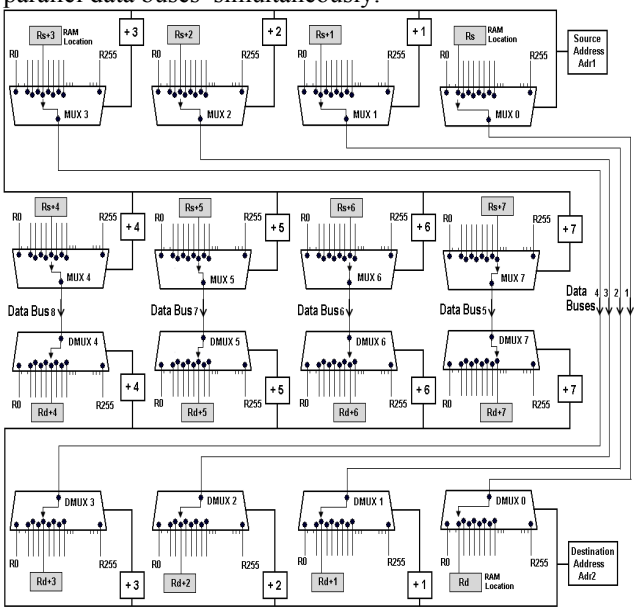

Fig.6 The block diagram of functional unit "FU-MBK"

The second modified vector instruction "GETMIN Adr1, Adr2" represented in the VHDL as shown in fig.7. When the machine code "C9h" (see fig.3) has been fetched, it decoded at line "664", the data buffer "V_Max_Min" loaded initially by value "FFh", the two lines "667" and "668" represent the data bus connected between the RAM location which pointed by the 1st operand "Genrl_Oprd_Byte_1" and a logical comparator, it compare between the value in the data bus and the stored value in the data buffer "V_Max_Min", if the value in the data bus less than the stored instantaneous value in the data buffer "V_Max_Min" then this value will be stored in the data buffer again. The FOR-LOOP in the line "666" increment "n" from 0 to 7, it means that 8 parallel data buses connected between 8 successive memory locations (start from initial value of 1st operand) and 8 comparators.

The comparators responsible for getting the minimum data bytes among the values received from the 8 memory locations. The line "671" transfer the lowest data byte to the memory location which pointed by 2nd operand "Genrl_Oprd_Byte_2".

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

Pipeline_ROM_1.vhd*

```
664  when x"C9"=>      --Vector instruction <GetMin Rn,Rm>
665            V_Max_Min:= X"FF";
666          for n in 0 to 7 loop --3 bytes
667             if RAM_location(conv_integer(Genrl_Oprd_Byte_1)+n) < V_Max_Min then
668                   V_Max_Min := RAM_location(conv_integer(Genrl_Oprd_Byte_1)+n);
669                   else null;
670             end if;
671                   RAM_location(conv_integer(Genrl_Oprd_Byte_2))<= V_Max_Min;
672          end loop;
673
```

Fig.7 VHDL code represent the functional unit "FU-GMn"

The functional unit "FU-GMn" includes 8 Multiplexers (256X1), 8 Multiplexers (2X1), 8 comparators, data buffer and Demultiplexer (1X256) as shown in fig.8. The 8 MUXs (256X1) is connected with the 8 comparator through 8 parallel data buses, the 1st data bus is connected between "MUX0" and "Comp0", the 2nd data bus connected between "MUX1" and "Comp1" and so on. Each MUX has 256 inputs connected with the outputs of the 256 RAM locations. The first operand "Adr1" specifies the first memory location "Rs". The subsequent incremented for the operand "Adr1" specifies the 7 consequence RAM locations "Rs+1", "Rs+2" up to "Rs+7". Each comparator compare between the values received from memory location and the instantaneous value stored in the data buffer, if the received value from memory less than the instantaneous value stored in the data buffer it load into the data buffer. The second operand "Adr2" specifies the destination memory location "Rd", all operations in the "FU-GMn" performed simultaneously.

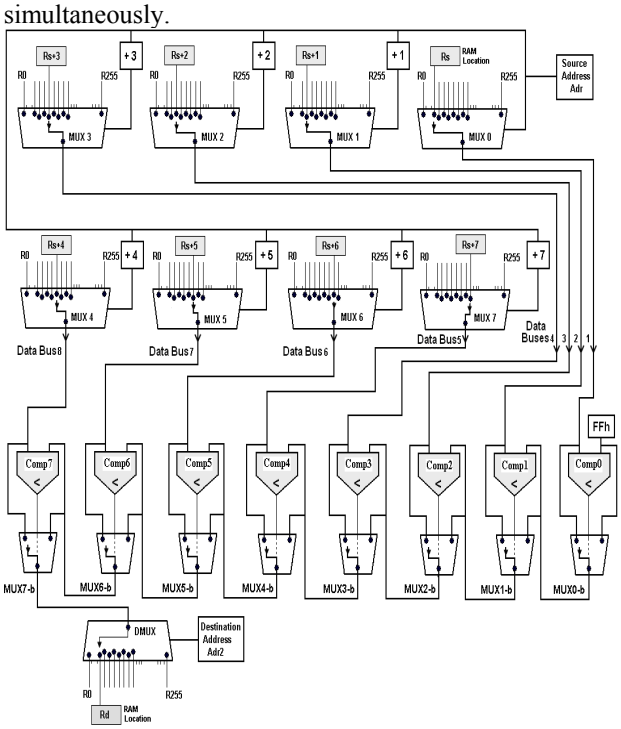

Fig.8 The block diagram of functional unit "FU-GMn"

3. Simulation and analysis

A large number of simulators have been developed to help investigate microprocessor design issues. These simulators can be broadly grouped into functional and performance simulators. A functional simulator provides a virtual implementation such that the outward functionality of a design is emulated. A performance simulator models the inner workings of a design, in only as much detail as necessary, to extract the desired quantitative measures of some dynamic behavior[11]. The "Mentor-graphic Modelsim" simulator provides possibility for advanced debugging and simulation of the VHDL code[12].

In the last few years, the Modelsim package become the more popular VHDL simulator, it has several different parameterized performance simulations, it can simulate parallel architecture, it has advanced signal timing analyzer, and it can investigate the contents of a memory locations instantaneously.

The processors are getting faster, yet application performance not keeping pace. On large commercial applications, average cycles-per-instruction (CPI) values may be as high[13]. The conventional µCs-51 executes their instructions in average from 1 to 4 cycles per instruction, and the advanced µCs-51 executes the same instructions only in one cycle per instruction.

Form the simulation point of view, it's required to simulate the execution scenario of the two developed command using a well known simulation package. The "Mentor-graphic Modelsim SE 6.5" simulator provides possibility for advanced debugging and simulation of the VHDL code [14, 15].

3.1 Pipeline simulation

The conventional µCs-51 is non-pipeline architecture, it executes its instructions in subsequent steps in its CPU, the CPU architecture built with many modules, and each module carries out single step. Recently this architecture is inefficient in speed of processing because it carried out its steps consequently, namely, just one module is activated during the instruction cycle, while the other modules are waiting it. By supplying the conventional µCs-51 family with the pipeline technique, it provides the capability of making those modules works together in parallel to improve the overall program execution. Those modules can't be worked in single instruction concurrently, because each step depends on the others. The pipeline include multiple of identical stages, each stage treat a single instruction with different treatments than the others. The pipelining doesn't completely parallel execution because just one instruction has been performed at a time.

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

Modified µC-51 with pipeline technique has only 3 main steps (fetch-decode-execute). In fig.9 shows portion of machine code program of the modified µC based on VHDL. Line "59" represents the op-code "C5h" and its associated operands "00h" and "20h".

Line 60 represents the op-code "C9h" and its associated operands "20h" and "40h". In line "61" the op-code "D5h" which has single operand "00h" and so on.

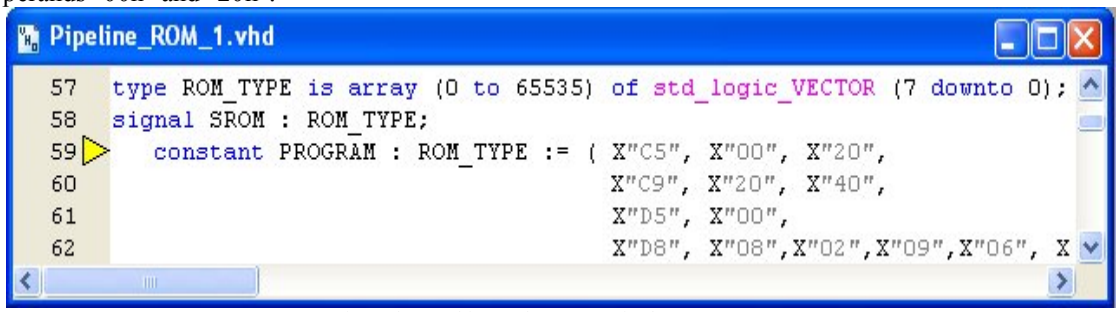

Fig.9 The machine code program in the VHDL.

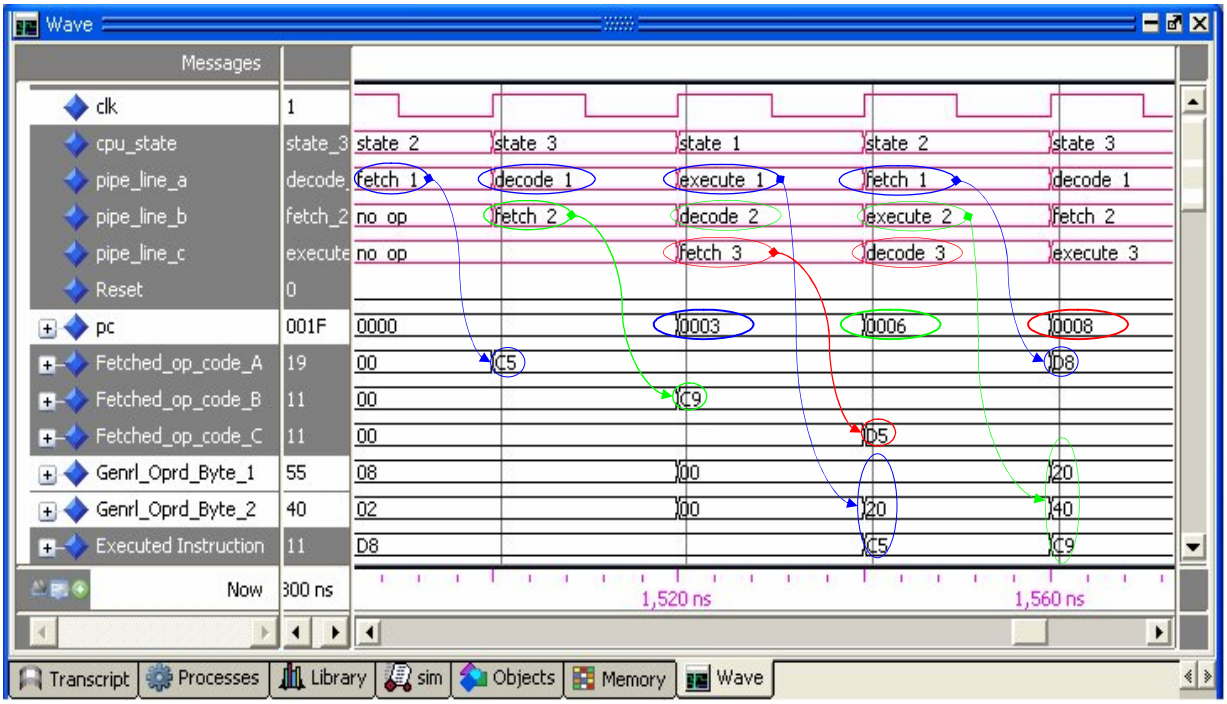

Fig.10 The simulation of 3 states pipeline execute each instruction in single clock pulse

The fig.10 indicates the simulated 3 states pipeline using "Modelsim", the trace line "Clk" represents the global clock pulses for the modified µC, the line "CPU_state" represents the current pipeline state which vary among state_1, state_2 and state_3. At time "1,480 ns" the stage "pipe_line_a" fetched the first op-code "C5h", after one clock pulse (20 ns), at time "1,500 ns" the "pipe_line_a" decoded the first op-code "C5h" during the fetched of 2nd op-code "C9h" in the "pipe_line_b". After one clock pulse, at time "1,520 ns" the "pipe_line_a" load the two operands (00h and 20h) into the adequate functional unit and executed the vector instruction "C5h" (as we will be described later) during the decoding of op-code "C9h" in the "pipe_line_b", and during fetching the op-code "D5" in the "pipe_line_c". At time "1,540 ns" the "pipe_line_a" fetched the 4th op-code "D8h" during the "pipe_line_b" loaded the two operands (20h and 40h) into the adequate functional unit and executing the vector instruction "C9h" (as we will be described later) during the decoded of op-code "D5h" in the "pipe_line_c".

From the above illustration it is clear that our modified µC executes its modified instructions in 3 steps; each step spent one clock pulse. By adding the µC with 3 states pipeline it can execute one in- order instruction per one clock pulse.

Thus; the pipelining enhanced the speed of processing for modified µC by a factor given as in equation (1)

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

$$Pipeline\ Speedup = \frac{(Average\ instruction\ time)_{unpipelined}}{(Average\ instruction\ time)_{pipelined}} \quad (1)$$

$$= \frac{Num\ of\ instructions * 3\ clock\ pulses}{Num\ of\ instructions * 1\ clock\ puls} = 3\ times$$

3.2 Vector instructions simulation

In this section we measure the performance of the modified µC relative to both conventional µC (such as AT89C52) and advanced µC (such as TAS3108). Assume there are two specific tasks for data manipulations to be achieved. The 1st task is transferring block of data from 8 RAM locations to others 8 RAM locations, the 2nd task is getting the minimum data byte among block of data bytes. We will utilize the simulator software package "Prog-Studio" to perform the two mentioned tasks based on the conventional instruction set, while the simulator "Modelsim" to perform the same mentioned tasks for the modified instructions. The performance parameters include two main parameters, the 1st parameter is the number of machine cycles required to perform each task, the 2nd parameter is the number of stored machine code bytes (in the program memory) required for each task.

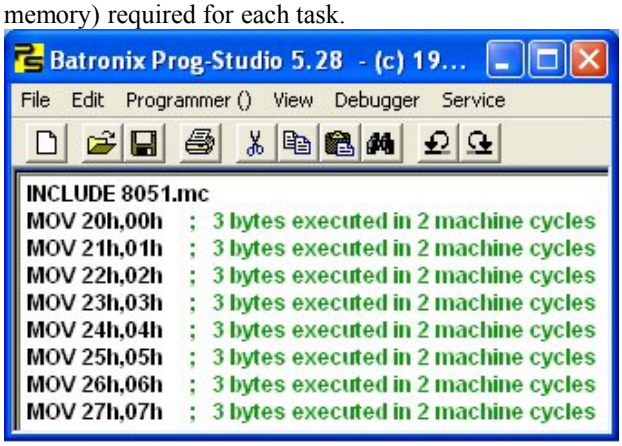

Fig.11 Assembly codes to transfer block of data

With respect to the first task (transfer block of data), assume the following scenario, it is required to transfer the contents of 8 successive RAM locations starting from register R0 to R7 into anther 8 successive RAM locations starting from address "20h" to address "27h". The conventional instructions to perform this task are shown in fig.11. Each line in this code transfer individual bytes from source RAM location to the corresponding destination RAM location. For more illustration, the first line code transfer contents of R0 to location "20h". Similarly, the next lines of the code move the sequenced RAM locations (R1, R2,…,R7) to another consequence RAM locations (21h, 22h,…, 27h) respectively.

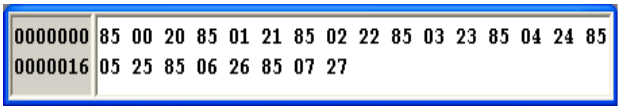

Fig.12 The machine bytes for transfer block of data.

Each conventional instruction occupied 3 bytes in the program memory, and executed in 2 machine cycles, so the overall conventional code for transferring 8 data bytes occupied (3*8=24 bytes) as shown in fig.12, and the mentioned conventional code needs totally (2*8=16 machine cycles) for execution.

The same task (transfer data block) performed by the modified vector instruction "MOVBK", its machine code illustrates in the line "59" in the VHDL code as shown in fig.9, the "C5h" represents the op-code, the "00h" represents the source operand, and the "20h" represents the destination operand.

The execution of "MOVBK" illustrates in the RAM forms of the simulator "Modelsim"; the fig.13a indicates the initial values of the RAM locations, specifically the highlighted different values from address "00h" to the "07h", and the highlighted "00h" values from address "20h" to "27h". After "MOVBK" has been executed the data block which started at address "00h" copied into the RAM locations which started at "20h" as shown in the fig.13b.

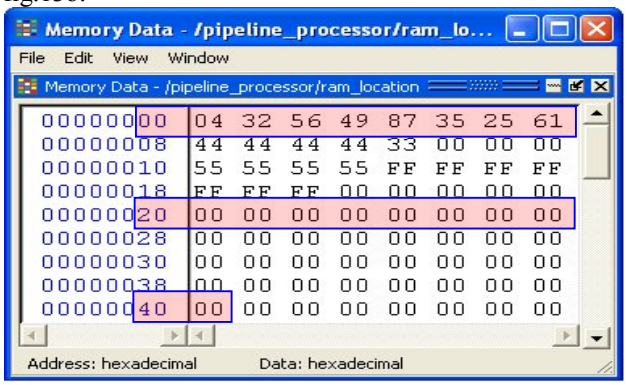

Fig.13a Initial values for RAM form of the simulator "Modelsim"

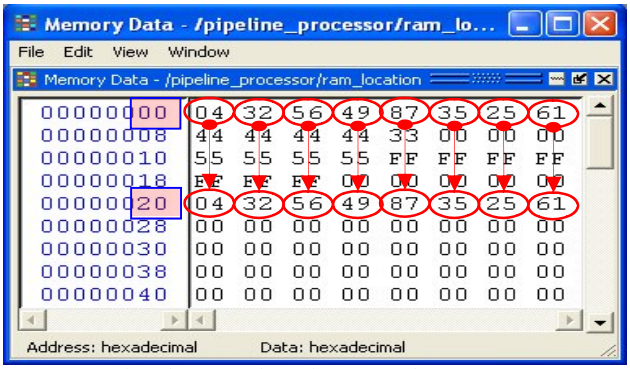

Fig.13b RAM values after executed "MovBK"

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

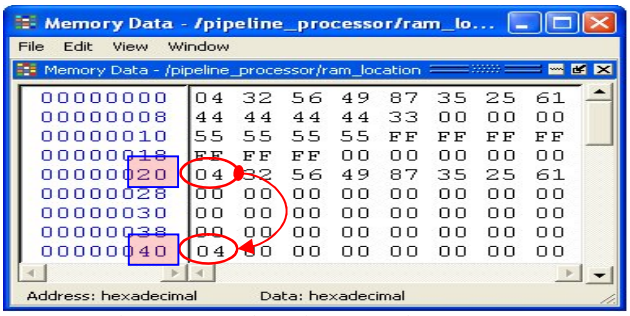

Fig.13c RAM values after executed "GETMIN"

Now we will calculate the time spent to perform the mentioned task for the conventional µCs-51, advanced µCs-51 and our modified µC-51. Firstly we will calculate the duration time for each machine cycle. The suffix "C" denoted for conventional µC, and the suffix "A" denoted for advanced µC while the suffix "M" denoted for modified µC.

From the conventional µCs-51 data sheets, the duration time for one machine cycle T_C is equal to 12 of clock pulses (T_{cp}) The maximum operating frequency for the conventional µCs-51 is 33 MHz so the duration time for one machine cycle as in equation (2).

$$T_C = 12\ T_{cp} = \frac{12}{f_{cp}} = \frac{12}{33\times10^6} = 0.3636\ \mu s \quad (2)$$

Similar the newer advanced µCs-51 such as DS89C420, DS89C430, DS89C440, DS89C450 from Dallas Semiconductor and TAS3108 from Texas Semiconductor, their machine cycle duration time T_A are equal to single clock pulse (T_{cp}) as in equation (3)

$$T_A = T_{cp} = \frac{1}{f_{cp}} = \frac{1}{33\times10^6} = 0.0303\ \mu s \quad (3)$$

Similar our modified µC machine cycle duration time T_M include single clock pulse (T_{cp}) as in equation (4), we selected 50 MHz as the operating frequency for our modified µC.

$$T_M = T_{cp} = \frac{1}{f_{cp}} = \frac{1}{50\times10^6} = 0.02\ \mu s \quad (4)$$

With aid of "Iron law" as in equation (5) we can calculate the total duration time for each program code[16, 17].

$$\frac{Time}{program} = \frac{No.of\ instructions}{program} \times \frac{No.of\ cycles}{instruction} \times \frac{time}{cycle} \quad (5)$$

The mentioned first task (move 8 data bytes) has assembly code with 8 similar conventional instructions equivalent to instruction "MOVBK", each instruction executed in 2 machine cycles. The conventional executed time will deliver in equation (6)

$$\left(time/program\right)_C = [8\times2]\times\frac{time}{cycle} \quad (6)$$
$$= 16\times T_C = 16\times0.3636 = 5.8\ \mu s$$

Similar the advanced executed time will deliver in equation (7)

$$\left(time/program\right)_A = [8\times2]\times\frac{time}{cycle} \quad (7)$$
$$= 16\times T_A = 16\times0.0303 = 0.4848\ \mu s$$

The instruction "MOVBK" executed in single machine cycle which include one clock pulse, the executed time of modified instruction is delivering in equation (8).

$$\left(time/program\right)_M = [1]\times\frac{time}{cycle} = T_M = 0.02\ \mu s \quad (8)$$

With aid of the Amdahl's law we can calculate the enhanced speeding up as shown in equation[16](9).

$$Speed\ up_{enhanced} = \frac{T_{exe\_old}}{T_{exe\_new}} \quad (9)$$

With aid of both Amdahl's law and Iron law we will determine the enhancement speeding ratio of the modified instruction "MOVBK" relative to the traditional instructions for conventional µCs-51 as in equation (10).

$$Speed\ up = \frac{T_{exe\_c}}{T_{exe\_m}} = \frac{\left(time/program\right)_C}{\left(time/program\right)_M} = \frac{5.8\ \mu s}{0.02\ \mu s} = 290.9\ \text{times} \quad (10)$$

Also we will determine the enhancement speeding ratio of the modified instruction "MOVBK" relative to the traditional instructions for advanced µCs-51 as in equation (11).

$$Speed\ up = \frac{\left(time/program\right)_A}{\left(time/program\right)_M} = \frac{0.4848\ \mu s}{0.02\ \mu s} = 24.24\ \text{times} \quad (11)$$

The storage ratio is an another important factor for evaluate the modified instructions, the instruction "MOVBK" needs 3 bytes to store its op-code and operands while the multiple traditional instructions which designed for performing the same task need (3*8=24) bytes as shown in fig.12, so the enhancement

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

of the storage ratio relative to conventional or advanced µCs will be as in equation (12)

$$Storage_{enhanced} = \left(\frac{Storage_{Advanced}}{Storage_{Modified}} \right) \quad (12)$$

$$= \frac{24\ bytes}{3\ bytes} = 8\ \text{times}$$

```
INCLUDE 89C52.mc
IF 21h < 20h THEN MOV 20h,21h   ; 12 bytes executed in 9 machine cycles
IF 22h < 20h THEN MOV 20h,22h   ; 12 bytes executed in 9 machine cycles
IF 23h < 20h THEN MOV 20h,23h   ; 12 bytes executed in 9 machine cycles
IF 24h < 20h THEN MOV 20h,24h   ; 12 bytes executed in 9 machine cycles
IF 25h < 20h THEN MOV 20h,25h   ; 12 bytes executed in 9 machine cycles
IF 26h < 20h THEN MOV 20h,26h   ; 12 bytes executed in 9 machine cycles
IF 27h < 20h THEN MOV 20h,27h   ; 12 bytes executed in 9 machine cycles
MOV 40h,20h                     ; 3 bytes executed in 2 machine cycles
```

Fig.14 Assembly codes to get minimum data byte

With respect to the second task (get max data byte), assume the following scenario, it is required to obtain the minimum value among the contents of 8 successive RAM locations start from address "20h" to "27h" and store it into the RAM location "40h". The conventional instructions to perform this task are shown in fig.14. Each IF-statement code check individual bytes with RAM location "20h", for example the first line code check the content of "21h" if it grater than the content of location "20h" it will be transferred into location "21h". And so on until the greatest value will be transferred to RAM location "40h".

DecAdr.	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000000	C0	E0	E5	21	C3	95	20	D0	E0	50	03	85	21	20	C0	E0
0000016	E5	22	C3	95	20	D0	E0	50	03	85	22	20	C0	E0	E5	23
0000032	C3	95	20	D0	E0	50	03	85	23	20	C0	E0	E5	24	C3	95
0000048	20	D0	E0	50	03	85	24	20	C0	E0	E5	25	C3	95	20	D0
0000064	E0	50	03	85	25	20	C0	E0	E5	26	C3	95	20	D0	E0	50
0000080	03	85	26	20	C0	E0	E5	27	C3	95	20	D0	E0	50	03	85
0000096	27	20	85	20	40											

Fig.15 The machine bytes for get minimum data byte

Each conventional IF-statement occupied 14 bytes in the program memory, and executed in 9 machine cycles. The last conventional instruction occupied 3 bytes and executed in 2 machine cycles. So the overall conventional code for getting minimum data byte occupied (7*14 + 3=101 bytes) as shown in fig.15.

The same task (get max data byte) performed by the modified vector instruction "GETMIN", its machine code illustrates in the line "60" in the VHDL code as shown in fig.9, the "C9h" represents the op-code, the "20h" represents the source operand, and the "40h" represents the destination operand.

The execution of "GETMIN" illustrates in the memory forms of the simulator "Modelsim"; the fig.13b indicates the values of the RAM locations, specifically the highlighted different values from address "20h" to the "27h", and the highlighted address "40h". After "GETMIN" has been determined the minimum data bytes "04h" which has address "20h" copied into the RAM locations which has address "40h" as shown in the fig.13c.

The conventional executed time for the mentioned second task will deliver in equation (13)

$$\left(time/program \right)_C = \left[(7 \times 9) + 2 \right] \times \frac{time}{cycle} \quad (13)$$

$$= 65 \times T_C = 65 \times 0.3636 = 23.64\ \mu s$$

Similar the advanced executed time will deliver in equation (14)

$$\left(time/program \right)_A = \left[7 \times 9 + 2 \right] \times \frac{time}{cycle} \quad (14)$$

$$= 65 \times T_A = 65 \times 0.0303 = 1.97\ \mu s$$

The instruction "GETMIN" executed in single machine cycle which include one clock pulse, the executed time of modified instruction is delivering in equation (15).

$$\left(time/program \right)_M = [1] \times \frac{time}{cycle} = T_M = 0.02\ \mu s \quad (15)$$

With aid of both Amdahl's law and Iron law we will determine the enhancement speeding ratio of the modified instruction "GETMIN" relative to the traditional instructions for conventional µCs-51 as in equation (16).

$$Speed\ up = \frac{\left(time/program \right)_C}{\left(time/program \right)_M} \quad (16)$$

$$= \frac{23.64\ \mu s}{0.02\ \mu s} = 1181.8\ \text{times}$$

Also we will determine the enhancement speeding ratio of the modified instruction "GETMIN" relative to the traditional instructions for advanced µCs-51 as in equation (17).

$$Speedup = \frac{\left(time/program \right)_A}{\left(time/program \right)_M} \quad (17)$$

$$= \frac{1.97\ \mu s}{0.02 \mu s} = 98.48\ \text{times}$$

The storage ratio is an another important factor for evaluate the modified instructions, the instruction "GETMIN" needs 3 bytes to store its op-code and operands while the multiple traditional instructions which designed for performing the same task need 101 bytes as shown in fig.15, so the enhancement of the storage ratio relative to conventional or advanced µCs will be as in equation (18)

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

$$Storage_{enhanced} = \left(\frac{Storage_{Advanced}}{Storage_{Modified}} \right) \quad (18)$$
$$= \frac{((7*12)+3)\ bytes}{3\ bytes} = 29 \text{ times}$$

4. Conclusion

In this paper we introduced the idea to expand the ISA of the famous µCs-51 family; we occupied only two machine codes from the extended ISA, now the framework is opened to fill up the new ISA with more creative instructions. We introduced two innovated vector instructions based on VHDL, the two modified instructions able to meet widely application domains, without any conflict with the main µC's ISA and its characteristics. The first instruction "MOVBK Adr1, Adr2" designed for transferring set of eight data bytes from consequence memory locations pointed by "Adr1" to others eight consequence memory locations pointed by "Adr2", the second instruction "GETMIN Adr1, Adr2" designed for getting the minimum data bytes for a set of eight data bytes from µC's memory location pointed by "Adr1" and store the minimum data bytes in memory location which pointed by "Adr2". The two vector instructions executed only in 3 clock pulses, while after adding the pipeline they executed in single clock pulse. Moreover the same tasks of the two vector instructions can be performed using multiple of conventional instructions, it executed in a great number of clock pulses. Finally, we have been compared our two vector instructions relative to their corresponding conventional instructions.

References

[1] Jonghee M. Youn, Sechul Shin, "A New Addressing Mode for the Encoding Space Problem on Embedded Processors", 7th Symposium on Application Specific Processors (SASP), IEEE, 2009.
[2] Abdelmoneim M. Fouda, Assem Badr and Abdelsamie kodb, "Modify the µCS-51 Architecture to SIMD, VLIW and Superscalar µC ", IJCSI, Vol.9, issue 1, No 1, Jan 2012.
[3] M.Suaib, A.Palaty, K.Sambhav, "Architecture of SIMD Type Vector Processor ", International Journal of Computer Applications, Volume 20, April 2011.
[4] Elena Roxana Buhus, "A System-On-Chip Approach in Designing a Dedicated RISC Microcontroller Unit Using the Field-Programmable Gate Array", Fifth International Conference on Systems, computer society IEEE, 2010.
[5] Web site, http://www.keil.com/dd/, last accessed on the 21st of April 2012, hour 15:14.
[6] Joseph Sharkey and Dmitry Ponomarev, " Balancing ILP and TLP in SMT Architectures through Out-of-Order Instruction Dispatch", 5th International Conference on parallel processing, IEEE, 2006.
[7] Sajjan G.Shiva, "Computer Organization, Design and Architecture", Marcel Deker,NY, third edition, revised and expanded, pp.181-450,2000
[8] Peter Ashenden and Jim lewis, "The designer's guide to VHDL", Elsevier Inc., third edition, 2008.
[9] Joydeep Ray, "High-Level Modeling and FPGA Prototyping of Microprocessors ", IEEE, 2003.
[10] Web site, www.Philips.com, "80C51 family programmer's guide and instruction set" last accessed on the 17th of Aug 2011, hour 02:03
[11] Joydeep Ray, "High-Level Modeling and FPGA Prototyping of Microprocessors", IEEE, 2003.
[12] R.C Cofer and Ben Harding "Rapid System Prototyping with FPGAs-Accelerating the Design Process", Burlington, USA, pp.35-150, 2006.
[13] Jeffrey Dean and James Hicks, "Hardware Support for Instruction-Level Profiling on Out-of-Order Processors", IEEE, 1997
[14] Web site, http://www.xilinx.com, last accessed on the 19th of Jan 2011, hour 20:00
[15] R.C Cofer and Ben Harding "Rapid System Prototyping with FPGAs-Accelerating the Design Process", Burlington, USA, pp.35-150, 2006
[16] A.R.Alameldeen, D.A.Wood, "IPC Considered Harmful for Multiprocessor Workloads", computer society IEEE, Aug 2006.
[17] H. Weik Martin, "Fiber Optics Standard Dictionary", Hall book, 3rd edition, 1997.

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.