ESPRIT LTR 21917 (Pegasus II)

Deliverable 2.1.4

ARM Port Report

Richard Black
University of Cambridge, University of Glasgow
June 1997

1 Introduction

This document describes a port of the Nemesis operating system to ARM based platforms. The ARM processor and its various platforms are particularly interesting because of their penetration into some of the novel areas of the computing industry, and because it has a number of architectural features which make for interesting research in the development of the Nemesis operating system.

Some background of the nature of ARM systems will be described, and an introduction to the relevant parts of the architecture. Then the ARM port design will be described in the context of these architectural features and the various decisions made. This is followed by a more detailed description of the current ARM ports, and the tools.

Finally I conclude with a description of some of the issues raised by the ARM work and discuss their effects on the continuing development of Nemesis.

2 Advanced Risc Machines

The ARM processor was originally developed by Acorn Computers (UK) in 1987. Advanced Risc Machines was a spin off of Acorn Computers in 1990 and since then it has become probably the most important processor designer in Europe. This has been achieved through two unusual business strategies: concentration on low power, and partnership with fabricators rather than competition.

2.1 ARM Strengths

The concentration on low power comes about due to the ARM core's simple RISC design. A typical core is as small as 35,000 transistors (compared with 275,000 for the Intel 80386 [**Intel**]). Designs are provided to fabrication partners and ASIC customers as *macro-cells*. This enables rapid development of very many different types of processor and application. Both the small die size and no necessity for high-dissipating packaging also make for high fabrication yield and hence low price.

For example the Arm610 processor, which has roughly the same performance as an Intel 486SX and was released around the same time, comes in a TQFP package of only 20mm on each side and 1mm thick. Also unlike the 486 it is cold to the touch.

ARM currently has 19 partners from around the world who make chips based on ARM designs or macro-cells [**ARM97**]. One of the most interesting of these is Digital Semiconductor who make the StrongARM family of processors. These include a blend of Digital's high performance expertise and ARM's low power expertise to provide exceptional price/performance and performance/power ratios. The SA110 can provide 185 MIPS at 450mW for as little as twenty dollars.

2.2 ARM deployment

ARM processors can be found in portable, consumer and embedded products. The ARM was selected by Apple for the Newton PDA and is also found in many modern mobile phones (e.g. Ascom and Ericsson), due to its low power requirements and low cost. It also features in smart cards due to its small size.

In the consumer environment it is favoured because of its high availability (multiple independent fabrication sources), low price, and ease of integration of macro-cells. One example is the 3DO Interactive Multiplayer which uses an ARM macro-cell on an ASIC to control the customised multimedia engines.

The ARM is also favoured for embedded products due to its simple and efficient bus architecture, small size, and simple packaging which eases manufacture. Such products are frequently found in the communications industry.

From the point of view of Nemesis the most interesting deployment of the ARM is the recent interest in information appliances. These fall primarily into two classes: the set-top box, and the network computer. The set top box idea is to replace conventional analogue television distribution with digital video channels (e.g. using MPEG) and use the television for other information such as surfing the Web as well. The set top box therefore must become a very simple computer capable of managing data channels and running a simple browser.

The network computer idea is that the user's machine should be again primarily an information provider. Data sourcing and major computation should be performed somewhere on the network. This mechanism reduces the noise and clutter on the desktop and also puts the main machines back into the hands of central administration with corporations reducing the time spent by employees on desktop machine management thus increasing productivity.

Such computers will be involved in large amounts of presentation and manipulation of video and audio streams, will require high bandwidth communications, and must operate in a timely manner. Nemesis is an ideal operating system for such platforms.

3 ARM Architecture

The ARM is a simple RISC 32-bit architecture. It has 16 visible registers one of which is the program counter. The program counter can be used as a base register to access data positioned close to the code, and can be assigned to to cause jumps. Branch instructions are PC relative and can optionally write the return address to another (fixed) register. It has two particularly unusual features:

- Conditionals. All instructions on the ARM have a conditional field (one encoding of which means "always"). This permits small conditional sections of code and expressions to be inlined avoiding branches and pipeline stalls. Another advantage is that multiple comparisons may be folded together by making succeeding comparisons conditional on the outcome of the earlier ones. Again this reduces the number of instructions executed.

- Multiples. As well as conventional load and store instructions the ARM architecture includes load and store multiple instructions. These instructions take a base register (which may be optionally adjusted by the size of the transfer) and a bitmask of the registers to be transferred. This makes stack manipulation instructions much more efficiently encoded, and more importantly permits the code to instruct the CPU directly about burst cycles on the memory bus. Usually processors include complicated circuitry to notice when one transfer is sequential with another and merge these accesses on the memory bus for efficiency. On the ARM this is not needed since such burst transfers are explicit. This gives the ARM a huge advantage in embedded applications where it can be trivial to use it as a software DMA engine (e.g. see [**Pratt95**, **Black95**]).

3.1 Interrupts

Another interesting feature is the FIQ (fast interrupt request) mode. The ARM has a very low latency interrupt whose enable is independent from conventional interrupt requests. The vector usage in the system is arranged so that the handler's first opcode forms the vector and reducing latency still further. When the processor takes a FIQ interrupt half of the register bank switches to a saved bank which gives the FIQ handler immediate access not only to scratch registers, but to its running state without any necessity to build stack frames or the like. Again this hugely reduces latency and allows the implementation of soft- or virtual- hardware and is sometimes used for memory refresh and virtual hardware [**Hayter94**, **Black95**, **Thacker96a**].

For other exception handling modes, only two registers are banked. One of these is the link register which contains the address in user mode at which the exception took place (and is therefore volatile from one exception to another) and the other is the register conventionally used for the stack pointer, which can be reused immediately. This register banking makes for very fast dispatch of system calls and interrupts.

3.2 Floating Point

The ARM architecture defines a set of instructions for performing operations within coprocessors, and for performing data transfer between the normal register file and the coprocessors. Within the architecture there are 16 coprocessors, one of which is reserved for CPU control registers. A further coprocessor is reserved for floating point support, but in practice this is very rarely needed in the applications for which the ARM is normally used.

If a coprocessor instruction is not handled then an undefined instruction trap results, and software can be used to emulate the operation instead. Normally it is more efficient, however, for the compiler to emit calls to the emulation library directly since this avoids the extra demultiplexing stage.

3.3 Calling Convention

Because the ARM processor can be used in many different circumstances there a number (or more correctly a family) or different procedure calling register conventions.[1] The principal conventions for the registers are summarised in table 1.

[1] We ignore here the obsolescent 26-bit processor address space conventions even though Nemesis has some support for them.

Physical Register	Logical Register	Use(s)
r0-r3	a1-a4	Argument (and result) registers
r4-r8	v1-v5	Callee saves registers
r9	sb / v6	Static Base or a further callee-saves register
r10	sl / v7	Stack Limit or a further callee-saves register
r11	fp / v8	Frame pointer
r12	ip	scratch registers and for stack-frame generation
r13	sp	Stack Pointer
r14	lr	Link Register (return address)
r15	pc	Program Counter

Table 1: Calling Conventions: Register Allocations

The static base register is designed for relocatable code which must access a data segment with only relative offsets. It is intended that inter-module jumps will reload this register whilst intra-module jumps will not. This is similar to e.g. the Alpha processor's "procedure variable" register. On other systems it can be used as a further callee-saves register.

The stack limit register is used, on some systems without memory management units, to ensure that stacks do not overflow, or to support non-contiguous stacks where additional "chunks" of stack memory can be dynamically allocated (with compiler support). On other systems it can be used as a further callee-saves register.

The frame pointer register can be elided on systems where it is known that the stack will *always* be contiguous (using it as another callee-saves register). To do so, however, prevents a backtrace being generated; it may also prevent engine-driven unmarshalling stubs and range table procedural exception handlers, two areas of active research on Nemesis.

In Nemesis I have chosen to use a calling convention where `r9` is allocated to `v6`, `r11` is allocated to `fp` and `r10` is used to hold the current thread's *pervasives* pointer.[2] This is compatible with the convention chosen for Wanda [**Dixon91**] and Fawn [**Black94**], two earlier research operating systems at the University of Cambridge.

3.4 Address Translation

The ARM core always uses virtual addresses and the cache is virtually addressed. These virtual addresses are translated using a MMU before being presented to the external memory system. Even exception vectors are fetched using (supervisor) virtual addresses.

The MMU unusually uses (two level) page table walking hardware in coordination with a TLB. TLB entries can map 1024, 64 or 4 Kbytes contiguously.[3] Permissions are checked using "MMU Domains"[4] where a page table entry specifies which domain (of 16) it belongs to, and a processor control register specifies the permission currently in force for the sixteen different domains. At power up the MMU provides one-to-one translation.

The use of a virtually addressed cache fits exceptionally well with the Nemesis virtual addressing model with a single address space but multiple protection domains. Any other

[2] For the importance of the pervasives on Nemesis see [**Leslie96**].

[3] 1024 Kbyte translations do not require a second level access.

[4] Not to be confused with the Nemesis' logical concepts of various types of domains.

model would require flushing the cache on a context switch, but this performance loss is not needed on Nemesis.

This almost complete lack of physical addressing by the CPU (i.e. only for page table accesses) frees the hardware designer from almost all constraints save the provision of bootstrapping code at physical address zero. As a result the hardware design can be significantly simplified. Examples can be seen on both the ARM targets on which Nemesis is now running. See section 5.1 on booting for details.

4 The Target Hardware

The core components of Nemesis have been ported to two different ARM based computers. This hardware can now be described.

4.1 The RiscPC

The RiscPC[**Acorn94**] is an ARM based personal computer designed and built by Acorn Computers. It has an Arm610 or Arm710 CPU and up to four SIMM slots. The motherboard includes an IOMD I/O controller and timer chip (based on the earlier IOC device), a VIDC20 video controller / RAMDAC, and the usual standard I/O devices. The I/O bus is DEBI (an upgraded version of the earlier PoduleBus) which has various devices available for it including Ethernet cards and an ATM interface designed at the Computer Laboratory as part of another project [**Leslie95**]. The VIDC20 ASIC also provides multiple sound channels up to 48kHz quality.

The memory bus on this computer runs at 32MHz with one word being transferred every other cycle. An additional unusual feature is that the main system data bus is used for transferring pixel data to the RAMDAC. This permits the system to be used without any VRAM where the main program DRAM is used for storing the screen image. If VRAM is present, the RAMDAC operates using 64-bit transfers where the low 32 bits use the system data bus and the high 32 bits use an exclusive data path. This mechanism means that the CPU (and other system DMA) access must be deferred whilst pixel transfers take place, affecting system performance.

The RiscPC is of interest because its components have been both further integrated and used in other appliances. A RiscPC (in HiFi Black rather than Computer Gray) was used with a JPEG decoder card and an ATM card by the ATML Set Top Box product. The ARM 7500 package is a combination of the Arm710 processor, the IOMD and the VIDC20. This chip has been used to construct Oracle's Network Computer Inc's Network Computer. Both these platforms must be used to handle and process multiple video and audio streams in a networking environment. They are sufficiently similar to the RiscPC that development of Nemesis on the RiscPC would lead to Nemesis being portable with great ease to these two embedded systems.

4.2 The Information Terminal

The Information Terminal (IT) [**Thacker96a**, **Thacker96b**] is a reference design made available to certain manufacturers by Digital Equipment Corporation's Systems Research Center. This device is in the same genre as a Network Computer. The device has a StrongARM (SA110) processor running at 235MHz and uses Synchronous burst DRAM clocked at 57MHz.

The system included an AD1843 audio codec, a CLPD6720 dual slot PCMCIA controller (ISA), an ISA VGA chip (CLGD5425), and an I87c42 keyboard / mouse controller. The system also includes 2Mbytes of flash.

A Xilinx FPGA chip provides ancillary logic and control, and implements sufficient of an ISA bus ARM bus bridge for the aforementioned ISA devices to work correctly. This chip can be programmed at power-up from a serial PROM (in which case the processor boots from the flash) or the serial PROM socket may have an umbilical cord fitted which is connected to the parallel port on a standard PC.

In the latter case the PC is responsible for programming the Xilinx chip, and then the PC must satisfy the processor's instruction and data fetches by shifting the opcodes and data bit serial across the umbilical link. Once the software has taken the Xilinx chip out of boot mode the umbilical link can be used for I/O between the IT and the PC (e.g. console).

The use of this umbilical is described further in section 5.4.2

This hardware is marketed as the "Wyse Winterm 4000 Enhanced Network Computer" and under other names. This hardware currently has the fastest interpreted scores for the Pendragon Java Benchmark [**Pendragon96**].

Two instances of the IT have been donated to the author by DEC SRC and are being used for Nemesis development.

4.3 The Shark

The Shark [**Digital97b**] is a new StrongARM reference design from DEC's Internet Appliances Group. This reference design is freely available on the internet and contains complete details (even down to the Gerber for the PCB!) to allow anyone to regenerate the machine. This is a very low cost design using only standard PC components (there is no FPGA). Digital's only intention is to stimulate the interest in using the StrongARM processor. It is our intention to port to this machine as soon as instances are available.

5 The Nemesis Port

There are three main areas of work, these are the Nemesis Trusted Supervisor Code (NTSC), the user space support code (including virtual address management), and the booting of the systems.

5.1 The NTSC

There are three components which require consideration when porting Nemesis to a new processor. These are the ntsc interface (system calls), interrupts and timer code. These will now be described.

5.1.1 System Calls

The `swi` opcode on the ARM is used to transfer control to the supervisor at a controlled entry point. This opcode has a component which is ignored by the processor. The intended mechanism is that the system call number can be placed in this field. This scheme is *not* used in Nemesis. Instead Nemesis makes use of the fact that system calls are always called with C calling conventions, and so the calling function can place the system call number in the `ip` register.[5] This can be used to demultiplex directly in the trap handler. The advantage of the Nemesis approach is that it is faster (complete demultiplex can be done in two instructions) and on the StrongARM does not pollute the data cache (and data TLB) for the re-read of the `swi` instruction.

Eventually it is intended to use this fast demultiplex to hand coded assembler for implementing the common ntsc calls. At the moment there is a single path which saves the current state in the `DCB` and enters some machine specific C code in the ntsc to handle the demultiplex. If the trap is to return then the usual mechanism for loading a context is invoked. Whilst this leaves scope for optimisation it was simple to code and debug and allowed progress with the rest of the system.

5.1.2 Interrupts

Interrupts are handled differently on different platforms. On the IT the FIQ interrupt is used for memory refresh and audio codec DMA and should never be disabled. It runs entirely asynchronously to the Nemesis image (it steals sufficiently few cycles sufficiently regularly, with sufficiently few references that it can be regarded as effecting a slight decrease in the clock rate of the CPU). On the RiscPC the FIQ interrupt can come from conventional devices which are handled by device drivers in the usual Nemesis way.

The interrupt handling code saves the current processor context in the `DCB` and enters the machine specific ntsc code. This code interrogates the interrupt controller and checks the list of registered stubs, dispatching if appropriate. Each stub returns a boolean which indicates whether the scheduler need by called. After invoking the stub for all pending interrupts either the scheduler is called, or the process context is resumed.

5.1.3 Timers

Since Nemesis is a soft real-time operating system which provides guarantees of CPU time to applications the implementation of the timer code is crucial to the operation of the system.

Local scheduling time[6] within Nemesis is represented as a 64-bit number of nanoseconds since the machine booted. The Timer code in the NTSC is required to provide the scheduler with accurate time, and to permit the setting and clearing of an alarm timer. This code is implemented very differently on the two platforms described.

Timers on the RiscPC. On the RiscPC the timer hardware provided is with the IOMD. This consists of two 2MHz down counters which raise an interrupt when they reach zero (and automatically reload). The counters are 16-bit but can only be read or programmed 8 bits at a time. Fortunately there exists a mechanism for latching the current value to get a consistent read, and to trigger a reload. The interrupt has an explicit clear mechanism. Unfortunately these accesses take internal effect based on the 2MHz clock rather than the system bus clock at which the accesses are performed; since the 2MHz clock is invisible to the processor this leads to some substantial difficulties in timing accuracy.

The solution chosen is to program one of the timers to have a maximal roll-over period and once programmed to leave it alone and use its regular interrupts (every 32.768ms) for updating the system time. Should an alarm be required by the scheduler more urgently, then

[5] This prevents the use of `swi` for a breakpoint, but an illegal instruction suffices instead.

[6] Conversion to a clock of human or network convenience is left for applications which desire it.

the second timer is programmed to interrupt after the required amount of time. Thus even though some small inaccuracy may result from the alarm programming, the correction will be applied when the next periodic interrupt occurs avoiding systematic drift. When reading the time the main counter is latched so it can be read (this does not affect its operation) and the interrupt bit is checked to ensure that the roll-over and latch did not occur at the same time.

Thus the time on the RiscPC should be accurate to approximately 500ns with a resolution of 500ns.

Timers on the IT. There is no standard timer hardware on the IT. The only timing device which can be used is the 48kHz codec interrupt (FIQ). The FIQ counts the number of interrupts and compares the current value against an alarm value. When they are equal it posts a software interrupt.

Thus the time on the IT should be accurate with a resolution of 20833 nano seconds.

5.1.4 NTSC Summary

The generic ARM NTSC is about seven hundred lines of (commented) C code, and about another seven hundred lines of assembler. The IT port has another five hundred lines of code, and the RiscPC roughly a similar amount. The NTSC work was quite manageable compared to the bootstrapping effort (see section 5.4).

5.2 User Space Code

There are various pieces of assembler code which need to be written for user space Nemesis code. These include the system call stubs, the thread context switching and exception handling code, thread startup and so on. All of these were straightforward.

The other code needed for a port to a new platform is the interrupt initialisation code, and the architecture or machine specific components of the memory system. The former was again straightforward, the latter was being done at the time that the virtual address management was being completely redesigned.[7] As a result it was decided to implement the minimum necessary code to provide a null bottom layer for the virtual address management code – no protection is actually enforced.

5.3 Current Status

The current status is that the core of the Nemesis system is running on both the RiscPC and the IT. Device driver work will continue as time permits, since this is not a core Nemesis target architecture. As the virtual address and subsequently memory management systems are developed it is intended to take the needs of the ARM based platforms into account, and develop and test code on them as well as the other hardware.

5.4 Bootstrapping

Very different mechanisms are used for booting the two targets. The simpler RiscPC method will be described first.

[7] This is the subject of deliverable 2.3.1.

5.4.1 Booting the RiscPC

The RiscPC boots into RiscOS the native operating system. This system has NFS support and can load user application images to address 0x8000. Such images are executed in 26-bit program space user mode.

Nemesis is begun in such a mode. Fortunately RiscOS (an insecure operating system) has a documented method to allow user applications to place themselves into the supervisor mode. Once in 26-bit supervisor mode the code then uses its supervisor permissions to get into 32-bit program supervisor mode. At this point it is possible to access the hardware devices such as the serial line.

It is not possible to begin the Nemesis image however; the CPU is currently running in a virtual address space created by RiscOS, and there is no way of determining what the memory mapping is. Worse, RiscOS frequently has a fairly scrambled virtual to physical page mapping, so copying the image into contiguous physical addresses may overwrite some pages before they are copied.

The only area of memory in the system which is guaranteed not to have something precious in it at this stage is the VRAM. The loader uses the primal record in the nexus[8] load image and then copies that much data (i.e. including itself) into the VRAM (at the logical addresses that RiscOS uses for the VRAM area). The loader then jumps to the copy of itself now in the VRAM (this code is entirely position independent and operates without stack memory using registers for state). The cache and MMU are then flushed and disabled in such a way that there is exactly the right number of instructions in the pipeline so that the last instruction to be fetched before the MMU is disabled[9] is an arithmetic read modify write on the program counter which transmits the flow of control into the same code but now running in one-to-one mapping from the physical VRAM.

The code must be extra careful at this point because the RiscPC contains no hardware to avoid bus clashes if write cycles are attempted to read only devices – the assumption being that the RiscOS page tables would prevent such accesses. The loader uses now uses the echo mechanism to determine which of the RAS and CAS lines are connected on the primary SIMM bank. Once that SIMM bank is sized a (first level) page table is built in the last 16Kbytes of that SIMM, mapping the DRAM linearly from virtual address zero and giving a one to one mapping for the primary address of all other devices in the system.[10]

When this is completed the loader copies the nemesis image back to 0x8000 and enters the rest of the Nemesis initialisation code.

5.4.2 Booting the IT

As mentioned earlier, the IT is connected to a PC parallel port using an umbilical cable. Special daemon software was written for Linux by the author to enable the PC to communicate using this cable; the software provides a TCP connection interface to the console stream available once the IT is running. The daemon is started from `inetd` and initialises the IT when activated.

[8]The nexus is a data structure describing the contents of the boot image including where all the modules name-spaces and programs are located, and their dependencies. It is put into the boot image immediately after the NTSC by the `nembuild` utility.

[9]The reader may be wondering why no fault will occur for the subsequent unexecuted but fetched addresses in the pipeline; this is due to the ARM's delayed prefetch abort feature, see a data-sheet for details.

[10]The first SIMM is not at physical address zero, and the physical address space is not completely decoded.

Initially some of the data lines are used to program the Xilinx chip. The Xilinx `rbt` file is read (and checked) and down-loaded (in slave serial mode). The connections between the Xilinx chip and the umbilical were reverse engineered using a multi-meter and [**Xilinx92**]. A similar programming software programming mechanism had been previously developed by the author for the Fairisle Port Controller [**Hayter94**].

When the Xilinx chip has been programmed is releases the CPU from reset. The CPU begins instruction fetch. The Xilinx chip is at this point providing a bit serial simplex channel using the data wires of the umbilical, with a synchronisation point every 16 bits. The CPU's accesses are stalled until each opcode has been shifted in by the PC.

At this stage it is essential to get the CPU running from cache as soon as possible.[11] DRAM refresh is performed in software on the IT using the interrupts from the codec,[12] and the serial link cannot predict when such interrupts will occur. Furthermore, the SDRAMs require some configuration which is time-sensitive, and the PC daemon is at the mercy of the linux scheduler. Additionally, the codec must be reset to reset the PCMCIA controller, so this must happen before memory is being used.

The daemon begins by down-loading a three opcode pre-primary loader which turns on the cache and jump back to address zero. There was a requirement to understand the StrongARM pipeline in detail. This was worked out with great tediousness by down-loading lots of little snippets of code into the IT since this work was performed before the publication of [**Digital97a**]. If the instruction cache is enabled in an instruction at address x then $x+4$ has been prefetched before the cache was enabled and $x+8$ (and the rest of its cache line) will be prefetched with the cache on irrespective of what the instruction $x+4$ does. Note that this is true *even* if $x+4$ is a predictable branch. Since the second instruction of the three writes the CPU control register the first cache line fetch begins at address 0x0C.

The cache lines are fetched critical word first. The burst read is wrapped at the end of the 4-word cache sub-block, then the other sub-block is accessed in the same order [**Digital96**]. It was decided that it would be infeasibly difficult to develop the primary loader if the assembler source had to be written so that the instructions could be presented along the umbilical in the order in which they were coded, taking into account conditional and relative branching. Instead a binary re-writer was implemented. This software takes a section of pure opcode only code and rewrites it such that it includes an initial thread of jump instructions which passes through all the cache blocks which will be loaded on their first instruction so that they will be loaded in incrementing address order. The final such opcode returns control to the second opcode of the input primary loader. The rest of the primary loader is rewritten to fit into the seven eighthes of the cache lines, relocating all the branches to take account of the non-contiguous and relocated nature.[13]

Once the primary loader is resident it resets the codec, programmes the SDRAM timing registers and performs initial precharge and refresh, and can then enable FIQ interrupts and hence perform DRAM refresh. The primary loader then enters a tight loop where it copies the secondary loader which is being down-loaded from the PC into memory.[14] Once complete,

[11] Fortunately the StrongARM's instruction cache can be enabled independently of the MMU, unlike the data cache or mixed cache on other ARM processors.

[12] At initialisation the codec is using 512 bit frames based on its 24.576MHz master clock giving 24kHz rather than 48kHz interrupts, but its reprogramming for 256 bit frames is left to the Nemesis initialisation code proper.

[13] There are two further complexities to do with the FIQ vector address and the pipeline timing of the final transfer to the secondary loader which are beyond the scope of this document.

[14] The secondary loader is actually placed at addresses "on top" of the primary loader, but since the primary

the primary loader flushes the instruction cache, destroying itself, with exactly three `nop` instructions and a branch to zero (the start of the secondary loader) in the pipeline.

The secondary loader is conventional code written in C which uses the umbilical as its output console. It uses BOOTP and TFTP to load the Nemesis operating system image. The code is based on a freely available loader designed for Intel PCs and used on the Intel PC for loading Nemesis. Since it supported the 3c509 Ethernet card it was modified so that it could drive a plug in PCMCIA version of the same card on the IT. The author merged some code from the GNU gzip utility so that compressed images can be loaded. The secondary loader includes some code which copies the Nemesis image into addresses starting at zero and starts it using techniques similar to ones described above.

5.4.3 Booting Summary

The RiscPC bootstrapping code is four hundred lines of assembler. The IT bootstrapping mechanism runs to roughly one thousand lines of Unix tool code (the umbilical daemon and the binary re-writer), plus approximately eight thousand lines of code which runs on the IT (some of which is the public domain BOOTP /,TFTP client).

5.5 Tools

The tools used for the ARM ports are based on `gcc` version 2.7.2 cross compiling from an HP9000s700 series running HP-UX 9.05. The binary utilities used have been tracking various releases of the gnu binutils distribution, particularly 2.5, 2.5.2, 2.6.2 and 2.7 – the binary utilities bfd library is also required for the `nembuild` utility. The ARM is a new architecture for these tools and many bugs were uncovered (especially in the linker and the 64-bit arithmetic support) and reported to Cygnus.

5.6 Lessons Learned

A number of things have been discovered by the process of running Nemesis on ARM based machines.

5.6.1 Timer Representation

It was discovered that the code to manipulate the time as a 64-bit number of nano seconds on a 32-bit machine was quite complicated and time consuming. For example the single C expression:

```
now = st->now + ((*st->ticks_addr) - st->base) * NS_PER_TICK;
```

which merely involves 64-bit multiplication assembles to no less than 54 instructions.

This has led to consideration of restructuring time within Nemesis so that the NTSC deals entirely in time in whatever the natural resolution of the machine is, and library code is provided for applications.

This would be a significant change and so may never actually be effected. Further evaluation will be performed when profiling is available later in the project.

loader is never resident in the IT outside of the instruction cache this is inconsequential.

5.6.2 Boot Virtual Addresses

We have seen that on the ARM it can be essential to make use of the MMU well before the Nemesis system is even begun in order to e.g. enable the cache, or ensure that memory is contiguous, or that the important vectors can have instructions placed in them. This has led to consideration of the concept of a boot virtual address space – i.e. a virtual address space mapping which already exists at the boot time of the operating system. Prior to this work Nemesis more or less considered that until it had enabled the MMU, that it was operating in physical addresses.

This distinction is now being included in the work on Virtual Address Management which will be reported in deliverable 2.3.1.

6 Conclusion

The ARM processor family is very important for embedded, set-top-box and network computer equipment. Nemesis has been shown to be easily portable to two very different machines within this architecture. Digital have shown interest in this work particularly with respect to the Shark reference design. Further work and evaluation of this port will continue throughout the Pegasus-II project ensuring that other aspects of the Nemesis design are not predicated on a small number of hardware designs.

7 Acknowledgements

The author would like to acknowledge the assistance of Ian Pratt and Paul Barham of the Computer Laboratory for help during the initial work on booting the IT system, and Mark Hayter of Digital SRC for donating the IT hardware and for various clarifications of its documentation.

References

[Acorn94] Acorn Computers Ltd. *Acorn Risc PC, Technical Reference Manual*, September 1994. ISBN 1 85250 144 8. (p 5)

[ARM97] ARM. *About Advanced RISC Machines.* `http://www.armltd.co.uk/CoInfo/CoBackground/`, 1997. (p 1)

[Black94] R.J. Black. *Explicit Network Scheduling.* Technical Report 361, University of Cambridge Computer Laboratory, December 1994. Ph.D. Dissertation. (p 4)

[Black95] R. Black. *DAN Framestore: Technical.* In ATM Document Collection 4 (The *Green* Book), chapter 2. University of Cambridge Computer Laboratory, October 1995. (p 3)

[Digital96] Digital Semiconductor. *SA110 Microprocessor Technical Reference Maunal*, revision b edition, June 1996. Order Number EC-QPWLB-TE. (p 10)

[**Digital97a**] Digital Semiconductor. *SA110 Microprocessor Instruction Timing*, revision a01 edition, April 1997. Order Number ESAE-002-A01. (p 10)

[**Digital97b**] Digital Internet Applicances Group. *Digital Network Appliance Reference Design*. `http://www.research.digital.com/src/iag/info/home.html`, May 1997. (p 6)

[**Dixon91**] M.J. Dixon. *System Support for Multi-Service Traffic*. Technical Report 245, University of Cambridge Computer Laboratory, September 1991. Ph.D. Dissertation. (p 4)

[**Hayter94**] M. Hayter and R. Black. *Fairisle Port Controller Design and Ideas*. In ATM Document Collection 3 (The *Blue* Book), chapter 23. University of Cambridge Computer Laboratory, March 1994. (pp 3, 10)

[**Intel**] *Intel Microprocessor Hall of Fame Technical Specifications*. `http://www.intel.com/intel/museum/25anniv/html/hof/techspecs.htm`. (p 1)

[**Leslie95**] I. Leslie. *DATM: A DEBI ATM Interface*. In ATM Document Collection 4 (The *Green* Book), chapter 4. University of Cambridge Computer Laboratory, October 1995. (p 5)

[**Leslie96**] I. M. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden. *The Design and Implementation of an Operating System to Support Distributed Multimedia Applications*. IEEE Journal on Selected Areas In Communications, 14(7):1280–1297, September 1996. Article describes state in May 1995. (p 4)

[**Pendragon96**] Pendragon. *CaffeineMark 2.01 scores: The Digital IT*. `http://www.webfayre.com/pendragon/jpr/jpr049702.html`, October 1996. (p 6)

[**Pratt95**] I. Pratt. *A key based framestore for the Desk Area Network*. In ATM Document Collection 4 (The *Green* Book), chapter 1. University of Cambridge Computer Laboratory, October 1995. (p 3)

[**Thacker96a**] C. Thacker. *IT Programming Guide*. Digital Equipment Corporation, Systems Research Center, August 1996. (pp 3, 5)

[**Thacker96b**] C. Thacker. *IT Theory of Operation*. Digital Equipment Corporation, Systems Research Center, August 1996. (p 5)

[**Xilinx92**] *The Programmable Gate Array Data Book*. Xilinx Inc, 1992. (p 10)