Advances in Computational Research
ISSN: 0975-3273 & E-ISSN: 0975-9085, Volume 5, Issue 1, 2013, pp.-153-156.
Available online at http://www.bioinfopublication.org/jouarchive.php?opt=&jouid=BPJ0000187

A NOVEL ALGORITHM FOR EFFICIENT MEMORY MANAGEMENT WITH PARALLEL COPROCESSOR FOR GARBAGE COLLECTION

KAPADIA V.V.[1]* AND THAKAR V.K.[2]

[1]Department of Computer Science & Engineering, The Maharaja Sayajirao University, Vadodara- 390 020, Gujarat, India.
[2]Department of Electronics and Communication, A.D. Patel Institute of Technology, New Vallabh Vidyanagar- 388 120, Gujarat, India.
*Corresponding Author: Email- kapadia_viral2005@yahoo.co.in

Received: July 15, 2013; Accepted: December 24, 2013

Abstract- Memory management on which researchers are mainly focusing are the different techniques related to memory management i.e. garbage collection techniques, scheduling, real time system, user oriented design and many more. The major problem found out in memory management is processor and operating system delays. Here a new design of processor is discussed in the paper that will improve to the over all system performance.

Keywords- Memory management, garbage collection techniques, scheduling, real time system

Citation: Kapadia V.V. and Thakar V.K. (2013) A Novel Algorithm for Efficient Memory Management with Parallel Coprocessor for Garbage Collection. Advances in Computational Research, ISSN: 0975-3273 & E-ISSN: 0975-9085, Volume 5, Issue 1, pp.-153-156.

Introduction

Now a day's size of the application program is increasing dramatically, same time data size is also increasing so we require that some trade of should be done in mark-sweep and coping collector. However, advanced garbage collection techniques have been developed to ameliorate the situation in two different ways:

One way for the solution of this problem is to split the work traditionally performed in a single collector invocation for smaller amount of time parts, incremental algorithms address the problem of increasingly longer interruptions. Second, by deriving some method to find out probable life time of the objects, generation-based algorithms aim to reduce the total amount of collection work that needs to be carried out at all.

Garbage Collection

However, we have to some price for automatic garbage collection. A general collection strategy is to allow the main program run for as long as possible, allocating new objects as needed. Only when memory resources become exhausted is the garbage collector activated, returning unused areas of memory to the allocator, and enabling the main program to continue [1].

Thus, the execution of the main program is completely halted while the garbage collector carries out its work. In many settings, such garbage collection pauses are completely acceptable.

During large and complex batch computations, it is of little interest whether, at some point in time, the computer is working to solve the given problem or merely performing garbage collection; what counts is the total amount of time spent. In fact, even though garbage collectors are very general in nature and usually not fine-tuned to handle any particular computation, they are usually highly optimized pieces of code and may out-perform many handwritten deallocation routines [2,3].

However, interruptions to the execution flow of the main program can be totally unacceptable. For instance, in real-time applications, a long garbage collection pause may in itself render the program useless. Examples of real-time applications include sound and animation, robot motion control, and certain communication systems [1].

Processor Design for Garbage Collection

We have designed the instruction set for the model. The processor we are designing is an 4-bit processor with following instruction set.

D/I	S0	S1	S2	D1	D2	D3

D/I: Direct/Indirect instruction

S0, S1, S2: For Op-code

D0, D1, D2, D3: Data

Op-codes for Different Operation

000 - Update the allocation

001 - Increase Reference

010 - for freeing the memory

100 - Updating the data

101 - for increasing reference of memory

Now we are discussing the overall approach found out for efficient memory utilization of memory.

Model

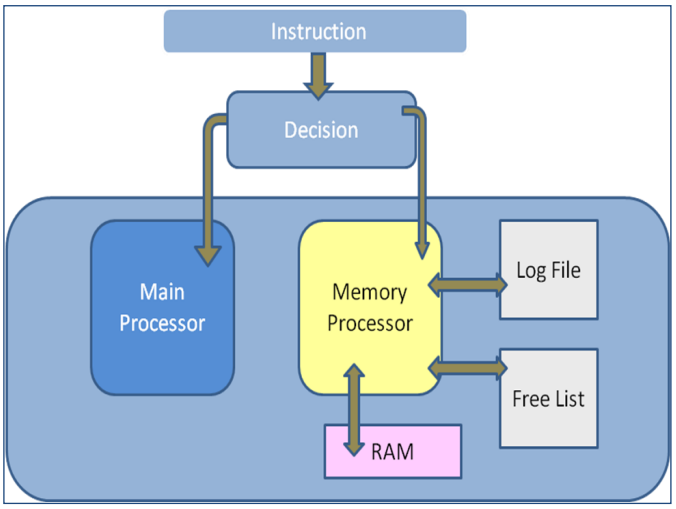

Fig. 1- Hardware Model

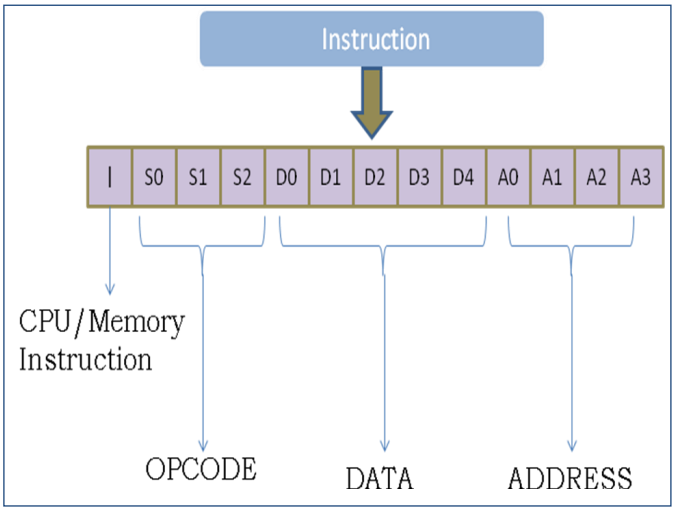

Fig. 2- Instruction Word

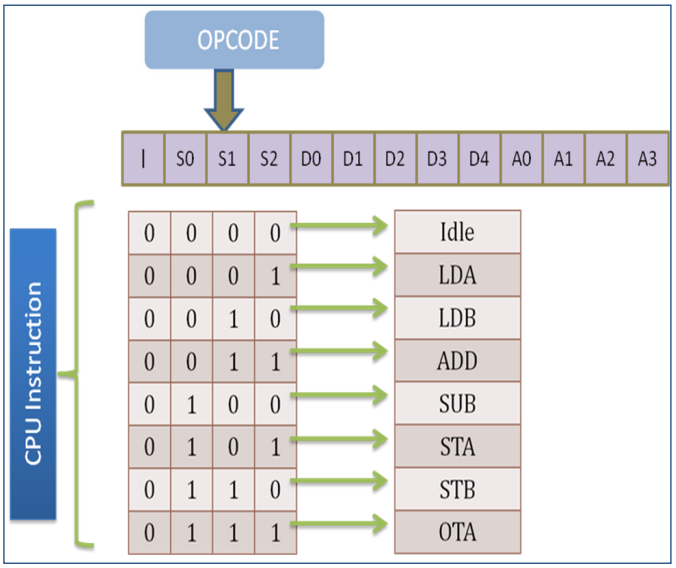

Fig. 3- CPU OPCODE

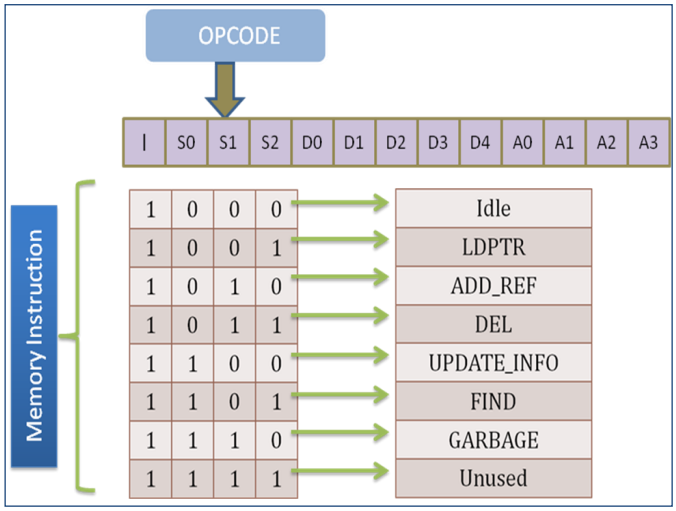

Fig. 4- Memory OPCODE

We have designed a small module for separate processor for garbage collection and the design is shown in [Fig-1]. [Fig-2] shows the instruction format of 8 bit processor that illustrates opcode and operand of the instruction word. [Fig-3] illustrates the CPU instruction that contains Addition, multiplication etc operations of traditional processor and [Fig-4] illustrates the instruction set for the memory processor. The processor switched between memory and CPU more through the first bit.

Process

One of the major task in this research area is model designing and model has been designed. This model is designed using tool VHDL and it is simulated on Xilinx ISE 9.2 Edition. It is simulated on the same tool. The major problem as discussed in the previous section is speed and lack of parallelism. As these features are very difficult to achieve under software part, designing new hardware becomes a necessity. Software works under the scenario shown below.

Steps Performed by Software Garbage Collection

- Instruction is loaded and now it has to be executed by Processor [Fig-5].
- First Memory Instruction is loaded and now it has to be executed by Processor [Fig-6].
- Memory instruction needs memory and has to call garbage collection process [Fig-7].
- Software garbage collector reclaim garbage and gives control back to processor mean while, processor was ideal but it wasn't able to execute next instruction although we have large instruction word and processor has already fetched the data at once [Fig-8].
- As memory instruction was over the processor executed the CPU instruction [Fig-9].

One of the probable solutions to this problem is designing a dedicated hardware processor for garbage collection process. [Fig-10] illustrates the case where a new processor is designed and dedicated for memory management as well as garbage collection. Additionally it also reflects that instructions are executed simultaneously on different processor (Assuming that one is CPU instruction and other is memory instruction and don't have any dependencies of each other) which will definitely help in improving the overall through put of the system.

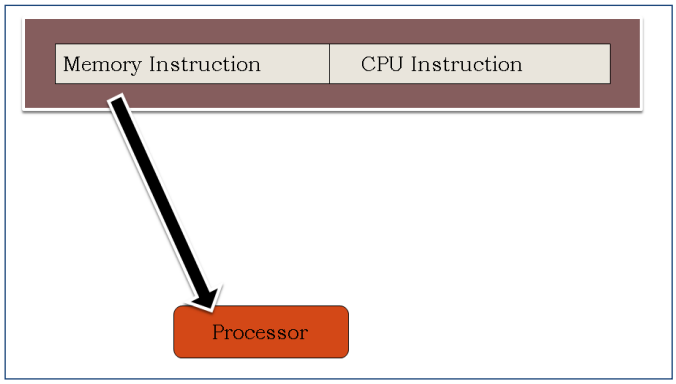

Fig. 5- Instruction referred to Processor

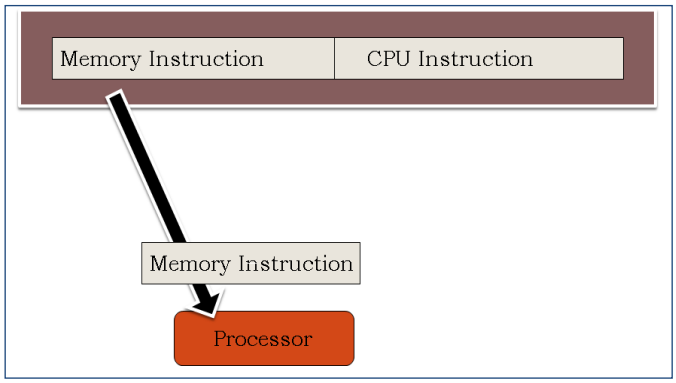

Fig. 6- Memory Instruction referred to Processor

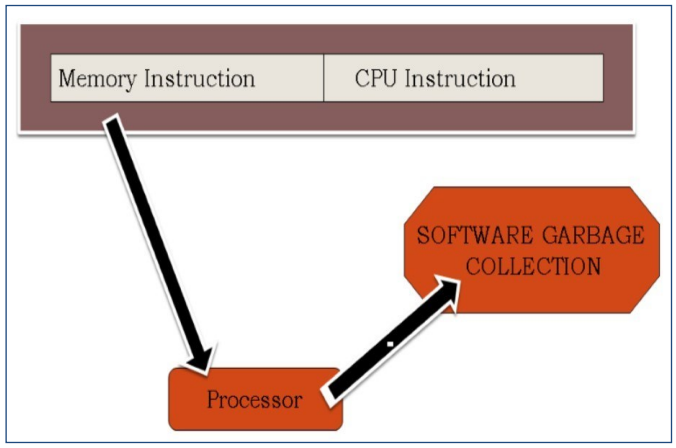

Fig. 7- Hands over the control to GC

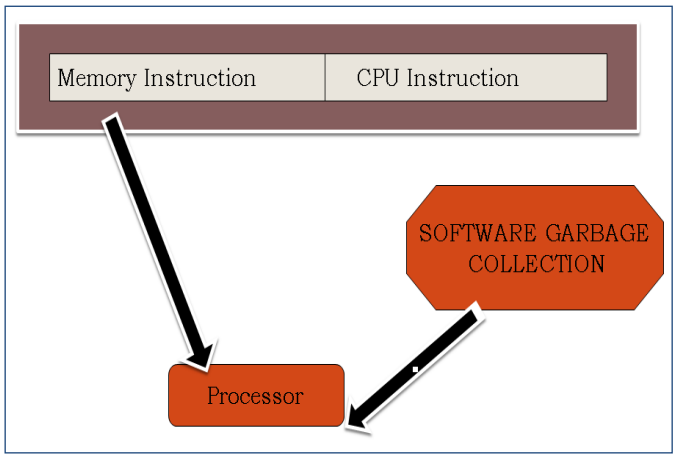

Fig. 8- GC responds back

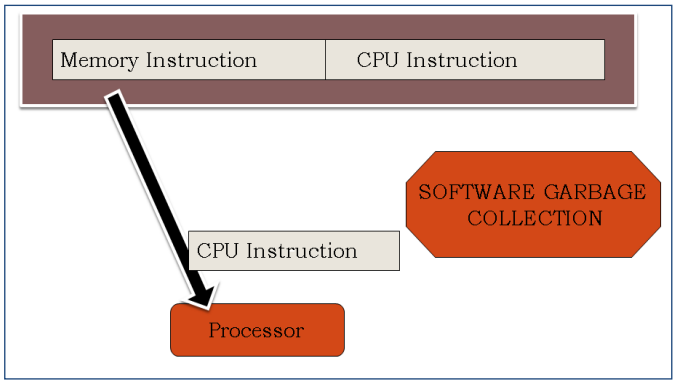

Fig. 9- Normal Instruction

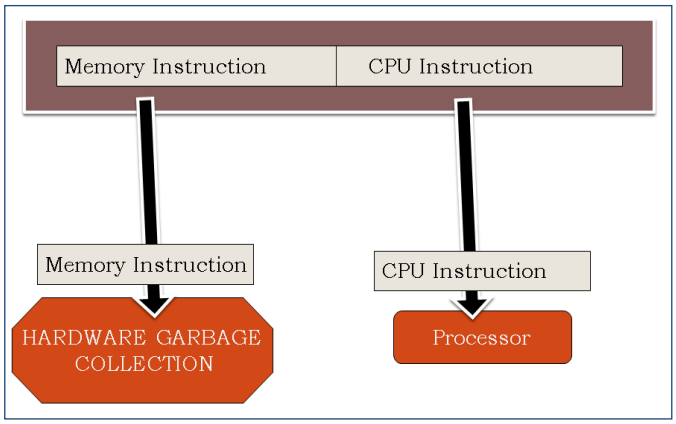

Fig. 10- Both Memory & CPU instruction

One of the major task in this research area is model designing and model has been designed. This model is designed using tool VHDL and it is simulated on Xilinx ISE 9.2 Edition.

It is simulated on the same tool. The major problem as discussed in the previous section is speed and lack of parallelism. As these features are very difficult to achieve under software part, designing new hardware becomes a necessity.

Algorithm

At any time related to memory processor may be confined into any one state i.e.

- *If* (Memory Available → True) have it
- *Else* (Memory Available → False) Needs Garbage collection.

This process needs to be identified as an instruction generated from the software.

If Software requires some processing and in effect to execute that task the processor requires the memory (In our approach we are identifying the same thing through instruction i.e. memory instruction)

- *If* (Garbage Collection Processor → Available)

The memory instruction should be again moved to waiting queue and till the garbage collection Co-processor find the memory block through under defined process.

Find the memory in Young generation area as there is a very higher probability of getting the free space and the life of objects residing in Young Generation is very less.

- *If* (FREE → Available)

Directly return the address of the space to processor and Update

the free List.

- *Else* (FREE → Not Available)

Need to move the most mature object in young generation (Also considering the remaining time left for use of the Object) and return the Free space to the processor.

This approach can also help in the pipelining process of processor and scheduler.

If the processor is Idle it will instruct the coprocessor of garbage collector to collect the garbage in incremental manner.

- *Garbage collection Process* (Counter → 25)

It collects the garbage for 25 objects and stops

- *While* Processor is Free
- *Repeat* (Step → i) and update free list

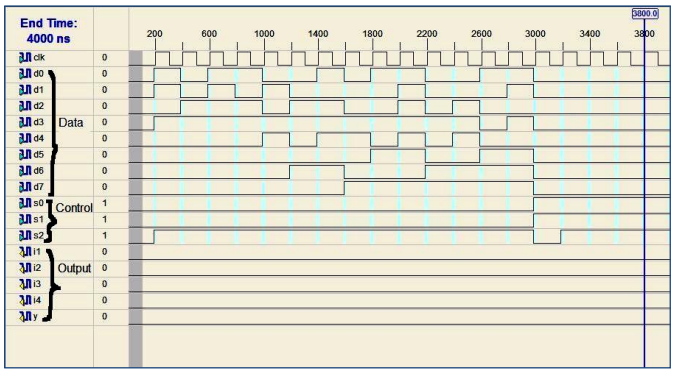

Fig. 11- Input Wave Form

Results

Inputs given to the system are in waveforms that is represented by [Fig-11] (Input wave form) that represents the input instruction given to the processor as per the model given in [Fig-2] and the corresponding output is shown in [Fig-12] that shows the garbage collection process is activated when ever instruction is not issued or i/p to the system is idle. Hence the table of results indicate that accuracy is 100% of finding the garbage hence its can be termed as consistent system.

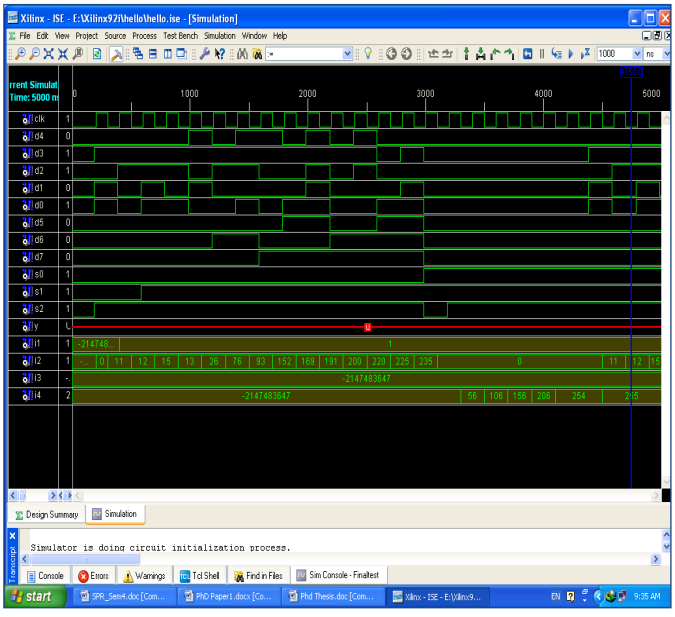

Fig. 12- Output Wave Form

Conclusion

Adopting this approach a new shift is likely to be introduced as none of the process has to wait for processor when garbage collection is performed, except for (exceptional memory requirement). The results indicate that if such co-processors are designed with efficiency then we may have a great speed up in overall system performance.

Conflicts of Interest: None declared.

References

[1] Seligmann J., Grarup S. (1993) *Incremental Mature Garbage Collection*, M.Sc. Thesis, Computer Science Department, Aarhus University, Denmark.

[2] Appel A.W. (1987) *Information Processing Letters*, 25(4), 275-279.

[3] Zorn B. (1993) *Software: Practice and Experience*, 23(7), 733-756.

[4] Cohen A., Rohou E. (2010) *Proceedings of the 47th Design Automation Conference*, ACM, 102-107.

[5] Srisa-an W., Lo C.T., Chang J.M. (2003) *IEEE Transactions on Mobile Computing*, 2(2), 89-101.

[6] Chang Y., Wellings A. (2010) *IEEE Transactions on Computers*, 59(8), 1063-1075.