Using the Acorn C/C++ Development Environment to write 32-bit RISC OS software

Copyright © 2002 Castle Technology Ltd

Paul Skirrow, Octopus Systems, December 2002, version 2.00

Based on a two part article which appeared in both Archive and Acorn User, November/December 2002

1 Acorn C/C++ Development Environment

The *Acorn C/C++ Development Suite* from Castle Technology Ltd is a major update to *Acorn C/C++* that Acorn released in 1994. It includes new 32-bit versions of the C compiler, ARM assembler, linker and all of the other tools that were in the original C/C++ pack. More importantly it enables software to be ported to Castle's new 32-bit IYONIX PC ahead of its launch.

Acorn C/C++ cost £250 in 1994 and was supplied with four printed manuals totalling nearly 1400 pages and several floppy discs. Castle supply a single CD for £99 which contains the new software, the updated manuals in PDF format, and some useful public domain software including *StrongEd, Zap, Perl* and 32-bit IYONIX versions of Colin Granville's *PDF reader* and *FTPc* for transferring files to or from FTP sites. All of these are installed automatically by the supplied installer and work with RISC OS 3.10 onwards.

As if that wasn't enough Castle is now including a new version of the *ABC Compiler* which was previously available from Pineapple Software and *SID*, an ARM disassembler and analysis tool.

1. 32-bit Mode

When Acorn designed the ARM processor back in the mid-80's it was widely regarded as an innovative and elegant design. One key feature was to combine the Program Counter (PC) with the Processor Status Register (PSR) into a single 32-bit register. This had the benefit that the PSR could be saved automatically with the PC to achieve fast subroutine calls and quick interrupt response. The disadvantage was that '26-bit mode' limited the PC to a maximum range of 64MByte.

When the Risc PC was launched in 1994 it used the ARM6 processor which overcame this problem by introducing a '32-bit mode'. However, for compatibility with existing software, most of RISC OS and all of the RISC OS applications continued to use the old 26-bit mode and the new 32-bit mode was largely ignored.

The latest generation of ARM processors, including the Intel XScale used in Castle's IYONIX PC, only operate in 32-bit mode and no longer provide the old 26-bit compatibility mode.

The new *Acorn C/C++ Development Suite* enables programmers to produce 32-bit compatible software suitable for anything from a humble ARM2-based Archimedes running RISC OS 3.10 to the new XScale-based IYONIX PC running a 32-bit version of RISC OS.

Converting Programs to 32-bit

Most RISC OS applications are written in BASIC, C or assembler. How should these programs be updated to work on a 32-bit computer?

Anything written in BASIC should work without any problem since it is an interpreted high level language which isn't translated into ARM machine code. BASIC programs that include assembler code will need the assembler sections updating as described below.

The vast majority of the commercial RISC OS applications are written in C. The good news is that C programs can easily be converted to run in a 32-bit environment re-compiling them using the new C compiler and linking them with new 32-bit compatible libraries on Castle's CD. The new version will work on existing 26-bit versions of RISC OS as well as 32-bit versions of RISC OS.

Programs that use other compiled languages or libraries will also need to be re-compiled and linked against the new libraries. New 32-bit versions of the ABC BASIC compiler and UnixLib libraries are also available.

Assembler programs are more complicated as they need to be modified by hand. In particular, instructions that manipulate the Processor Status Register (PSR) in 26-bit mode (such as MOVS PC, TEQP, LDM ^ etc) will not work in 32-bit mode. If the code is only intended for ARM6 and later processors (ie RISC OS 3.50 onwards) then these instructions may simply be replaced by the new MSR and MRS instructions. If compatibility with ARM2 and ARM3 computers is required than more cunning code sequences are needed to achieve compatibility across the whole range of processors.

There is no easy way of automating the conversion process, although the ARM Club's free ARMalyser tool is helpful in identifying what needs converting. This may sound difficult but the changes are actually simpler than the changes needed to make software run on the StrongARM when it was launched in 1996 and I'll cover this in more detail in part 2 next month.

The Pace Branch

Some history is necessary to explain the background of the new C/C++ Development Suite:

RISCOS Ltd obtained a source code licence from Acorn in 1999 and took over development of RISC OS 4 and this version is what we see in modern RISC OS computers. Meanwhile Acorn (which became Element 14) and then Pace continued to develop RISC OS 4 internally for their Set Top Box products and the Bush Internet television. These two branches developed in different ways − RISCOS Ltd added enhancements for desktop users while Pace made other improvements and converted RISC OS and its various modules to be 32-bit compatible.

The C compiler, tools, libraries and modules that Castle are now distributing are from the Pace branch and not from RISCOS Ltd. This is necessary to get 32-bit compatibility but it does mean that some of the extra features that RISCOS Ltd added to the toolbox modules, for example, may not be available in Castle's 32-bit versions. Developers need to be aware of this and ensure that their code does not use any of the features that are unique to the RISCOS Ltd modules.

2. Acorn C/C++

You might be wondering how the new acorn C/C++ tools differ from the ones that RISCOS Ltd and Pace released 18 months ago. Although they are similar, the new tools have evolved considerably since that release and offer several new features as well as fixing a lot of problems, some of which affected the generation of 32-bit code.

The ARM assembler supports all of the new ARM processors including the Intel XScale. A `-cpu` assembler option allows the target processor to be specified and the assembler will warn of any instructions which are unavailable on the specified processor. This is useful as you might want to specify the ARM2 as the target processor but use MSR and MRS instructions after testing for 32-bit mode. The assembler will warn when you do this, but still allow it, thus making it easy to write code which runs on old and new processors.

Many of the other tools have been updated. The AMU make utility has been improved significantly and now provides much of the functionality of the popular Gnu make tool including string manipulation operators. It also allows makefiles to include other makefiles and this is especially useful for managing large projects (indeed this feature is used to build RISC OS itself).

The C Module Header Generator (CMHG) now uses the C pre-processor. This allows C header files to be included so that symbolic names can be used rather than hexadecimal numbers. For example:

```
#include "window.h"
#include "services.h"

swi-chunk-base-number:   Window_SWIChunkBase
service-call-handler:    Window_services Service_ModeChange
```

CMHG also sets the new '32-bit compatible' flag which is used by RISC OS 5 to identify and reject 26-bit modules.

The new C compiler generates faster code and implements many features from the new C99 standard.

C99

The original Acorn C/C++ software released in 1994 complied with the 1989 ANSI C (American National Standards Institute) standard which is usually referred to as C90 since it was adopted by ISO (International Standards Organisation) in 1990.

ISO have now produced a new standard for the C programming language which is commonly referred to as C99. It adds several new features which will be widely welcomed by C programmers, while maintaining backwards compatibility as far as possible. C99 adds some new keywords and tightens some restrictions, so there is a small chance that old programs will generate some errors or warning in which case a convenient `-C90` flag can be used to make the compiler behave in accordance to the C90 standard.

The new Acorn C/C++ compiler does not fully comply with the C99 standard and in particular it doesn't implement all of the new library functions, but it does implement most of the more useful features. The following should give a taste of a few of the improvements which are described fully in the new Acorn C/C++ manual included on the CD:

1. 64-bit integers are implemented via the `long long` type. Long long literals use the `LL` suffix and they can be printed using the new `ll` format specifier:

```
#include <stdlib.h>
#include <stdio.h>
unsigned long long int MSB_set = 1ULL <<63;
printf("MSB_set is %lld in decimal\n", MSB_set);
printf("or %llx in hexadecimal\n", MSB_set);
```

2. Statements and declarations may be interleaved, allowing declarations to be positioned to make code more readable:

```
foo();int b = 5;bar();
```

3. Loop variables may be declared in the `for` statement:

```
for (int i=0; i<10; i++)
```

Variable length arrays, varying in multiple dimensions may be passed as function parameters:

```
void fred(int w, int h, char d[h][w])
{
  for (int y=0; y<h; y++)
    for (int x=0; x<w; x++)
      d[y][x] ^= 0x80;
}
```

4. The last element of a structure may be an array of unspecified size and this is referred to as a flexible array member:

```
struct flex { int len; int data[]; };
struct flex *p=malloc(sizeof(struct flex)+10*sizeof(p->data[0]));
for (int i=0; i<10; i++)
  p->data[i] = i;
```

The object pointed to by `p` behaves as if it had been declared as:

```
struct flex{int len; int data[10]}
```

It is still the programmer's responsibility to allocate the necessary memory space correctly.

5. `snprintf` prints a formatted string to a buffer like `sprintf` but take an extra parameter which specifies the size of the buffer:

```
char buffer[20];
len = snprintf(buffer, sizeof buffer, “Result is %s”, s);
if (len >= sizeof buffer) printf(“Buffer too small\n”);
```

These are just a few examples − there are plenty of other improvements.

New Modules

New 32-bit versions of the shared C library, floating point emulator and other modules for distribution are supplied by Castle in a `!System` directory on the CD. The new shared C library is needed for any programs built to run in 32-mode (this is the compiler default) and any programs that use 64-bit integers or any of the other new C99 features or C99 library functions. If none of these are being used then there is no need to distribute the new C library with your application (as a general rule the new modules should only be distributed and installed if they are really needed).

ABC BASIC Compiler

The ABC BASIC Compiler has been updated to run in a 32-bit environment and it generates 32-bit code suitable for running RISC OS 3.10 to RISC OS 5. A comprehensive release note covers its use, but the full manual isn’t included at present as the original electronic copy of the manual has been lost.

ABC compiles BBC BASIC programs into ARM machine code which runs significantly faster than interpreted BASIC and it is often smaller too. With today’s fast computers execution speed is less important than it was, but compiling BASIC programs also provides a convenient way of packaging them up to deter casual observation of the program code.

The ABC compiler is more rigorous than BASIC and places some extra requirements on the programmer. This isn’t too much of a problem for properly written programs using good programming style and structure but may catch out the unwary. There are a few other restrictions too − EVAL to evaluate an expression is not supported, although its most common use to evaluate hexadecimal string is available using an extended form of VAL, for example:

```
PRINT VAL("&"+hex$)
```

Whole array operations are not supported, so you need to write loops to copy or initialise arrays rather than using a single assignment. In-line assembler (for ARM architecture version 3) is supported and it is assembled at run-time.

The compiler also provides some new features which are not available in interpreted BASIC. Constants may be declared to enable the compiler to generate more efficient code and to ensure that the value is not erroneously changed at run-time.

Documentation

The original Acorn C/C++ pack included four manuals: *Acorn C/C++, Desktop Tools, Acorn Assembler and User Interface Toolbox.* The first three have been updated significantly with details of the new features, but the toolbox manual hasn’t been updated for the first release.

The CD also includes the *RISC OS 3 Programmer’s Reference Manual, volumes 1-5a* which describes RISC OS 3.60, but not later versions.

All of the manuals are provided in Adobe PDF (Portable Document Format) and may be read using the latest version of Colin Granville’s public domain PDF reader which is included on the CD. This works extremely well especially as it has been enhanced specifically for this project. In particular, PDF version 1.01.1.08.15 renders thin lines correctly and it even has the option to highlight cross-references in blue. This is a very welcome feature as you can easily see the cross-reference links and click on them to jump to the relevant section. Even Adobe’s own Acrobat reader cannot highlight cross-references in this way (unless the author has changed the style in the document, but then it cannot be turned off for printing). Clicking with adjust causes a new PDF window to open allowing you to view several pages simultaneously.

The manuals can be printed single-sided or double-sided using !PDF but it is unlikely that many ink-jet users will print them in their entirety due to the cost. Castle may publish printed manuals in the future (at extra cost) if there is enough demand.

3. Conclusion

The Acorn C/C++ Development Suite is available now at the special launch price of £99* inc VAT from Castle. It enables software writers to produce 32-bit applications and it has already been used by several major developers to produce 32-bit versions of their software which run on the prototype IYONIX PCs.

Next month I'll explore some of the technical issues for programmers, particularly assembly language programmers.

* The full RRP is £199 inc VAT − the special £99 introductory price expires on 31st December 2002. More details and on-line ordering form are on http://www.castle.uk.co/castle/software.htm

2 Developing software for 32-bit versions of RISC OS

1. Introduction

Last month I gave an overview of the new Acorn C/C++ Development Environment CD which is now available from Castle Technology Ltd. It is a major upgrade to Acorn's original C/C++ pack with 32-bit versions of the compiler and other tools and many new features, including support for many C99 features.

The CD also includes updated *Acorn C/C++*, *Acorn Assembler*, and *Tools* manuals and the full *RISC OS 3 Programmer's Reference Manual* (volumes 1-5 and 5a) and *RISC OS 3 Style Guide* in PDF format. These manuals all include indexes and make good use of the cross-reference facilities available in PDF.

The latest versions of Zap, StrongEd, Perl, FTPc and PDF are also included − although freely available on the Internet it is useful to have these essential utilities provided on the CD, if only to raise awareness of them.

This month I'll summarise the architectural differences of the latest ARM processors, such as the Intel XScale*, and explain why and how to update existing software to run in a 32-bit environment. In particular, I'll explain the changes to expect in some RISC OS API (Application Programmer Interfaces) and discuss changes that must be made to assembler programs to make them suitable for all ARM processors.

* Strictly the term XScale® defines the microarchitecture used and not the processor. I have used the term 'Intel XScale' to mean Intel ARM-based processors, such as the the Intel 80321 I/O Processor, which implement the Intel XScale microarchitecture.

Compatibility

Since last month's article several people have contacted me expressing concern that developers must now choose whether to develop software for 32-bit environments or whether to stick with the existing 26-bit market which currently has more users. Fortunately, this is not the case.

Let's be absolutely clear − software that is produced for a 32-bit version of RISC OS will still work with a 26-bit version of RISC OS. The same version will work on the whole range of RISC OS computers from an ARM2 based Archimedes running RISC OS 3.10 to an IYONIX PC using an Intel XScale and RISC OS 5. Developers do not need to choose which machines to develop software for − they can develop for the whole range using Castle's latest 32-bit tools. Distributors will not need to stock 26-bit and 32-bit versions − updated software will work on the whole range of machines.

When we talk about converting software to 32-bit, we are not producing 32-bit only versions − we are producing software that will run in a 26-bit or 32-bit environment. We are making it 32-bit ready, not removing 26-bit compatibility. Indeed, many developers are now shipping 32-bit ready applications and you may even be using them on your current computer. There is no intrinsic benefit in using 32-bit versions though, until you start using a modern ARM chip such as the Intel XScale processor when 32-bit versions are essential.

For the assembler programmer, writing code that works on 26-bit and 32-bit processors can require a little thought, as explained below. Fortunately, C and BASIC users have it easy as Castle's tools do most of the hard work for you.

Updates

Early versions of the CD didn't include *ABC* or the *Programmer's Reference Manuals* as Castle were keen to make the 32-bit tools available to software developers as early as possible and well in advance of their new 32-bit IYONIX PC being launched. Everybody who bought an early CD should have received an update by now with updated software, ABC and the manuals included.

Future updates and release notes will appear on Castle's C/C++ support web site at:
http://www.iyonix.com/c-support.html

What is 32-bit?

What do we mean when we talk about 32-bit processors? Isn't the ARM already 32-bit? Well, yes, it is. The ARM architecture has always used a 32-bit word length, a 32-bit data bus and 32-bit registers. Indeed, the early Archimedes was promoted as being a 32-bit computer to replace Acorn's earlier computers based on the 8-bit 6502 microprocessor.

However, while the data bus has always been 32-bit, the address bus and the Program Counter (PC) on the ARM2 and ARM3 processors were only 26-bits wide, thus limiting the maximum amount of addressable memory to 64MBytes. This meant that less address pins were needed on the chip and it also gave the original ARM designers a cunning way of improving performance, as explained in '26-bit Mode' below.

The ARM6 introduced ARM architecture version 3, providing a full 32-bit address bus and 32-bit program counter, but it also provided a 26-bit compatibility mode which was used by RISC OS and by all RISC OS applications. This limited the amount of memory that could be used for executable code, and this results in the maximum application slot size of 28MBytes in RISC OS 3 and RISC OS 4.

The modern ARM processors such as the ARM9, ARM10 and Intel XScale families use ARM architecture version 4T or 5TE which remove the 26-bit backwards compatibility mode in favour of the new 16-bit 'Thumb' architecture extension. As a result RISC OS will need to run in 32-bit mode on these processors and RISC OS applications need to be 26/32-bit neutral so that they behave the same in 26-bit and 32-bit modes.

Table 1 shows a list of RISC OS computers with the ARM processor used, the architecture that that processor uses and the processor modes available.

To summarise: the ARM2 and ARM3 only provide a 26-bit mode. The ARM6, ARM7 and StrongARM provide 26-bit and 32-bit modes but RISC OS 3 and RISC OS 4 only use 26-bit mode. The ARM9, ARM10 and XScale processors only provide a 32-bit mode. All existing programs C and assembler programs will need to be updated so that they will work in both 26-bit and 32-bit modes. Most BASIC programs will unchanged.

Computer	Processor	Architecture	Modes
A300/A400	ARM2	2	26-bit only
A5000/A4	ARM3	2	26-bit only
A3000/A4000	ARM250	2	26-bit only
Risc PC	ARM610/710	3	26-bit and 32-bit
A7000	ARM7500	3	26-bit and 32-bit
A7000+	ARM7500FE	3	26-bit and 32-bit
Strong ARM Risc PC	StrongARM110	4	26-bit and 32-bit
IYONIX PC	XScale	5TE	32-bit only

Table 1 ARM processor architecture version used in RISC OS computers

32-bit OS for the Risc PC?

Some people have asked if they will be able to upgrade their Risc PC to run a 32-bit version of RISC OS. While it would be technically possible to produce a version of RISC OS for the Risc PC that used 32-bit

mode it is unlikely to happen for several reasons. Firstly, it would be an enormous undertaking and the effort is much better expended on developing new XScale based computers. Secondly it would require users to upgrade all of their software to 32-bit mode. More importantly, there would be little point. Programs running in 32-bit mode will run no faster on a Risc PC than they would in 26-bit mode. Nor is a 32-bit version of RISC OS needed to run new applications, because, as I've explained, they will work in 26-bit or 32-bit mode.

The real benefits of using 32-bit mode are that it enables us to use fast, modern processors such as the Intel XScale, and this results in an enormous performance increase. In fact no new ARM processor has been designed with 26-bit support since the Strong ARM in 1995, so we need 32-bit support if we are to use a processor that is less than 7 years old.

2. The Technical Bits

This section covers the technical details which are particularly relevant for assembly language programmers − skip to the conclusion if this isn't relevant for you.

26-bit Mode

When the ARM is working in 26-bit mode the Program Counter (PC) with the Program Status Register (PSR) are both held in register R15. This is possible because the PC only needs 24-bits leaving 8 bits free for the PSR (although the PC is notionally 26-bits wide, it always holds a word address, ie an address that is a multiple of 4, so the bottom two bits are always 0).

The original ARM designers at Acorn made use of this to fit the PSR flags alongside the PC in register R15 using the bits that the PC wasn't using for the PSR, as shown in fig. 1. The flags are used to represent negative, zero, carry, overflow conditions and interrupt disable, FIQ (Fast Interrupt reQuest) disable and processor mode bits (M0 and M1).

Fig. 1 ARM 26-bit mode combines the PC and PSR in R15

This approach wasn't just to use the register space efficiently − the big benefit is that all process status held in the PSR is preserved in R14 automatically when an exception occurs, such as an interrupt or SWI. This means that the condition flags, the interrupt status and processor mode are all saved without needing any extra instructions or registers.

For example, when calling a subroutine with the BL (Branch with Link) instruction it saves the whole of R15 (ie the PC and the PSR) in R14, the Link Register. When the subroutine exits it can use a `MOVS PC,LR` instruction to restore the PC and PSR in R15 with the saved value in R14.

32-bit Mode

When working in 32-bit mode R15 holds the 32-bit PC while the PSR is a register in its own right, called the CPSR or Current Program Status Register, as shown in fig. 2.

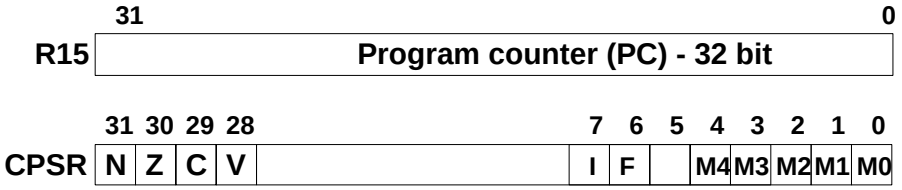

Fig. 2 ARM 32-bit mode provides one 32-bit register for the PC and one for the CPSR

Note that some of the extra bits in the CPSR are used to provide up to 32 processor modes in total, although not all of these are currently defined. Table 2 shows a full list of processor modes together with the ARM architectures that support each one.

M4-M0	Mode	Available in Architectures	Description
00000	User_26	v2, v3	User mode, 26-bit
00001	FIQ_26	v2, v3	FIQ mode, 26-bit
00010	IRQ_26	v2, v3	IRQ mode, 26-bit
00011	SVC_26	v2, v3	Supervisor mode, 26-bit
10000	User_32	v3, v4, v5	User mode 32-bit
10001	FIQ_32	v3, v4, v5	FIQ mode, 32-bit
10010	IRQ_32	v3, v4, v5	IRQ mode, 32-bit
10011	SVC_32	v3, v4, v5	Supervisor mode, 32-bit
10111	Abort_32	v3, v4, v5	Abort mode, 32-bit
11011	Undef_32	v3, v4, v5	Undefined instruction mode, 32-bit
11111	System_32	v4, v5	System mode

Table 2 Processor modes in different ARM Architectures

Later ARM architectures provide even more bits in the CPSR, such as a Thumb mode flag and DSP overflow flag. The remaining bits are reserved for future expansion. It is important that programmers take care to write code such that these bits in the PSR are never modified, otherwise their code may not run on future ARM processors.

The CPSR is no longer saved in the same register as the PC when an exception occurs. Instead, each exception mode provides a *Saved Program Status Register* or SPSR, that is used to preserve the value of the CPSR when the exception occurs. For example, when an interrupt occurs the CPSR is saved in SPSR_irq.

Fig. 3 shows the register set on the ARM3, while fig. 4 shows the register set in later architectures (but note that not all of the modes are available in all of the architectures as shown in table 2).

User mode	SVC mode	IRQ mode	FIQ mode
R0			
R1			
R2			
R3			
R4			
R5			
R6			
R7			
R8			R8_fiq
R9			R9_fiq
R10			R10_fiq
R11			R11_fiq
R12			R12_fiq
R13	R13_svc	R13_irq	R13_fiq
R14	R14_svc	R14_irq	R14_fiq
R15 (PC/PSR)			

Fig. 3 Register set in ARM Architecture 2

User and User26 mode	System	SVC and SVC26 mode	IRQ and IRQ26 mode	ABT mode	UND mode	FIQ and FIQ26 mode
R0						
R1						
R2						
R3						
R4						
R5						
R6						
R7						
R8						R8_fiq
R9						R9_fiq
R10						R10_fiq
R11						R11_fiq
R12						R12_fiq
R13		R13_svc	R13_irq	R13_abt	R13_und	R13_fiq
R14		R14_svc	R14_irq	R14_abt	R14_und	R14_fiq
R15 (PC)						

CPSR					
	SPSR_svc	SPSR_irq	SPSR_abt	SPSR_und	SPSR_fiq

Fig. 4 Registers in ARM Architecture 3, 4 and 5

Two new instructions are provided in ARM architecture version 3 to access the CPSR and SPSR:

```
MRS Rd, PSR
MSR PSR_fields,Rd
```

MSR also has an immediate form:

```
MSR PSR_fields,#immediate
```

The MSR instruction moves to the Current PSR or specified SPSR from register Rd (where Rd is R0-R14), while MRS moves to Rd from the specified immediate constant to the Current PSR or SPSR. The fields suffix can be used to specify which fields in the CPSR are being updated: c for control fields, x for extension fields, s for status fields and f for flags. When updating the mode flags for example, the c suffix can be used to indicate that no other fields should be changed.

`MRS` and `MSR` instructions are available in ARM architecture version 3 and later and hence are available on the Risc PC, A7000 and above.

These instructions can also be used to access the Saved PSR when running in an exception mode. For example, when a SWI is executed from user mode the CPSR is saved in SPSR_svc. The SWI handler running in supervisor mode can access the saved value using the MSR instruction:

```
MRS R0, SPSR_svc
```

`MRS` and `MSR` behave as `NOP` instructions in ARM architecture 2.

Writing 26-bit/32-bit Neutral Code

In order to write code that will behave the same in 26-bit and 32-bit modes certain code sequences must be avoided:

1. Avoid instructions that include the PSR in the PC when in a 26-bit mode, as they will behave differently in a 32-bit mode. For example, do not use:

```
ADD a1, a1, PC
```

Instead use:

```
ADD a1, PC, a1
```

This behaves the same in 26-bit and 32-bit modes (ignoring the PSR in both cases).

2. LR must contain the PSR flags on entry to all functions if the code is running in a 26-bit mode, so the following is illegal:

```
          ADR LR, return          ; No PSR flags!
          MOV PC, Rw
return
```

The follow is legal:

```
          MOV LR, PC
          MOV PC, R2
return
```

3. You must not use `MOVS` to return from a function in 32-bit mode:

```
MOVS PC, ...
```

This will not restore flags in the same way − in 32-bit exception mode it restores the CPSR from the SPSR for the current mode (this is unlikely to be useful though as the SPSR is subject to alteration by an interrupt routine unless IRQs have been disabled). MOVS may be used safely after ensuring that the code is running in 26-bit mode (an alternative mechanism should then be used if 32-bit mode is detected).

4. Do not attempt to restore flags in an LDM instruction in 32-bit mode:

```
          LDM {...,PC}^
```

This will not work in 32-bit mode, but may be used after testing the mode.

5. Function calls made with BL will not save the PSR. It is often easiest to change function calls to not require flag preservation. If flag preservation is required and the code is only intended for an ARM6 or later the CPSR can be saved and restored explicitly using MRS and MSR.

If ARM2/ARM3 compatibility is required then check the current processor mode and choose whether to use TEQP or MSR.

6. Be aware that SWIs are no longer required to preserve flags.

7. Avoid instructions like CMP PC, #&80000000 to set the V flag − this will not work if the program counter is above &80000000.

8. To test whether you are in a 26-bit mode;

```
          TEQ R0, R0   ; Set Z flag
          TEQ PC, PC   ; Z set if in 32-bit mode
```

This works because the second operand to TEQ includes the flags in 26-bit mode, but not in 32-bit mode, whereas the first operand never includes the flags. We are therefore comparing PC with PC+PSR if in 26-bit mode, but comparing PC with PC in 32-bit mode. This will set Z in 32-bit mode but not in 26-bit mode since we know at least one bit (the Z flag) is set in the PSR. Note that the first instruction can be omitted if a flag is known to be set (eg the V flag) or if not in user mode (since one of the mode flag bits will be set).

9. Modules and AIF files must have the new ‘32-bit compatible’ flag set in order to work on 32-bit systems.

10. Use macros to manipulate the PSR wherever possible to minimise the complexity of entry and exit sequences in subroutines.

Example

The code shown in listing 1 demonstrates how to call a SWI from within an interrupt routine. It manipulates the PSR to switch to supervisor mode, calls the SWI, then restores the original mode.

```
TEQ     PC, PC           ; Test for 32-bit mode
MRSEQ   R8, CPSR         ; 32-bit case: Use MRS to access CPSR
MOVNE   R8, PC           ; 26-bit case: Use MOV in 26-bit mode
ORR     R9, R8, #3       ; Set mode bits for SVC26 or SVC32
MSREQ   CPSR_c, R9       ; 32-bit case: set CPSR from R9
TEQNEP  R9, #0           ; 32-bit case: set PSR from R9
NOP                      ; Avoids problems on ARM2
STR     R14, [R13, #-4]! ; Save R14 (faster than STMFD
                         ;  on some CPUs)
SWI     XOS_AddCallBack  ; Call the SWI
LDR     R14, [R13], #4   ; Restore R14
TEQ     PC, PC           ; Test for 32-bit mode
MSREQ   CPSR_c, R8       ; 32-bit case: Restore the CPSR
TEQNEP  R8, #0           ; 26-bit case: Restore the CPSR
NOP                      ; Avoids problem on ARM2
```

Listing 1: Calling a SWI from an IRQ routine (all CPUs)

The complexity of this example occurs because of the need to support pre-ARM 6 processors that don't have the MRS and MSR instruction (ie RISC OS 3.1 machines). If RISC OS 3.1 support is not required, it reduces to the code shown in listing 2. This code uses MRS and MRS instructions without testing the mode as they are available on ARM6 and ARM7 processors even when running in 26-bit mode.

Beware that the StrongARM processor has a bug: conditional MSR instructions which update the c field of the CPSR flags cause the following instruction to be executed twice. The above example is safe as the following instruction is idempotent (i.e. it may be executed multiple times without affecting its behaviour).

```
MRS     R8, CPSR
ORR     R9, R8, #3       ; IRQ26->SVC26, IRQ32->SVC32
MSR     CPSR_c, R9
STR     R14, [R13, #-4]! ; Push R14
SWI     XOS_AddCallBack
LDR     R14, [R13], #4   ; Pop R14
MSR     CPSR_c, R8
```

Listing 2: Calling a SWI from an IRQ routine (ARM600 and later)

Note since this complexity only occurs in interrupt routines which are generally quite small it is quite feasible to write two interrupt routines, one for 26-bit mode and one for 32-bit mode, and simply install the relevant one at run-time.

SID and ARMalyser

The C/C++ CD contains SID, a powerful disassembler capable of generating objasm format listings with warnings to flag suspicious code sequences that could cause problems in 32-bit mode.

The ARM club have also developed a useful tool called ARMalyser which is available from their web site: http://www.armclub.org.uk/free

Memory Map

The memory map will change in RISC OS 5 to allow application slots larger than 28MBytes. As a result the RMA, screen memory, system heap, supervisor stack, 'Cursor/System/Sound' area and the ROM will all be moved to a high memory address (typically above &F0000000). This will have several consequences for C and BASIC programmers, as well as assembly language programmers. Things to watch out for:

1. Clearing bits &FC000003 bits from pointers.
2. Doing signed comparison of addresses.
3. Code which uses negative values to indicate a pointer is invalid.
4. Using high-order bits of pointers as flags.

The last two point affects some of RISC OS APIs, as explained below.

RISC OS APIs

Many RISC OS APIs (Application Program Interfaces), particularly in the WIMP, are documented as accepting a negative number or a number less than or equal to 0 to represent an invalid pointer. If a negative number is used it must be -1 now as other negative numbers will be interpreted as valid pointers (in particular, watch out for &80000000 being used). 0 may still be used where it is allowed in the current API definition.

Note that it is possible to test for 0 or -1 with the single instruction `TEQ Rn,Rn,ASR#31` which will set the Z flag if Rn contains 0 or -1.

Some APIs are defined to accept flags in the top bits of a register which also holds an address. This restricts the addressing range to 64MBytes and a new API is needed in RISC OS5.

For example, in RISC OS 3 and RISC OS 4, OS_ReadLine interprets R0 as a 26-bit address, with 6 flag bits, 4 currently unused. In RISC OS 5 OS_ReadLine now interprets R0 as a 32-bit address, and acts as though the flags are all clear. This reflects the most common usage, and allows applications not wanting to use the flags to remain unaltered.

A new SWI, OS_ReadLine32, takes its flags in R4. Bits 31 and 30 correspond to the original flags. Bits 29-8 are reserved and must be zero. Bits 7-0 are used as the echo byte (if bit 30 is set). As before, R4 is preserved by the call.

Other SWIs which will change include OS_SubstituteArgs, OS_HeapSort and OS_File as they all currently expect 26-bit pointers.

Full details of the API changes will appear soon on Castle's 32-bit web site at http://www.iyonix.com/32bit

Dynamic Areas

Since applications will now be able to claim large amounts of memory for their application slot there is no need for them to use dynamic areas, except where data needs to be shared between multiple applications.

Excessive use of dynamic areas by applications is now discouraged and it is likely that you will be able to get more memory in your application slot than you will in a dynamic area.

3. Conclusion

This article should serve as a useful introduction to the issues of writing assembly code that is 26-bit/32-bit neutral, but serious developers should refer to the C/C++ documentation and the documentation on Castle's 32-bit web site at: http://www.iyonix.com/32bit.

If writing a lot of assembler code it is recommended that you obtain the latest ARM Architecture Reference Manual (commonly referred to as the ARM ARM). It is available as a printed book from bookshops (around £32) or free of charge on CD from ARM. See http://www.iyonix.com/32bit for details.

The Acorn C/C++ Development Suite is available now at the special launch price of £99* inc VAT from Castle. It enables software writers to produce 32-bit applications before the IYONIX PC is launched and a list of '32-bit ready' applications will appear on the www.iyonix.com web site soon. Watch out for the new 'IYONIX OK' badge which will be used to advertise IYONIX PC compatible software.

Most BASIC programs will work on the IYONIX PC with little, if any, modification. Most C programs just need re-compiling, but beware of the API changes mentioned above. Assembler programs need to be updated by hand and will require the most effort. In all cases, the resulting application will be suitable for

26-bit and 32-bit versions of RISC OS.

The real benefits of using 32-bit will be seen when running familiar applications on the IYONIX PC − the speed difference is certainly impressive and makes the conversion work very worthwhile...

* The full RRP is £199 inc VAT − the special £99 introductory price expires on 31st December 2002. More details and on-line ordering form are on http://www.castle.uk.co/castle/software.htm

Intel and *XScale* are trademarks of Intel Corp.
ARM and *Thumb* are trademarks of ARM Ltd.