RYERSON UNIVERSITY

Faculty of Engineering, Architecture, and Science

Department of Electrical and Computer Engineering

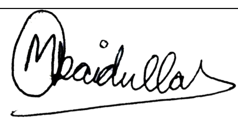
Course Number	EE8207
Course Title	High Performance Computer System Design
Semester/Year	Winter/2016

Instructor	Dr. Nagi N. Mekhiel

Assignment No.	**1**

Assignment Title	Installing and Using SimpleScalar Simulator

Submission Date	29^{th} January 2016
Due Date	29^{th} January 2016

Student Name	Muhammad Obaidullah
Student ID.	500671408
Signature*	

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: www.ryerson.ca/senate/current/pol60.pdf.

1 Objectives of the lab

1. Installing SimpleScalar
2. Measure ISA Statistics (frequency of each type and cost associated)
3. Running Different Applications
4. Measure Application Performance
5. Generate Traces for the Application

2 What is SimpleScalar ?

SimpleScalar is an architectural simulator which simulates the behavior of a computing device. We can use SimpleScalar to leverage faster, more flexible software development cycle. It can be used to study the issues and performance of any software code and design more efficient compilers that exploit pipelining features.

Figure 2.1: Black box analogy of SimpleScalar simulator. [1]

Different executables of SimpleScalar are available to execute as follows:

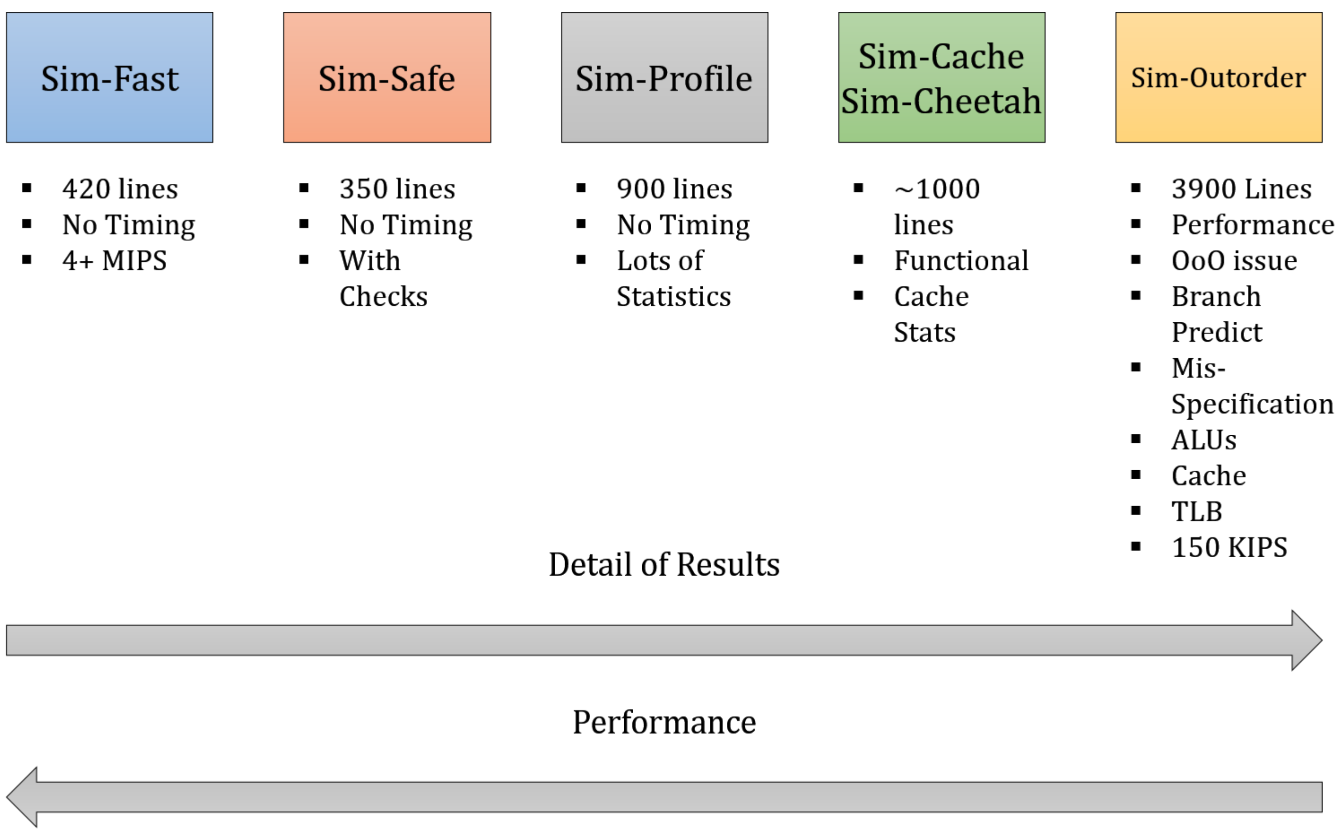


Figure 2.2: Different SimpleScalar executables which emulate different types of Instruction Set Architecture (ISA). Sim-Outorder is the most complex ISA emulator with support for out-of-order instruction execution.[1]

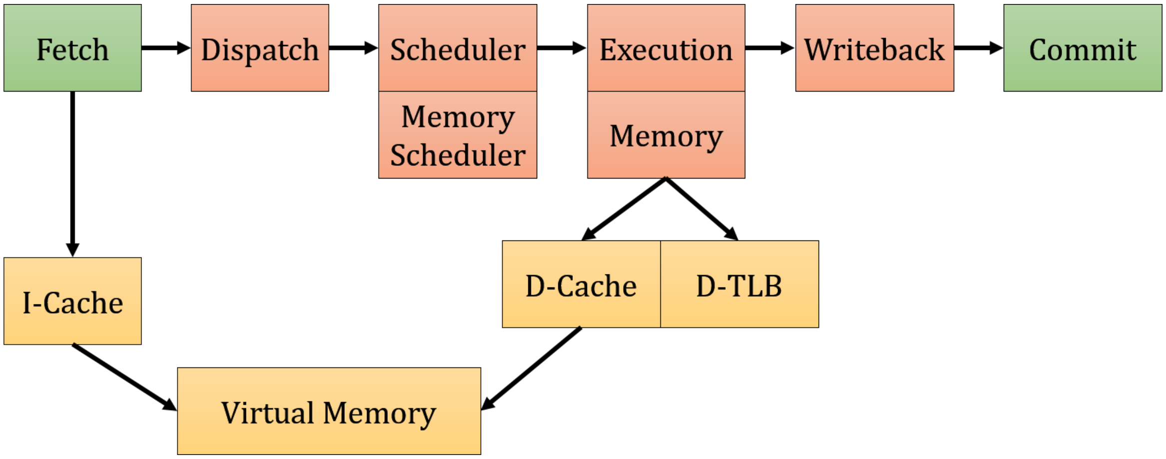


Figure 2.3: Pipeline for sim-outorder emulator which supports out of order intruction execution.[2]

3 Installing SimpleScalar

Typing the following command installs Simple Scalar Version 3.0

```
>> SScalarsetup
```

4 Measuring Statistics

Execution of the executable simulates run of the program and reports several useful characteristics.

4.1 Compiling the Code to Run

1. The C code which is to be executed on the simulator can be compiled using the compiler at $/SimpleScaler - 3.0d/bin/sslittle - na - sstrix - gcc$. The command to generate an object file for the code is:

```
~/SimpleScaler-3.0d >> ./bin/sslittle-na-sstrix-gcc -c ~/Documents/coe818/bench1.c
```

2. To generate an executable to run using the simulator, following command is typed:

```
~/SimpleScaler-3.0d >> ./bin/sslittle-na-sstrix-gcc ~/Documents/coe818/bench1.c
```

   This will generate a *.out file which is executable using the SimpleScaler simulator.

3. To execute the a.out executable, following command is typed:

```
~/SimpleScaler-3.0d >> ./simplesim-3.0/sim-safe a.out
```

4.2 Application 1: Dhrystone Program

Dhrystone program is an old benchmark which was written in 1984 by Reinhold Weicker and measured integer performance of processors and compilers. Since then, it has been replaced by more complex benchmarking programs such as SPEC and CoreMark.

Dhrystone evaluates general-purpose integer performance of the DUT (Device Under Test). However it does not resemble any real-life program, is very susceptible to compiler optimizations, and due to the small code size, it may fit in the instruction cache of a modern CPU hence diluting instruction fetch performance.

Following are the results from running this benchmark program:

```
sim: ** simulation statistics **
sim_num_insn              533507945 # total number of instructions executed
sim_num_refs              215504362 # total number of loads and stores executed
sim_elapsed_time                 19 # total simulation time in seconds
sim_inst_rate         28079365.5263 # simulation speed (in insts/sec)
ld_text_base             0x00400000 # program text (code) segment base
ld_text_size                  28080 # program text (code) size in bytes
ld_data_base             0x10000000 # program initialized data segment base
ld_data_size                  11876 # program init'ed `.data' and uninit'ed `.bss' size in bytes
ld_stack_base            0x7fffc000 # program stack segment base (highest address in stack)
ld_stack_size                 16384 # program initial stack size
ld_prog_entry            0x00400140 # program entry point (initial PC)
ld_environ_base          0x7fff8000 # program environment base address address
ld_target_big_endian              0 # target executable endian-ness, non-zero if big endian
mem.page_count                   17 # total number of pages allocated
mem.page_mem                    68k # total size of memory pages allocated
mem.ptab_misses                  19 # total first level page table misses
mem.ptab_accesses        2565216032 # total page table accesses
mem.ptab_miss_rate           0.0000 # first level page table miss rate
```

4.3 Application 2: Hydro Fragment Program

This benchmark program contains 1 normal for loop which iterates 1000 times and 2 nested for loops which loops $1000 \times 1000 = 1,000,000$ times. In total, the loop C instructions to execute are $2 \times (1,000,000) + 1,000 = 2,001,000$ times.

Since this C program works with doubles and integers, floating point and integer functionality of the DUT (Device Under Test) are evaluated.

Following are the results from running this benchmark program:

```
sim: ** simulation statistics **
sim_num_insn                 809943 # total number of instructions executed
sim_num_refs                 255435 # total number of loads and stores executed
sim_elapsed_time                  1 # total simulation time in seconds
sim_inst_rate           809943.0000 # simulation speed (in insts/sec)
ld_text_base             0x00400000 # program text (code) segment base
ld_text_size                  24096 # program text (code) size in bytes
ld_data_base             0x10000000 # program initialized data segment base
ld_data_size                   4096 # program init'ed `.data' and uninit'ed `.bss' size in bytes
ld_stack_base            0x7fffc000 # program stack segment base (highest address in stack)
ld_stack_size                 16384 # program initial stack size
ld_prog_entry            0x00400140 # program entry point (initial PC)
ld_environ_base          0x7fff8000 # program environment base address address
ld_target_big_endian              0 # target executable endian-ness, non-zero if big endian
mem.page_count                   13 # total number of pages allocated
mem.page_mem                    52k # total size of memory pages allocated
mem.ptab_misses                  13 # total first level page table misses
mem.ptab_accesses           4102180 # total page table accesses
mem.ptab_miss_rate           0.0000 # first level page table miss rate
```

4.4 Application 3: Dhrystone Program

This program is the same as application 1 because the code is similar. However, there are very slight changes in the simulation results.

Following are the results from running this benchmark program:

```
sim: ** simulation statistics **
sim_num_insn                533507901 # total number of instructions executed
sim_num_refs                215504359 # total number of loads and stores executed
sim_elapsed_time                   19 # total simulation time in seconds
sim_inst_rate           28079363.2105 # simulation speed (in insts/sec)
ld_text_base               0x00400000 # program text (code) segment base
ld_text_size                    28080 # program text (code) size in bytes
ld_data_base               0x10000000 # program initialized data segment base
ld_data_size                    11876 # program init'ed '.data' and uninit'ed '.bss' size in bytes
ld_stack_base              0x7fffc000 # program stack segment base (highest address in stack)
ld_stack_size                   16384 # program initial stack size
ld_prog_entry              0x00400140 # program entry point (initial PC)
ld_environ_base            0x7fff8000 # program environment base address address
ld_target_big_endian                0 # target executable endian-ness, non-zero if big endian
mem.page_count                     17 # total number of pages allocated
mem.page_mem                      68k # total size of memory pages allocated
mem.ptab_misses                    19 # total first level page table misses
mem.ptab_accesses          2565215844 # total page table accesses
mem.ptab_miss_rate             0.0000 # first level page table miss rate
```

5 Traces for Applications

Sim-outorder produces detailed history of all instructions executed including instruction stage transitions.

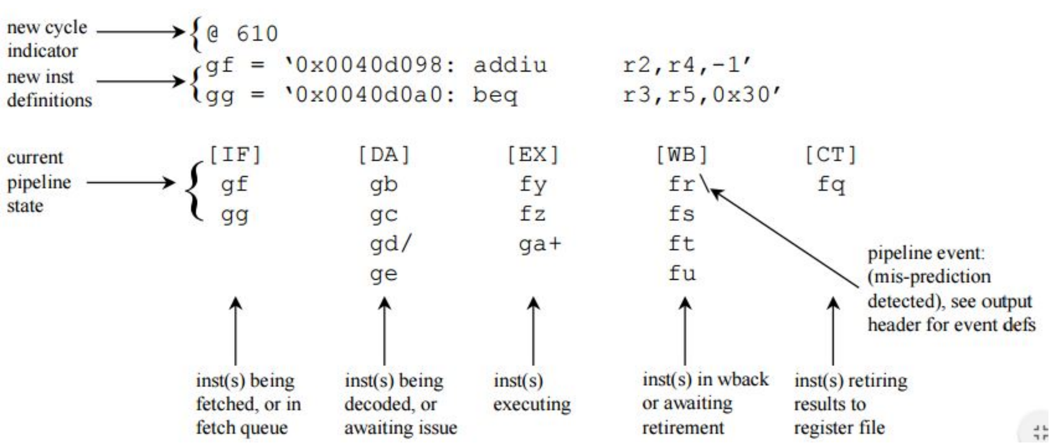


Figure 5.1: Reading and analyzing trace files. [1]

Traces are generated by using the following command:

```
sim-outorder -ptrace FOO.trc :1000 test-math
```

To view the trace file, following command is run to instantiate pipeview program so that the trace file is parsed and is displayed in a proper manner:

```
pipeview.pl FOO.trc
```

5.1 Application 1: Dhrystone

Complete trace files are not attached due to huge size. But cycles which are important are given below:

```
@ 32
+ 0 0x00400140 0x00000000 lw        r16,0(r29)
* 0 IF 0x00000003
+ 1 0x00400148 0x00000000 lui       r28,0x1001
* 1 IF 0x00000000
+ 2 0x00400150 0x00000000 addiu     r28,r28,-31904
* 2 IF 0x00000000
+ 3 0x00400158 0x00000000 addiu     r17,r29,4
* 3 IF 0x00000000
@ 33
+ 4 0x00400140 0x00000000 [internal ld/st]
* 4 DA 0x00000000
* 0 DA 0x00000000
* 1 DA 0x00000000
* 2 DA 0x00000000
* 3 DA 0x00000000
@ 34
* 0 EX 0x00000010
* 1 EX 0x00000000
* 3 EX 0x00000000
@ 35
* 3 WB 0x00000000
* 1 WB 0x00000000
* 0 WB 0x00000000
* 4 EX 0x00000003
* 2 EX 0x00000000
@ 36
* 2 WB 0x00000000
```

```
@ 72
+ 9 0x00400180 0x00000000 sw        r18,-32588(r28)
* 9 IF 0x00000001
+ 10 0x00400188 0x00000000 addiu     r29,r29,-24
* 10 IF 0x00000000
+ 11 0x00400190 0x00000000 addu      r4,r0,r16
* 11 IF 0x00000000
+ 12 0x00400198 0x00000000 addu      r5,r0,r17
* 12 IF 0x00000000
@ 73
+ 13 0x00400180 0x00000000 [internal ld/st]
* 13 DA 0x00000000
* 9 DA 0x00000000
* 10 DA 0x00000000
* 11 DA 0x00000000
* 12 DA 0x00000000
@ 74
* 9 EX 0x00000010
* 10 EX 0x00000000
* 11 EX 0x00000000
* 12 EX 0x00000000
@ 75
* 12 WB 0x00000000
* 11 WB 0x00000000
* 10 WB 0x00000000
* 9 WB 0x00000000
* 13 WB 0x00000000
@ 76
* 13 CT 0x00000003
- 13
* 9 CT 0x00000003
- 9
* 10 CT 0x00000000
- 10
* 11 CT 0x00000000
- 11
* 12 CT 0x00000000
- 12
```

Figure 5.2: Generated trace file for application 1. Where it clearly be seen that the instructions are fetched, decoded, executed, and then data is written back.

5.2 Application 2: Hydro Fragment

Complete trace files are not attached due to huge size. But cycles which are important are given below:

```
@ 32
+ 0 0x00400140 0x00000000 lw        r16,0(r29)
* 0 IF 0x00000003
+ 1 0x00400148 0x00000000 lui       r28,0x1001
* 1 IF 0x00000000
+ 2 0x00400150 0x00000000 addiu     r28,r28,-32032
* 2 IF 0x00000000
+ 3 0x00400158 0x00000000 addiu     r17,r29,4
* 3 IF 0x00000000
@ 33
+ 4 0x00400140 0x00000000 [internal ld/st]
* 4 DA 0x00000000
* 0 DA 0x00000000
* 1 DA 0x00000000
* 2 DA 0x00000000
* 3 DA 0x00000000
@ 34
* 0 EX 0x00000010
* 1 EX 0x00000000
* 3 EX 0x00000000
@ 35
* 3 WB 0x00000000
* 1 WB 0x00000000
* 0 WB 0x00000000
* 4 EX 0x00000003
* 2 EX 0x00000000
@ 36
* 2 WB 0x00000000
```

```
@ 337
+ 44 0x00402d50 0x00000000 addiu     r29,r29,-16
* 44 IF 0x00000003
+ 45 0x00402d58 0x00000000 andi      r5,r5,255
* 45 IF 0x00000000
@ 338
* 44 DA 0x00000000
* 45 DA 0x00000000
@ 339
* 44 EX 0x00000000
* 45 EX 0x00000000
@ 340
* 45 WB 0x00000000
* 44 WB 0x00000000
@ 341
* 44 CT 0x00000000
- 44
* 45 CT 0x00000000
- 45
@ 342
@ 343
```

Figure 5.3: Generated trace file for application 2.

```
@ 617
em = `0x00402da0: lui       r8,0x7efe'
en = `0x00402da8: ori       r8,r8,65279'
eo = `0x00402db0: sll       r2,r5,8'
ep = `0x00402db8: or        r9,r5,r2'

 [IF]      [DA]      [EX]      [WB]      [CT]
  em                  el        ek        ei
  en                                      ej
  eo
  ep

@ 618
eq = `0x00402dc0: sll       r2,r9,16'
er = `0x00402dc8: or        r9,r9,r2'
es = `0x00402dd0: lw        r6,0(r4)'
et = `0x00402dd8: nor       r7,r0,r8'

 [IF]      [DA]      [EX]      [WB]      [CT]
  eq        em                  el        ek
  er        en
  es        eo
  et        ep

@ 619
eu = `0x00402dd0: [internal ld/st]'
ev = `0x00402de0: addiu     r4,r4,4'
ew = `0x00402de8: addu      r3,r6,r8'
ex = `0x00402df0: nor       r2,r0,r6'
ey = `0x00402df8: xor       r3,r3,r2'

 [IF]      [DA]      [EX]      [WB]      [CT]
  ev        en        em                  el
  ew        ep        eo
  ex        eq
  ey        er
            es
            et
```

Figure 5.4: Command line output when using pipeview program to see trace files.

5.3 Application 3: Dhrystone

Complete trace files are not attached due to huge size. But cycles which are important are given below:

```
@ 115
+ 16 0x00401800 0x00000000 addiu     r29,r29,-24
* 16 IF 0x00000003
+ 17 0x00401808 0x00000000 sw        r31,16(r29)
* 17 IF 0x00000000
+ 18 0x00401810 0x00000000 jal       0x402d80
* 18 IF 0x00000000
@ 116
* 16 DA 0x00000000
+ 19 0x00401808 0x00000000 [internal ld/st]
* 19 DA 0x00000000
* 17 DA 0x00000000
* 18 DA 0x00000004
@ 117
* 18 EX 0x00000000
* 16 EX 0x00000000
@ 118
* 16 WB 0x00000000
* 18 WB 0x00000000
* 17 EX 0x00000010
@ 119
* 16 CT 0x00000000
- 16
* 17 WB 0x00000000
* 19 WB 0x00000000
@ 120
* 19 CT 0x00000003
- 19
* 17 CT 0x00000003
- 17
* 18 CT 0x00000000
```

```
@ 260
+ 38 0x00404200 0x00000000 sw        r16,16(r29)
* 38 IF 0x00000001
+ 39 0x00404208 0x00000000 bne       r17,r0,0x20
* 39 IF 0x00000000
@ 261
+ 40 0x00404200 0x00000000 [internal ld/st]
* 40 DA 0x00000000
* 38 DA 0x00000000
* 39 DA 0x00000004
@ 262
* 39 EX 0x00000000
* 38 EX 0x00000010
@ 263
* 38 WB 0x00000000
* 39 WB 0x00000000
* 40 WB 0x00000000
@ 264
* 40 CT 0x00000000
- 40
* 38 CT 0x00000000
- 38
* 39 CT 0x00000000
- 39
@ 265
@ 266
```

Figure 5.5: Generated trace file for application 3.

```
@ 607

 [IF]      [DA]      [EX]      [WB]      [CT]
            ei                  eg        ef
            ej                            eh
            ek
            el/

@ 608

 [IF]      [DA]      [EX]      [WB]      [CT]
            ek        ei                  eg
            el/       ej

@ 609

 [IF]      [DA]      [EX]      [WB]      [CT]
            el/       ek        ei
                                ej
```

Figure 5.6: Command line output when using pipeview program to see trace files.

6 Conclusions

1. Since, there were no memory page misses in any of the applications tested, it is possible that the programs were too short to account for cache miss rate which is evident and plays a vital role in real-world programs.

2. Speed of the simulation or instructions executed over time reduced when floating point operations are added in application 2 (Hydro Fragment Program). This is because of additional pipeline hazards introduced due to floating point unit operations.

3. Program size for applications 1 and 3 is 28K while for application 2 is 24K. Both of these programs are small enough to fit inside any modern CPU's instruction cache. Therefore results obtained from these program might not be close to real hardware.

4. Percentage of load/stores in different applications according to formula $\%\ Of Load/Store = \frac{sim\_num\_refs}{sim\_num\_insn} \times 100\%$ is given in the figure below:

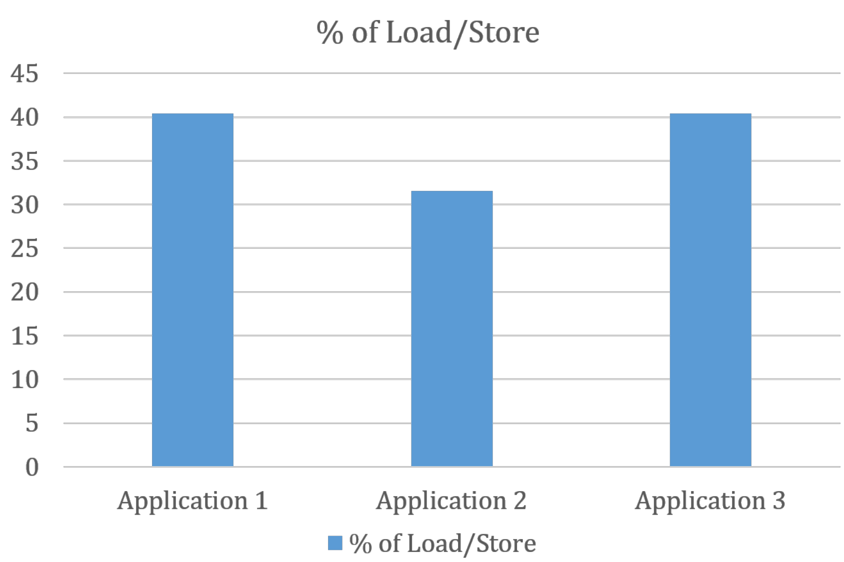


Figure 6.1: Application 2 requires less load/store instructions compared to Dhrystone applications.

5. Since applications 1 and 3 require more load/store instructions than application 2, it is more feasible to use accumulator based architecture. However, this will increase the memory bandwidth required by the processor. If not many cores are attached with the memory access bus, then it is a good option to use accumulator based ISA for dhrystone applications.

REFERENCES

[1] S. LLC. (2001, December) Simplescalar tutorial slides. SimpleScalar LLC. [Online]. Available: http://www.simplescalar.com/docs/simple_tutorial_v4.pdf

[2] D. B. T. M. Austin. (1997) The simplescalar tool set, version 2.0. SimpleScalar LLC. 2395 Timbercrest Court, Ann Arbor, MI 48105. [Online]. Available: http://www.simplescalar.com/docs/users_guide_v2.pdf