Application Note 37

Startup configuration of ARM Processors with MMUs

Document Number: ARM DAI 0037A
Issued: October 1996

ENGLAND
Advanced RISC Machines Limited
90 Fulbourn Road
Cherry Hinton
Cambridge CB1 4JN
UK
Telephone: +44 1223 400400
Facsimile: +44 1223 400410
Email: info@armltd.co.uk

GERMANY
Advanced RISC Machines Limited
Otto-Hahn Str. 13b
85521 Ottobrunn-Riemerling
Munich
Germany
Telephone: +49 89 608 75545
Facsimile: +49 89 608 75599
Email: info@armltd.co.uk

JAPAN
Advanced RISC Machines K.K.
KSP West Bldg, 3F 300D, 3-2-1 Sakado
Takatsu-ku, Kawasaki-shi
Kanagawa
213 Japan
Telephone: +81 44 850 1301
Facsimile: +81 44 850 1308
Email: info@armltd.co.uk

USA
ARM USA Incorporated
Suite 5
985 University Avenue
Los Gatos
CA 95030 USA
Telephone: +1 408 399 5199
Facsimile: +1 408 399 8854
Email: info@arm.com

World Wide Web address: http://www.arm.com

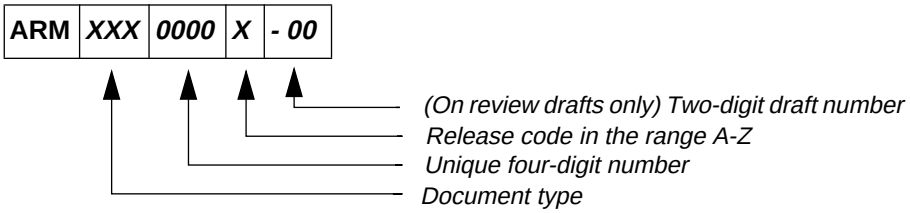
ARM
Advanced RISC Machines

Proprietary Notice

Key

Document Number

This document has a number which identifies it uniquely. The number is displayed on the front page and at the foot of each subsequent page.



Document Status

The document's status is displayed in a banner at the bottom of each page. This describes the document's confidentiality and its information status.

Confidentiality status is one of:

ARM Confidential	Distributable to ARM staff and NDA signatories only
Named Partner Confidential	Distributable to the above and to the staff of named partner companies only
Partner Confidential	Distributable within ARM and to staff of all partner companies
Open Access	No restriction on distribution

Information status is one of:

Advance	Information on a potential product
Preliminary	Current information on a product under development
Final	Complete information on a developed product

Change Log

Issue	Date	By	Change
A	October 1996	JS	First release

Table of Contents

1 Introduction

The early ARM processors (pre-ARM6) implemented a 26-bit address range and were little endian only. ARM6 and later ARM cores expanded the address range to 32-bits, and provided both little- and big-endian support.

This application note examines how the choice of address range and endianism is controlled on ARM processors containing an ARM Memory Management Unit (MMU). This includes all revisions of all ARM 6- and ARM 7-based cached processors:

- ARM600
- ARM610
- ARM700
- ARM710
- ARM710a
- ARM710ac
- ARM7100
- ARM7500
- ARM7500FE.

The selection of endianism is also relevant to:

- ARM810
- StrongARM.

2 26-bit and 32-bit configurations

The first ARM designs implemented a 26-bit address space, a combined program counter and status register, and limited processor modes for dealing with exceptions. With the ARM6 and ARM7, these changed to a full 32-bit address space, separate program counter and status registers, and additional processor modes.

To allow the running of operating systems written for earlier 26-bit ARM processors, ARM6 and ARM7 offer a 26-bit configuration to allow backwards compatibility. The recommended operation for new designs is 32-bit configuration. Furthermore, when in a 32-bit configuration, a 26-bit mode may be entered by software for partial emulation of earlier systems.

The configuration controls whether a 26 or 32-bit mode is entered upon an exception, and also whether 32-bit modes are available (they are not available in 26-bit configuration). In addition, if the core is in a 26-bit mode, but in 32-bit configuration, writing to the vector table (addresses 0x0 -> 0x1c) will cause a data abort (a vector exception). The abort allows a 32-bit OS to intercept 26-bit applications writing to the vector, to allow veneer code to make a transition from the 32-bit mode entry to the 26-bit mode handler, and back again when the handler returns.

Further details of the 26-bit and 32-bit configurations can be found in *ARM Application Note 11: Differences between the ARM6 Series and Earlier ARM Processors*.

2.1 Configuration control

The ARM6 and ARM7 cores provide the signals PROG32 and DATA32 to control whether the core will start up in 26 or 32-bit configuration. These signals are not pinned out on ARM6 or ARM7 based processors with MMUs, but are instead controlled by coprocessor 15 register 1, bits 4 and 5 respectively.

At reset on ARM processors with MMUs, the PROG32 and DATA32 bits in CP15 register 1 are set low and place the processor in 26-bit configuration. Because of this configuration setting, after reset the processor enters the reset handler executing in 26-bit supervisor mode.

Setting the PROG32 and DATA32 bits high will change to 32-bit configuration. However, it will not change the current processor mode, so the processor will continue operating in the current 26-bit mode until an explicit mode change is carried out. In addition, if an exception is generated before an explicit mode change, it will cause a 32-bit mode to be entered but will typically restore the 26-bit mode that was current at the time the exception was generated upon its return.

2.2 Changing configuration and mode

To change from a 26-bit configuration to a 32-bit configuration, and change from a 26-bit mode to a 32-bit mode, you must:

1. Use an MCR instruction to set bit 4 (PROG32) and bit 5 (DATA32) of CP15 register 1.
2. Use an MSR instruction to change the CPSR control byte to its standard value on reset in 32-bit configuration, which is:

a) I and F bits set to disable interrupts
b) Bit 5 clear
c) Mode bits equal to 10011, to select SVC32 mode.

Thus the necessary startup code to change into 32-bit configuration and 32-bit

```
MOV r14, #0x70          ; Set PROG32 and DATA32 (and late abort)
MCR p15, 0, r14, c1, c0
MOV r14, #0xD3          ; I,F bits set; not Thumb; SVC32 mode.
MSR CPSR_c, r14         ; Writes only bits 7:0 of the CPSR
```

The following points should be noted:

- The processor must be in a privileged mode to execute this code.
- The machine cannot access a memory location above 64 MBytes before this code is executed.
- The machine cannot execute code beyond 64 MBytes before this code is executed.
- Upon reset, the processor will set PROG32 and DATA32 to low, regardless of their previous state. Therefore, this code must always be run by your reset handler.
- This code should be run as soon as possible after reset - probably as the first thing done in the reset handler. This is to avoid the risk of executing instructions whose effects differ (sometimes subtly) between 26-bit and 32-bit modes, or of entering an exception vector in a 26-bit mode. (In particular, do not enable interrupts and do not execute a SWI, an undefined instruction or an instruction that may cause a prefetch or data abort before running this code.)
- On ARM6-based cached processors, this code also selects the late abort model. This is done to achieve a startup state consistent with that of ARM7-based cached processors, on which this is the only abort model. If your code is for ARM6-based processors only, and you want to use the early abort model, replace the initial MOV r14,#0x70 instruction with MOV r14,#0x30.
- This code should not be used on processors without an MMU because the MCR instruction will cause an undefined instruction exception to be taken on such processors.

2.3 Configuration on future processors

ARM810 and StrongARM are different to earlier ARMs because the 26-bit configuration is not implemented on these processors. Both are permanently in 32-bit configuration, though the 26-bit modes are implemented. However, on reset, the processor will be 32-bit mode (SVC32). Therefore on these processors, no part of the code described in ***section 2.2 Changing configuration and mode on page 5*** is needed.

Attempting to cause a configuration change on ARM810 or StrongARM by writing to bits 4 or 5 of CP15 reg 1 is simply ignored. These bits are effectively hardwired to 1, setting 32-bit code and 32-bit data configuration.

Therefore, if the code described above is run on these processors it will no do anything. Code which is intended to run both on the affected processors, and on ARM810 or StrongARM, can run the code described in ***section 2.2 Changing configuration and mode on page 5*** regardless which processor is being used.

2.4 26-bit modes on future ARM cores

You should note that the 26-bit modes also have a limited lifespan. They have already been dropped completely in the ARM7TDMI core, and they are not likely to be implemented on any ARM cores subsequent to the ARM8 and StrongARM. The use of these modes is therefore not recommended.

3 Big or little endian configuration

The ARM can treat words in memory as being stored in either little endian or big endian format. In little endian format, the least significant byte of a word is stored in bits 0-7 of an addressed word. In big endian format, the least significant byte of a word is stored in bits 24-31 of an addressed word.

3.1 Endianism control

The ARM core provides signal BIGEND to control whether the core will start up in big or little endian configuration. This signal is not pinned out on ARM processors with MMUs, but is instead controlled by coprocessor 15 register 1, bit 7.

At reset on ARM processors with MMUs the BIGEND bit in CP15 register 1 is set low and so places the processor in little endian configuration.

3.2 Configuring endianism

If a little endian system is being used, no further action is required because this is the default configuration. However, if big endian configuration is required it is necessary to use an MCR instruction to set bit 7 (BIGEND) of CP15 register 1.

Therefore, the startup code to give a 32-bit address space, described in ***section 2.1 Configuration control on page 5***, could be modified to cause a change into big endian configuration as follows:

```
MOV r14, #0xF0          ; Set BIGEND, PROG32 and DATA32
                        ; (and late abort)
MCR p15, 0, r14, c1, c0
MOV r14, #0xD3          ; I,F bits set; not Thumb; SVC32 mode.
MS  CPSR_c, r1          ; Writes only bits 7:0 of the CPSR
```

This could be simplified on processors that do not implement 26-bit configurations, as only BIGEND then needs setting:

```
MOV r14, #0x80          ; Set BIGEND
MCR p15, 0, r14, c1, c0
```

Again, the code to change the endianism of the processor should be executed as soon as possible after reset, and certainly before any subword-sized memory accesses are made.

Note *The endianism configuration has no effect on how ARM instructions are fetched from memory, and you should assemble or compile all of the application code (including the startup code) for the endianism that your system will run in once the appropriate selection is made.*

Big or little endian configuration