Comparison between Intel®Atom™and ARM Cortex-A53

Behnam Saeedi

Abstract

The Intel®'s Atom™was developed during 2008 by Intel®corporation. This processor was originally intended to be a low consumption mobile chip.[1] The purpose of this paper is to analyze some of the characteristics of this system and compare them to an Arm Cortex-A53 Architecture which is used in raspberry pi 3's processor. It also covers instruction set design, data-path design and the memory hierarchy of Atom™and ARM processors. Furthermore this paper will go over some of the possible advantages and disadvantages of each system.

Contents

1 Introduction to Architecture

Computer architecture analyzes a hardware in an abstract level. It describe the functionality and implementation of the system. In other words Computer architecture explain the system from a high level standpoint by covering system layout. Furthermore in the organization of the system, we describes modules and unites within a computer system and how they are interact with one another.

In this document we will compare two processors and identify their differences. Those two processors are Intel®Atom™and ARM Cortex-A53. Furthermore, this paper covers some of the methods for finding detailed information about those two processors.

1.1 Intel®Atom™

The Intel®Atom™was originally developed in April 2008.[2] This processor was originally designed to be ultra-low-power IA-32 and x86-64 microprocessors available to a wide range of mobile computers.[1] Since 2008 and 8 this processor has seen many modifications and changes to be a better fit for the market. It was originally only available to system manufacturers and it was a permanent installation soldered to the motherboard. Even though the Atom™was not available to home users it was obtainable with an ITX motherboards all in one package.

Latest Atom™was released in September 2013. This CPU clocks anywhere between 1.60GHz on a 32nm dual core Centerton S1220 to 2.40GHz on 22nm octa core Avoton C2750.[3]

This system also had several short comings mainly due to bad marketing. Atom processors are great for consuming low amount of power. This leads to heat reduction and an increase in the battery life if this chip is meant to work off of a battery. The problem on the other hand is that they processor is not powerful enough to handle the demands of an average computer user.[4] Specially, when windows vista came out, it was a poorly optimized OS filled with bugs and impractical software solutions to non existing problems. Atom™simply did not have enough processing power to handle the work load leading to a slow user experience.

for the purpose of this paper we are using an Intel®Atom™CPU N270 from Diamondville series. This single core processor has a i686 architecture and was released at 2008. By default, Atom™N270 clocks at 1.60 GHz.

1.2 Arm Cortex-A53

We all remember the days which we used to purchase computer related magazines to see the coolest newest technological breakthroughs in the field of computer science. Some of those magazines had a complementary floppy disk loaded with with the electronic version of the same magazine or a sponsored game. I remember when I received a magpi issue number 40 I got something that I did not expect. That issue of magpi came with a raspberry pi zero. In course 8 years the technology has advanced so much that we went from receiving a floppy disk to a getting an actual computer with our issue of magazine. Needless to say, this computer packs more computing power than a popular laptop produced in 2006.[5][6]

Raspberry Pi computers offered a user friendly environment for people with less technical knowledge to use this computer for their projects. Furthermore, it is a great teaching tool for non electrical engineering or computer science related engineering fields. The most important feature of raspberry pi computers are the 40 GPIO pins that are available to the user. Furthermore, the newer Raspberry pi 3 also has wireless network RF module for connecting to network and Bluetooth. All of these features, make Raspberry pi a versatile computer for a small cost of $ 35. [7]

At the heart of this small computer is an ARM processor. different raspberry pis use different ARM processors. Raspberry pi 1 used an old version of ARMv6 architecture and its performance was somewhat equivalent to a smart phone.[8] Raspberry pi 2 uses an ARM Cortex-A7 and Raspberry pi 3 uses an ARM Cortex-A53. [9][10] For the purpose of his paper we use the processor on raspberry pi 3. The Arm Cortex-A53 is a 64-bit architecture developed by ARM Holdings.

2 Instruction Set Design

processors could be classified based on the size of their instruction set. This refers to simplicity of instructions rather than the amount of instructions available. These classifications are Complex instruction set computing (CISC) and Reduced instruction set computing (RISK). These instruction sets are usually decided by a committee based on the purpose they want that processor to serve.

2.1 RISC vs CISC

- **RISC:** or Reduced instruction set computing is an architecture where the instructions are simplified to a point where the entire instruction could be completed in one CPU clock cycle.
- **CISC:** or Complex instruction set computing is an architecture where instructions are not simplified. Each instruction can take multiple clock cycles in order to be completely executed.

each system has its own advantages and disadvantages. The deciding factor for which architecture to use completely depends on the manufacturer, purpose of the CPU and the market they are trying to appeal to. Some of the differences between these two architectures and their advantages and disadvantages is thoroughly explained in an article titled " RISC vs. CISC".[11]

2.1.1 Intel®Atom™

the x86 instruction set suggests that Intel processors are all designed based on the CISC architecture. Furthermore windows 95 did not even have support for RISC architecture.[12] However, this is where the confusion begins. Some sources suggest that Intel®'s Atom™architecture is much more complex than RISC and CISC alone. Since the Pentium Pro all x86 processors have been internally a reduced instruction set architecture. These systems have a CISC decoder to convert it to RISC and then it is passed to the CPU with a RISC architecture.[12][13] According to this, Atom™processors are based on RISC architecture that are wrapped inside a CISC architecture.

2.1.2 ARM Cortex-A53

This processor architecture uses a reduced instruction set as stated in the documentation for the Cortex-A53. [14] All of arm processors are 32-bit or 64-bit RISC multi-core processors. [15]

2.2 Available Addressing Mode

There are three basic types of Addressing available. These three basic addressing methods are:

- Register addressing
- Immediate addressing
- Memory addressing

These basic addressing modes are available to many different architectures including x86.

2.2.1 Intel®Atom™

There are 5 addressing modes present in x86 architecture.

- **Register**
- **Immediate:** where second operand is an immediate constant.
- **Direct memory:** where instruction or data is directly loaded from memory through a specified address.
- **Direct offset:** This addressing is similar to Direct memory just using arithmetic to modify address
- **Register indirect:** where it accesses memory by using addresses stored in registers.

2.2.2 ARM Cortex-A53

There are 3 main addressing modes available in ARM architecture:

- **An address expression**
- **A pre-indexed address:** Where the address generated is used immediately.
- **A post-indexed address:** Where the address generated later replaces the base register.

These addressing methods are available in detail online.[16]

2.3 Address Length

The address size is stored at /proc/cpuinfo in linux. This information could be obtained using:

```
# /proc/cpuinfo
cat /proc/cpuinfo
getconf WORD_BIT
getconf LONG_BIT
arch
```

2.3.1 Intel®Atom™

This syntax returns the following result on the Atom™N270:

```
cat /proc/cpuinfo
processor    : 0
vendor_id    : GenuineIntel
cpu family    : 6
model        : 28
model name    : Intel(R) Atom(TM) CPU N270   @ 1.60GHz
stepping    : 2
microcode    : 0x218
cpu MHz        : 1600.000
cache size    : 512 KB
physical id    : 0
siblings    : 2
core id        : 0
```

```
cpu cores	: 1
apicid		: 0
initial apicid	: 0
fdiv_bug	: no
f00f_bug	: no
coma_bug	: no
fpu		: yes
fpu_exception	: yes
cpuid level	: 10
wp		: yes
flags		: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat clflush
-dts acpi mmx fxsr sse sse2 ss ht tm pbe nx constant_tsc arch_perfmon pebs bts
-aperfmperf pni dtes64 monitor ds_cpl est tm2 ssse3 xtpr pdcm movbe lahf_lm dtherm
bugs		:
bogomips	: 3191.91
clflush size	: 64
cache_alignment	: 64
address sizes	: 32 bits physical, 32 bits virtual
power management:

processor	: 1
vendor_id	: GenuineIntel
cpu family	: 6
model		: 28
model name	: Intel(R) Atom(TM) CPU N270   @ 1.60GHz
stepping	: 2
microcode	: 0x218
cpu MHz		: 1333.000
cache size	: 512 KB
physical id	: 0
siblings	: 2
core id		: 0
cpu cores	: 1
apicid		: 1
initial apicid	: 1
fdiv_bug	: no
f00f_bug	: no
coma_bug	: no
fpu		: yes
fpu_exception	: yes
cpuid level	: 10
wp		: yes
flags		: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat clflush dts
-acpi mmx fxsr sse sse2 ss ht tm pbe nx constant_tsc arch_perfmon pebs bts
-aperfmperf pni dtes64 monitor ds_cpl est tm2 ssse3 xtpr pdcm movbe lahf_lm dtherm
bugs		:
bogomips	: 3191.91
clflush size	: 64
cache_alignment	: 64
address sizes	: 32 bits physical, 32 bits virtual
getconf WORD_BIT
32
getconf LONG_BIT
32
```

```
arch
i686
```

2.3.2 ARM Cortex-A53

This syntax returned the following result on ARM Cortex-A53:

```
cat /proc/cpuinfo
processor       : 0
model name      : ARMv7 Processor rev 4 (v7l)
BogoMIPS        : 76.80
Features        : half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpae
-evtstrm crc32
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x0
CPU part        : 0xd03
CPU revision    : 4

processor       : 1
model name      : ARMv7 Processor rev 4 (v7l)
BogoMIPS        : 76.80
Features        : half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpae
-evtstrm crc32
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x0
CPU part        : 0xd03
CPU revision    : 4

processor       : 2
model name      : ARMv7 Processor rev 4 (v7l)
BogoMIPS        : 76.80
Features        : half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpae
-evtstrm crc32
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x0
CPU part        : 0xd03
CPU revision    : 4

processor       : 3
model name      : ARMv7 Processor rev 4 (v7l)
BogoMIPS        : 76.80
Features        : half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpae
-evtstrm crc32
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x0
CPU part        : 0xd03
CPU revision    : 4
```

```
Hardware        : BCM2709
Revision        : a22082
Serial          : 0000000053279c46

getconf WORD_BIT
32
getconf LONG_BIT
32
arch
ARM71
```

As it turned out the Atom™N270 has the same word size as the ARM Cortex-A53. Furthermore, we can see that the ARM processor has 2 cores and the Atom™N270 has only 1 core.

3 Data-path Design

3.1 Number of Registers

3.1.1 Atom™

x86 specification indicates that there are 6 16-bit, 8 32-bit and 16 64-bit general purpose registers available. Furthermore, there are 16-bit, 32-bit and 64-bit x87 optional registers available to FPUs.

3.1.2 Arm Cortex-A53

The Arm architecture has 31 64-bit integer register and 32 128 bit registers for floating point.[17]

3.1.3 Tricks

There are several ways to optimize the register use for compiler purposes. These methods could be explained with having a deeper understanding of the abstraction over the actual hardware and the memory usage.

- It is actually faster to use
  ```
  xor reg, reg
  ```
  than
  ```
  mov reg 0
  ```
  to clear a register. The reason to this becomes apparent when we look into the structure of Intel architecture and opcodes. The opcode for "xor" is only 2 bytes long where assigning a 32-bit value requires a 5 byte opcode.
- There is a way to compute x!=0 without branching!
  ```
  xor      eax, eax ;remember trick 1, this clears the register
  cmp      ecx, 1
  adc      eax, eax
  ```

3.2 Pipeline analysis

Pipelining is the task of parallelizing the instructions required for a process. This technique could be used to optimize a process and increase the efficiency of the processor. This efficiency could be with respect to time, temperature, power consumption or any single or group of other performance metrics.

3.2.1 Intel®Atom™

This processor has a deep pipeline. The Intel®'s Atom™pipeline is 16 stages with 13 stage penalty. [18]

IF1	IF2	IF3	ID1	ID2	ID3	SC	IS	IRF
Instruction Fetch			Instruction Decode			Instruction Dispatch		Source operand Read

AG	DC1	DC2	EX1	FT1	FT2	IWB/DC1
Data Cache Access			Execute	Exception and MT handling		Commit

Furthermore, This silicon is capable of handling high core frequencies. [18]

3.2.2 ARM Cortex-A53

ARM Cortex-A53 has an 8-stage pipeline. [14] This pipeline is more advanced than the more traditional 5-stage pipeline. The 8-stage pipeline is similar to the pipeline in iPhone 3GS processors. [19]

Fetch	Decode 1	Decode 2	Decode 3	Operand	Execute 1	Execute 2	Retire
Fetch instruction	Decode instruction			Operand	Execute instruction		Retire

Pipelines increase the efficiency of the processor by increasing the throughput of the CPU. Furthermore, it increases the Arithmetic unit's performance at the cost of losing simplicity. Atom's long pipeline allows the CPU to run at a very high frequency. ARM Cortex-A53 architecture's recommended clock speed is at 1.2 GHz. The Atom™however can run anywhere between 1.60 to 2.40 GHz clock speed.

The problem with a large pipeline is the branching. In Atom™, 16 stages of pipeline will be flushed as soon as the process branches out. This takes a toll on the performance making the processor slower in performing its task. Also the general assumption in programming is that each instruction is executed before the next one. In this case, out-of-order execution of the instructions could lead to corruption of data. Furthermore, instructions in Intel®'s Atom™has much higher latency due to all of the flip flops that are placed. I attempted to compare the performance of the two processors. The following code helps us to measure the instruction latency and throughput of an Intel®processor.[20]

```
//***************************************
// This code was found at:
//https://software.intel.com/en-us/articles/measuring-instruction-latency-and-throughput
// Article Measuring Instruction Latency and Throughput \cite{Code:1}
// Intel's developer zone
//***************************************
#include <stdio.h>
#include "LatThpt.zip"
int main( void )
{
float f;
LatThpt_Init();
LatThpt_PrepInt128();
printf( "XMM i128 Latency:" );
printf( "MOVDQA MOVDQU PSHUFD PMULLW
POR PMADDWD PUNPCKLQDQ" );
printf( "Reg<-Reg        " );
LatThpt_MeasureLatXmm( movdqa );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureLatXmm( movdqu );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureLatXmmImm( pshufd );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureLatXmm( pmullw );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureLatXmm( por );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureLatXmm( pmaddwd );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureLatXmm( punpcklqdq );
printf( "%.2f        ", LatThpt_GetClocks() );
printf( "Mem<-Reg<-Mem        " );
LatThpt_MeasureLatXmmMem( movdqa );
```

```
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureLatXmmMem( movdqu );
printf( "%.2f        ", LatThpt_GetClocks() );
printf( "xxx        " );
printf( "xxx        " );
printf( "xxx        " );
printf( "xxx        " );
printf( "xxx        " );
printf( " " );
LatThpt_PrepInt128();
printf( "XMM i128 Throughput:" );
printf( "------------------" );
printf( " MOVDQA MOVDQU PSHUFD PMULLW POR        PMADDWD PUNPCKLQDQ" );
printf( "Reg<-Reg        " );
LatThpt_MeasureThptXmm( movdqa );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureThptXmm( movdqu );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureThptXmmImm( pshufd );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureThptXmm( pmullw );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureThptXmm( por );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureThptXmm( pmaddwd );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureThptXmm( punpcklqdq );
printf( "%.2f        ", LatThpt_GetClocks() );
printf( "Reg<-Mem        " );
LatThpt_MeasureThptXmmMemLoad( movdqa );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureThptXmmMemLoad( movdqu );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureThptXmmImmMemLoad( pshufd );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureThptXmmMemLoad( pmullw );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureThptXmmMemLoad( por );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureThptXmmMemLoad( pmaddwd );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureThptXmmMemLoad( punpcklqdq );
printf( "%.2f        ", LatThpt_GetClocks() );
printf( "Mem<-Reg        " );
LatThpt_MeasureThptXmmMemStore( movdqa );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureThptXmmMemStore( movdqu );
printf( "%.2f        ", LatThpt_GetClocks() );
printf( "xxx        " );
printf( "xxx        " );
printf( "xxx        " );
printf( "xxx        " );
printf( "xxx        " );
printf( " " );
```

```
LatThpt_PrepSPFP();
printf( "XMM SPFP Latency:" );
printf( "------------------" );
printf( " MOVAPS MOVUPS SHUFPS MULPS DIVPS        MOVHLPS MOVLHPS" );
printf( "Reg<-Reg        " );
LatThpt_MeasureLatXmm( movaps );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureLatXmm( movups );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureLatXmmImm( shufps );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureLatXmm( mulps );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureLatXmm( divps );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureLatXmm( movhlps );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureLatXmm( movlhps );
printf( "%.2f        ", LatThpt_GetClocks() );
printf( "Mem<-Reg<-Mem        " );
LatThpt_MeasureLatXmmMem( movaps );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureLatXmmMem( movups );
printf( "%.2f        ", LatThpt_GetClocks() );
printf( "xxx        " );
printf( "xxx        " );
printf( "xxx        " );
printf( "xxx        " );
printf( "xxx        " );
printf( " " );
LatThpt_PrepSPFP();
printf( "XMM SPFP Throughput:" );
printf( "------------------" );
printf( "MOVAPS        MOVUPS SHUFPS MULPS DIVPS        MOVHLPS        MOVLHPS" );
printf( "Reg<-Reg        " );
LatThpt_MeasureThptXmm( movaps );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureThptXmm( movups );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureThptXmmImm( shufps );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureThptXmm( mulps );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureThptXmm( divps );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureThptXmm( movhlps );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureThptXmm( movlhps );
printf( "%.2f        ", LatThpt_GetClocks() );
printf( "Reg<-Mem        " );
LatThpt_MeasureThptXmmMemLoad( movaps );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureThptXmmMemLoad( movups );
printf( "%.2f        ", LatThpt_GetClocks() );
```

```
LatThpt_MeasureThptXmmImmMemLoad( shufps );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureThptXmmMemLoad( mulps );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureThptXmmMemLoad( divps );
printf( "%.2f        ", LatThpt_GetClocks() );
printf( "xxx        " );
printf( "xxx        " );
printf( "Mem<-Reg        " );
LatThpt_MeasureThptXmmMemStore( movaps );
printf( "%.2f        ", LatThpt_GetClocks() );
LatThpt_MeasureThptXmmMemStore( movups );
printf( "%.2f        ", LatThpt_GetClocks() );
printf( "xxx        " );
printf( "xxx        " );
printf( "xxx        " );
printf( "xxx        " );
printf( "xxx        " );
printf( " " );
LatThpt_Free();
return 0;
}

#this c code returns the following results on Pentium 4 CPU with an 11 stage pipeline
#clocked at 1.70 GHz:
XMM i128 Latency:
-----------------
MOVDQA MOVDQU PSHUFD PMULLW POR PMADDWD PUNPCK LQDQ
Reg<-Reg 6.32 6.03 4.00 6.03 2.18 6.00 2.03
Mem<-Reg<-Mem 9.71 14.03 xxx xxx xxx xxx xxx
XMM i128 Throughput:
-----------------
MOVDQA MOVDQU PSHUFD PMULLW POR PMADDWD PUNPCKLQDQ
Reg<-Reg 1.00 1.00 2.05 2.00 2.00 2.18 2.01
Reg<-Mem 1.00 2.04 2.04 2.17 2.01 2.00 2.00
Mem<-Reg 1.75 8.21 xxx xxx xxx xxx xxx
XMM SPFP Latency:
-----------------
MOVAPS MOVUPS SHUFPS MULPS DIVPS MOVHLPS MOVLHPS
Reg<-Reg 6.49 6.20 4.24 6.04 39.79 4.03 2.19
Mem<-Reg<-Mem 9.65 14.23 xxx xxx xxx xxx xxx
XMM SPFP Throughput:
-----------------
MOVAPS MOVUPS SHUFPS MULPS DIVPS MOVHLPS MOVLHPS
Reg<-Reg 1.01 1.00 2.00 2.17 39.60 2.20 2.00
Reg<-Mem 1.00 2.04 2.00 2.04 39.79 xxx xxx
Mem<-Reg 1.79 8.21 xxx xxx xxx xxx xxx
```

3.3 Use of microcode

Microcodes are low level instructions that are use in processors and microprocessors in order to split the instructions into smaller more manageable instructions. This concept sits between the hardware and the architectural level of CPU. [21]

Microcode concept and the hardcoded implementation are very similar and the advantages and disadvantages are not that noticeable from an abstract programming perspective. This is mainly the system architect team's decision. in some systems this microcode is could be updated (use of FPGAs for CU). We can detect this microcode using the following bash command:

```
dmesg | grep -i micro
```

3.3.1 Intel®'s Atom™

Atom™uses microcode. This microcode could be updated in debian systems from termian using apt-get:

```
#Intel:
sudo apt-get install intel-ucode
#AMD:
sudo apt-get install amd64-ucode
```

running the "dmesg" with "grep" returned:

```
dmesg | grep -i micro
microcode: CPU0 microcode updated early to revision 0x218, date = 2009-04-10
microcode: CPU0 sig=0x106c2, pf=0x4, revision=0x218
microcode: CPU1 sig=0x106c2, pf=0x4, revision=0x218
microcode: Microcode Update Driver: v2.01 , Peter Oruba
```

As we can see microcode is present in an Intel®'s Atom™

3.3.2 ARM Cortex-A53

This command did not return any value when ran on a Raspberry pi 3. Furthermore, referring to ARM Cortex-A53's manual it becomes clear that ARM does not use microcode. However, they use something similar. ARM licenses the hardware-description language source code to manufacturers in order to give them the ability to modify their hardware. This language describes the behavior of the micro architecture.

4 Memory Subsystem

In computer architecture memory has an hierarchy based on its speed and availability to the processor. In order for a process to run correctly it needs to have its instrcutions loaded from the main memory to ram, then to caches and finally in the registered available to the CPU.

Register
L1 Cache
L2 Cache
Random access memory
Magnetic/Optical Disk
Magnetic Tape

In this section we will look into Limits, Caches and virtual memory. Furthermore, we will look into how to see these values in Linux running on an Intel®'s Atom™processor and an ARM Cortex-A53 architecture based processor. We have already talked about registers and the number of registers available to the two systems.

4.1 Limits

CPUs can only address certain amount of address space. The concept of Limit refers to the amount of memory you can theoretically make available to certain processor. This concept is also referred to as the RAM limit.

For example, the 16-bit processors can only handle 15 MB of RAM with 1 MB reserved to OS. The 32-bit x86 architecture could have handled up to ~3.8 GB of RAM (without PAE). A 64-bit system can handle up to 16 Exabytes of RAM. To see if the system is 32 bit or 64 bit we can use the command:

```
getconf LONG_BIT
```

4.1.1 Intel®'s Atom™N270

```
getconf LONG_BIT
32
```

4.1.2 ARM Cortex-A53

```
getconf LONG_BIT
32
```

Both of these systems are 32-bit CPU making them operate with maximum of ~3.8 GB of RAM.

4.2 Caches

Referring to section 2.3 where we discussed the addressable length we used the command:

```
cat /proc/cpuinfo
```

We can use the same command to get the cache size. However, a better alternative to this is to use:

```
lscpu
```

4.2.1 Intel®'s Atom™N270

```
L1d cache:              24K
L1i cache:              32K
L2 cache:              512K
```

4.2.2 ARM Cortex-A53

The same command does not return any cache size in an ARM Cortex-A53 based architecture. However, in a Raspberry Pi 3 it has the following amount of cache available to it:

```
L1d cache:              32K
L1i cache:              32K
L2 cache:              512K
```

4.3 Virtual Memory

In computer science the operating system sets the memory space in a way to make the entire memory space available to all of the software at the same time. This happens through the concept of Virtual Memory. It is the maximum possible memory that a software could possible get in a given system. The following instruction shows the virtual memory page size available to a system:

```
$ getconf PAGESIZE
```

4.3.1 Intel®'s Atom™N270

```
$ getconf PAGESIZE
4096
```

4.3.2 ARM Cortex-A53

```
$ getconf PAGESIZE
4096
```

Both of these systems have a page size of 4096 KB.

5 Conclusion

During the course of this document we discussed some of the features of Intel®'s Atom™CPU and The ARM Cortex-A53 architecture. We compared these processors in their abilities and features and looked at their advantages and disadvantages over one another. The Intel®'s Atom™processor was a technological breakthrough in low power CPUs for consumer use. This allowed production of windows friendly net-books to be available at low cost for the consumers. However, this processor was misused when it was released. When this processor came out, Microsoft had released its brand new operating system Windows™Vista and many companies in the industry tried to have the newest windows to run on computers equipped with this CPU. Vista had many problems when it came out and it demanded a powerful processor to keep it running. Furthermore, the new OS was not efficient at all. Intel®'s Atom™simply was not powerful enough to be used for Windows®Vista. On the other hand, Raspberry Pi 3 also has a processor that is very powerful but it is not being advertised as a full computer. The operating system it runs is a light Linux compiled specifically for that ARM architecture. These cheap systems are very capable if they are used under correct circumstances and for the right purposes. The system which I used to run these codes were purchased from eBay for under $40. The Atom processor was mounted on a Dell n series Inspiron 910 mini netbook and was purchased for $30. ($6 cheaper than a the Raspberry pi 3 with ARM Cortex-A53).

References

[1] "Intel's atom n2600, n2800 and d2700: Cedar trail, the heart of the 2012 netbook," http://www.anandtech.com/show/5273/intels-atom-n2600-n2800-d2700-the-heart-of-the-2012-netbook, accessed: 2016-11-12.
[2] "Intel®atom™processors," http://www.intel.com/content/www/us/en/processors/atom/atom-processor.html, accessed: 2016-11-12.
[3] "Intel atom," https://en.wikipedia.org/wiki/Intel_Atom#History, accessed: 2016-11-12.
[4] "Forbes: Intel's not-so-mighty atom," http://www.forbes.com/forbes/2009/1005/technology-intel-atom-chips-digital-tools.html, accessed: 2016-11-12.
[5] "Lenovo thinkpad t60p review," http://www.notebookreview.com/notebookreview/lenovo-thinkpad-t60p-review-pics-specs/, accessed: 2016-11-12.
[6] "Hp pavilion dv6000t," http://www.notebookreview.com/notebooks/hp-pavilion-dv6000t/, accessed: 2016-11-12.
[7] "Raspberry pi 3 has wi-fi and bluetooth, 64-bit chip, still just $35," http://arstechnica.com/information-technology/2016/02/raspberry-pi-3-has-wi-fi-and-bluetooth-64-bit-chip-still-just-35/, accessed: 2016-11-12.
[8] "Bcm2835 - high definition 1080p embedded multimedia applications processor," https://web.archive.org/web/20120513032855/http://www.broa
accessed: 2016-11-12.
[9] "Raspberry pi 2 on sale now at $35," https://www.raspberrypi.org/blog/raspberry-pi-2-on-sale/, accessed: 2016-11-12.
[10] "Raspberry pi 3 on sale now at $35," https://www.raspberrypi.org/blog/raspberry-pi-3-on-sale/, accessed: 2016-11-12.
[11] E. Roberts, "Risc vs. cisc," accessed: 2016-11-12.
[12] "Intel x86 processors cisc or risc? or both??" http://sunnyeves.blogspot.com/2009/07/intel-x86-processors-cisc-or-risc-or.html, accessed: 2016-11-12.
[13] "Intel's atom architecture: The journey begins," http://www.anandtech.com/show/2493, accessed: 2016-11-12.
[14] "Cortex-a53 processor," https://www.arm.com/products/processors/cortex-a/cortex-a53-processor.php, accessed: 2016-11-12.
[15] "What is arm," http://whatis.techtarget.com/definition/ARM-processor, accessed: 2016-11-12.
[16] "Arm: Introduction to arm: Addressing modes," http://www.davespace.co.uk/arm/introduction-to-arm/addressing.html, accessed: 2016-11-12.
[17] "Arm: Architecture," https://www.arm.com/files/downloads/ARMv8_Architecture.pdf, accessed: 2016-11-12.
[18] "Intel's atom architecture: The journey begins," http://www.anandtech.com/show/2493/11, accessed: 2016-11-12.
[19] "Understanding the iphone 3gs," http://www.anandtech.com/show/2798/2, accessed: 2016-11-12.
[20] "title of article," URL.goes.here, accessed: 2016-11-12.
[21] J. G. Kent, Allen; Williams, *Encyclopedia of Computer Science and Technology*, vol. 28.