ARM 7TDMI

Data Sheet

Document Number:ARM DDI 0029E

Issued: August 1995

Change Log

Issue	Date	By	Change
A (Draft 0.1)	Sept 1994	EH/BJH	Created.
(Draft 0.2)	Oct 1994	EH	First pass review comments added.
B	Dec 1994	EH/AW	First formal release
C	Dec 1994	AW	Further review comments
	Mar 1995	AW	Reissued with open access status. No change to the content.
D draft1	Mar 1995	AW	Changes in line with the ARM7TDM datasheet. Further technical changes.
D	Mar 1995	AW	Review comments added.
E	Aug 1995	AP	Signals added plus minor changes.

ARM
Advanced RISC Machines

Key:

Open Access	No confidentiality

To enable document tracking, the document number has two codes:

Major release

-	Pre-release
A	First release
B	Second release
etc	etc

Draft Status

	Full and complete
draft1	First Draft
draft2	Second Draft
etc	etc
E	Embargoed (date given)

TOC

Contents

Contents

Contents

1 Introduction

This chapter introduces the ARM7TDMI architecture, and shows block, core, and functional diagrams for the ARM7TDMI.

1.1 Introduction

The ARM7TDMI is a member of the Advanced RISC Machines (ARM) family of general purpose 32-bit microprocessors, which offer high performance for very low power consumption and price.

The ARM architecture is based on Reduced Instruction Set Computer (RISC) principles, and the instruction set and related decode mechanism are much simpler than those of microprogrammed Complex Instruction Set Computers. This simplicity results in a high instruction throughput and impressive real-time interrupt response from a small and cost-effective chip.

Pipelining is employed so that all parts of the processing and memory systems can operate continuously. Typically, while one instruction is being executed, its successor is being decoded, and a third instruction is being fetched from memory.

The ARM memory interface has been designed to allow the performance potential to be realised without incurring high costs in the memory system. Speed-critical control signals are pipelined to allow system control functions to be implemented in standard low-power logic, and these control signals facilitate the exploitation of the fast local access modes offered by industry standard dynamic RAMs.

1.2 ARM7TDMI Architecture

The ARM7TDMI processor employs a unique architectural strategy known as *THUMB*, which makes it ideally suited to high-volume applications with memory restrictions, or applications where code density is an issue.

1.2.1 The THUMB Concept

The key idea behind THUMB is that of a super-reduced instruction set. Essentially, the ARM7TDMI processor has two instruction sets:

- the standard 32-bit ARM set
- a 16-bit THUMB set

The THUMB set's 16-bit instruction length allows it to approach twice the density of standard ARM code while retaining most of the ARM's performance advantage over a traditional 16-bit processor using 16-bit registers. This is possible because THUMB code operates on the same 32-bit register set as ARM code.

THUMB code is able to provide up to 65% of the code size of ARM, and 160% of the performance of an equivalent ARM processor connected to a 16-bit memory system.

1.2.2 THUMB’s Advantages

THUMB instructions operate with the standard ARM register configuration, allowing excellent interoperability between ARM and THUMB states. Each 16-bit THUMB instruction has a corresponding 32-bit ARM instruction with the same effect on the processor model.

The major advantage of a 32-bit (ARM) architecture over a 16-bit architecture is its ability to manipulate 32-bit integers with single instructions, and to address a large address space efficiently. When processing 32-bit data, a 16-bit architecture will take at least two instructions to perform the same task as a single ARM instruction.

However, not all the code in a program will process 32-bit data (for example, code that performs character string handling), and some instructions, like Branches, do not process any data at all.

If a 16-bit architecture only has 16-bit instructions, and a 32-bit architecture only has 32-bit instructions, then overall the 16-bit architecture will have better code density, and better than one half the performance of the 32-bit architecture. Clearly 32-bit performance comes at the cost of code density.

THUMB breaks this constraint by implementing a 16-bit instruction length on a 32-bit architecture, making the processing of 32-bit data efficient with a compact instruction coding. This provides far better performance than a 16-bit architecture, with better code density than a 32-bit architecture.

THUMB also has a major advantage over other 32-bit architectures with 16-bit instructions. This is the ability to switch back to full ARM code and execute at full speed. Thus critical loops for applications such as

- fast interrupts
- DSP algorithms

can be coded using the full ARM instruction set, and linked with THUMB code. The overhead of switching from THUMB code to ARM code is folded into sub-routine entry time. Various portions of a system can be optimised for speed or for code density by switching between THUMB and ARM execution as appropriate.

1.3 ARM7TDMI Block Diagram

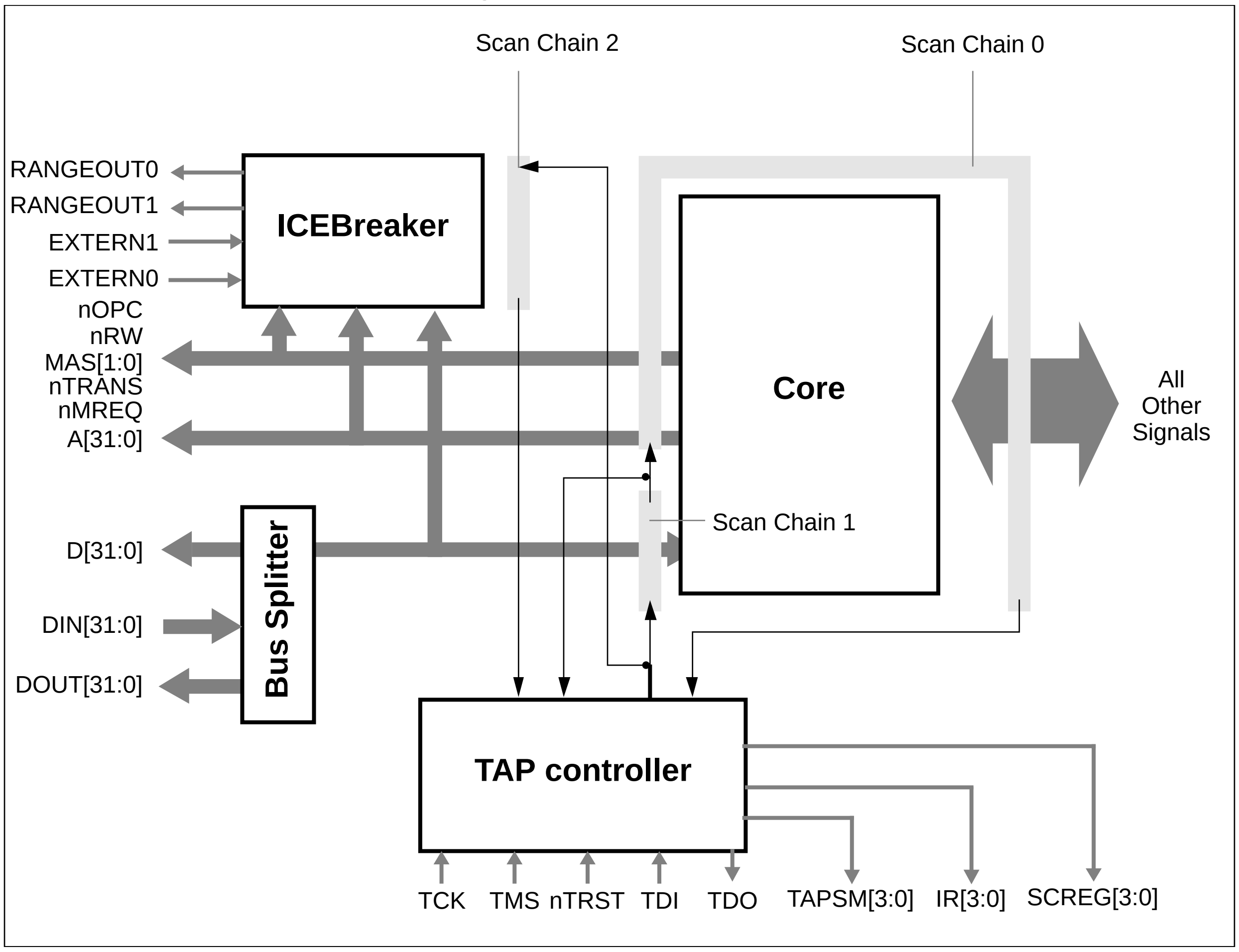


Figure 1-1: ARM7TDMI block diagram

1.4 ARM7TDMI Core Diagram

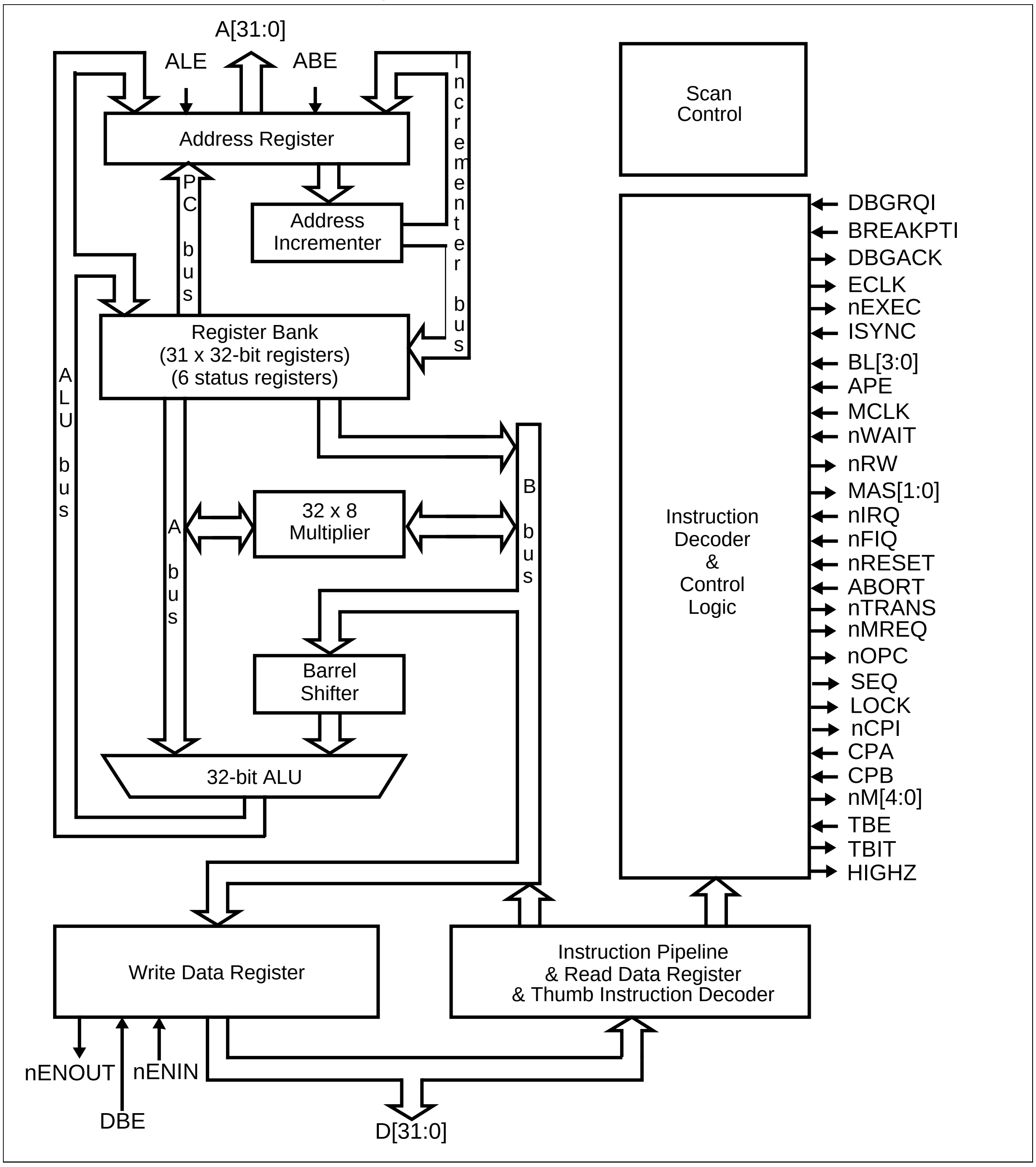


Figure 1-2: ARM7TDMI core

1.5 ARM7TDMI Functional Diagram

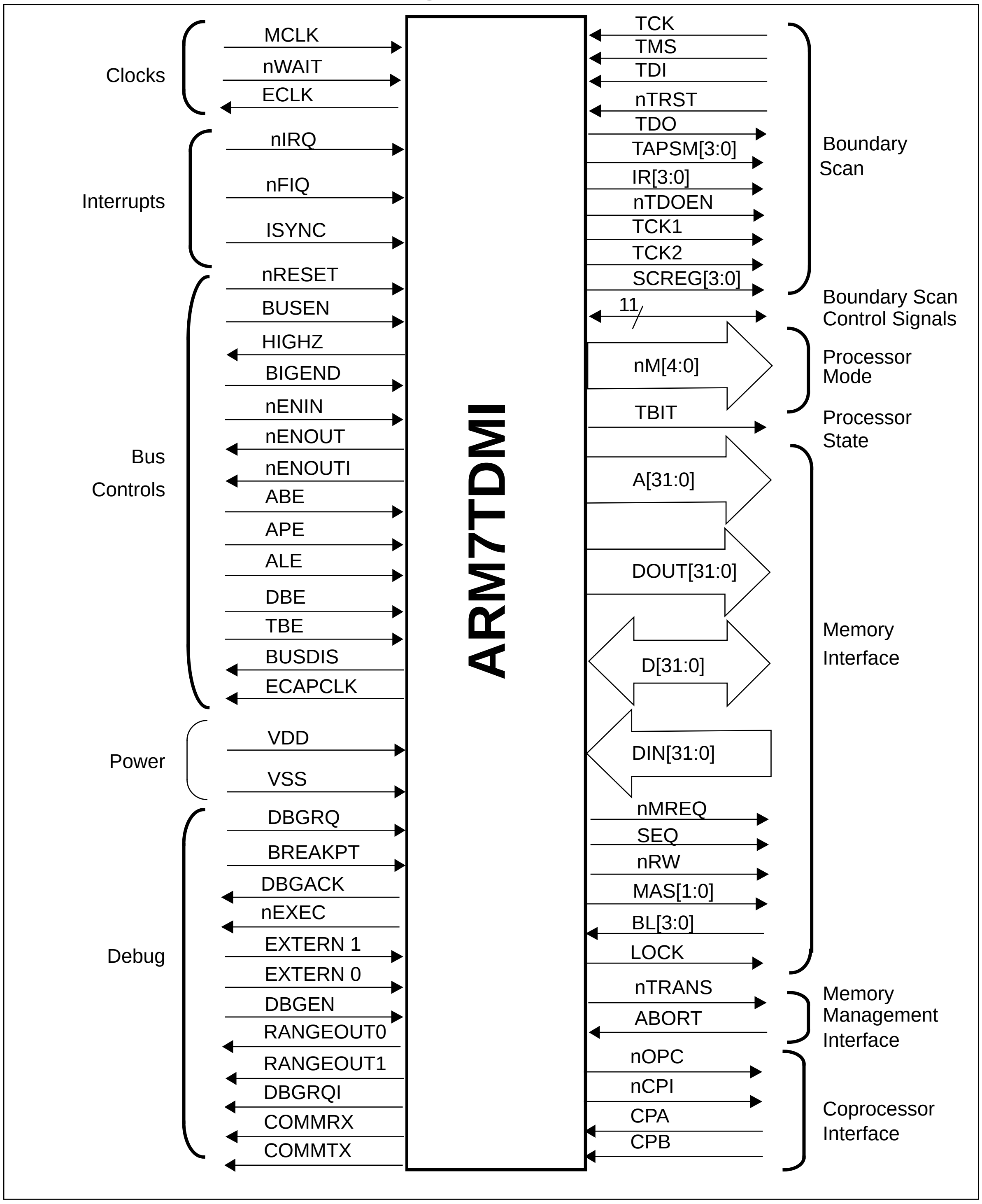


Figure 1-3: ARM7TDMI functional diagram

2 Signal Description

This chapter lists and describes the signals for the ARM7TDMI.

2.1 Signal Description

The following table lists and describes all the signals for the ARM7TDMI.

Transistor sizes

For a 0.6 µm ARM7TDMI:

INV4 driver has transistor sizes of	p = 22.32 µm/0.6 µm
	N = 12.6 µm/0.6 µm

INV8 driver has transistor sizes of	p = 44.64 µm/0.6 µm
	N = 25.2 µm/0.6 µm

Key to signal types

IC	Input CMOS thresholds
P	Power
O4	Output with INV4 driver
O8	Output with INV8 driver

Name	Type	Description
A[31:0] Addresses	08	This is the processor address bus. If **ALE** (address latch enable) is HIGH and **APE** (Address Pipeline Enable) is LOW, the addresses become valid during phase 2 of the cycle before the one to which they refer and remain so during phase 1 of the referenced cycle. Their stable period may be controlled by **ALE** or **APE** as described below.
ABE Address bus enable	IC	This is an input signal which, when LOW, puts the address bus drivers into a high impedance state. This signal has a similar effect on the following control signals: **MAS[1:0]**, **nRW**, **LOCK**, **nOPC** and **nTRANS**. **ABE** must be tied HIGH when there is no system requirement to turn off the address drivers.
ABORT Memory Abort	IC	This is an input which allows the memory system to tell the processor that a requested access is not allowed.
ALE Address latch enable.	IC	This input is used to control transparent latches on the address outputs. Normally the addresses change during phase 2 to the value required during the next cycle, but for direct interfacing to ROMs they are required to be stable to the end of phase 2. Taking **ALE** LOW until the end of phase 2 will ensure that this happens. This signal has a similar effect on the following control signals: **MAS[1:0]**, **nRW**, **LOCK**, **nOPC** and **nTRANS**. If the system does not require address lines to be held in this way, **ALE** must be tied HIGH. The address latch is static, so **ALE** may be held LOW for long periods to freeze addresses.

Table 2-1: Signal Description

Name	Type	Description
APE Address pipeline enable.	IC	When HIGH, this signal enables the address timing pipeline. In this state, the address bus plus **MAS[1:0]**, **nRW**, **nTRANS**, **LOCK** and **nOPC** change in the phase 2 prior to the memory cycle to which they refer. When **APE** is LOW, these signals change in the phase 1 of the actual cycle. Please refer to ➲ *Chapter 6, Memory Interface* for details of this timing.
BIGEND Big Endian configuration.	IC	When this signal is HIGH the processor treats bytes in memory as being in Big Endian format. When it is LOW, memory is treated as Little Endian.
BL[3:0] Byte Latch Control.	IC	These signals control when data and instructions are latched from the external data bus. When **BL[3]** is HIGH, the data on **D[31:24]** is latched on the falling edge of **MCLK**. When **BL[2**] is HIGH, the data on **D[23:16]** is latched and so on. Please refer to ➲ *Chapter 6, Memory Interface* for details on the use of these signals.
BREAKPT Breakpoint.	IC	This signal allows external hardware to halt the execution of the processor for debug purposes. When HIGH causes the current memory access to be breakpointed. If the memory access is an instruction fetch, ARM7TDMI will enter debug state if the instruction reaches the execute stage of the ARM7TDMI pipeline. If the memory access is for data, ARM7TDMI will enter debug state after the current instruction completes execution.This allows extension of the internal breakpoints provided by the ICEBreaker module. See ➲ *Chapter 9, ICEBreaker Module*.
BUSDIS Bus Disable	O	This signal is HIGH when INTEST is selected on scan chain 0 or 4 and may be used to disable external logic driving onto the bidirectional data bus during scan testing. This signal changes on the falling edge of **TCK**.
BUSEN Data bus configuration	IC	This is a static configuration signal which determines whether the bidirectional data bus, **D[31:0]**, or the unidirectional data busses, **DIN[31:0]** and **DOUT[31:0]**, are to be used for transfer of data between the processor and memory. Refer also to ➲ *Chapter 6, Memory Interface*. When **BUSEN** is LOW, the bidirectional data bus, **D[31:0]** is used. In this case, **DOUT[31:0]** is driven to value 0x00000000, and any data presented on **DIN[31:0]** is ignored. When **BUSEN** is HIGH, the bidirectional data bus, **D[31:0]** is ignored and must be left unconnected. Input data and instructions are presented on the input data bus, **DIN[31:0]**, output data appears on **DOUT[31:0]**.
COMMRX Communications Channel Receive	O	When HIGH, this signal denotes that the comms channel receive buffer is empty. This signal changes on the rising edge of **MCLK**. See ➲*9.11 Debug Communications Channel* on page 9-14 for more information on the debug comms channel.

Table 2-1: Signal Description (Continued)

Name	Type	Description
COMMTX Communications Channel Transmit	O	When HIGH, this signal denotes that the comms channel transmit buffer is empty. This signal changes on the rising edge of **MCLK**. See ➲*9.11 Debug Communications Channel* on page 9-14 for more information on the debug comms channel.
CPA Coprocessor absent.	IC	A coprocessor which is capable of performing the operation that ARM7TDMI is requesting (by asserting **nCPI**) should take **CPA** LOW immediately. If **CPA** is HIGH at the end of phase 1 of the cycle in which **nCPI** went LOW, ARM7TDMI will abort the coprocessor handshake and take the undefined instruction trap. If **CPA** is LOW and remains LOW, ARM7TDMI will busy-wait until **CPB** is LOW and then complete the coprocessor instruction.
CPB Coprocessor busy.	IC	A coprocessor which is capable of performing the operation which ARM7TDMI is requesting (by asserting **nCPI**), but cannot commit to starting it immediately, should indicate this by driving **CPB** HIGH. When the coprocessor is ready to start it should take **CPB** LOW. ARM7TDMI samples **CPB** at the end of phase 1 of each cycle in which **nCPI** is LOW.
D[31:0] Data Bus.	IC 08	These are bidirectional signal paths which are used for data transfers between the processor and external memory. During read cycles (when **nRW** is LOW), the input data must be valid before the end of phase 2 of the transfer cycle. During write cycles (when **nRW** is HIGH), the output data will become valid during phase 1 and remain valid throughout phase 2 of the transfer cycle. Note that this bus is driven at all times, irrespective of whether **BUSEN** is HIGH or LOW. When **D[31:0]** is not being used to connect to the memory system it must be left unconnected. See ➲ *Chapter 6, Memory Interface*.
DBE Data Bus Enable.	IC	This is an input signal which, when driven LOW, puts the data bus **D[31:0]** into the high impedance state. This is included for test purposes, and should be tied HIGH at all times.
DBGACK Debug acknowledge.	04	When HIGH indicates ARM is in debug state.
DBGEN Debug Enable.	IC	This input signal allows the debug features of ARM7TDMI to be disabled. This signal should be driven LOW when debugging is not required.
DBGRQ Debug request.	IC	This is a level-sensitive input, which when HIGH causes ARM7TDMI to enter debug state after executing the current instruction. This allows external hardware to force ARM7TDMI into the debug state, in addition to the debugging features provided by the ICEBreaker block. See ➲ *Chapter 9, ICEBreaker Module* for details.

Table 2-1: Signal Description (Continued)

Name	Type	Description
DBGRQI Internal debug request	04	This signal represents the debug request signal which is presented to the processor. This is the combination of external **DBGRQ**, as presented to the ARM7TDMI macrocell, and bit 1 of the debug control register. Thus there are two conditions where this signal can change. Firstly, when **DBGRQ** changes, **DBGRQI** will change after a propagation delay. When bit 1 of the debug control register has been written, this signal will change on the falling edge of **TCK** when the TAP controller state machine is in the RUN-TEST/IDLE state. See *Chapter 9, ICEBreaker Module* for details.
DIN[31:0] Data input bus	IC	This is the input data bus which may be used to transfer instructions and data between the processor and memory.This data input bus is only used when **BUSEN** is HIGH. The data on this bus is sampled by the processor at the end of phase 2 during read cycles (i.e. when **nRW** is LOW).
DOUT[31:0] Data output bus	08	This is the data out bus, used to transfer data from the processor to the memory system. Output data only appears on this bus when **BUSEN** is HIGH. At all other times, this bus is driven to value 0x00000000. When in use, data on this bus changes during phase 1 of store cycles (i.e. when **nRW** is HIGH) and remains valid throughout phase 2.
DRIVEBS Boundary scan cell enable	04	This signal is used to control the multiplexers in the scan cells of an external boundary scan chain. This signal changes in the UPDATE-IR state when scan chain 3 is selected and either the INTEST, EXTEST, CLAMP or CLAMPZ instruction is loaded. When an external boundary scan chain is not connected, this output should be left unconnected.
ECAPCLK Extest capture clock	O	This signal removes the need for the external logic in the test chip which was required to enable the internal tristate bus during scan testing. This need not be brought out as an external pin on the test chip.
ECAPCLKBS Extest capture clock for Boundary Scan	04	This is a **TCK2** wide pulse generated when the TAP controller state machine is in the CAPTURE-DR state, the current instruction is EXTEST and scan chain 3 is selected. This is used to capture the macrocell outputs during EXTEST. When an external boundary scan chain is not connected, this output should be left unconnected.
ECLK External clock output.	04	In normal operation, this is simply **MCLK** (optionally stretched with **nWAIT**) exported from the core. When the core is being debugged, this is **DCLK**. This allows external hardware to track when the ARM7DM core is clocked.
EXTERN0 External input 0.	IC	This is an input to the ICEBreaker logic in the ARM7TDMI which allows breakpoints and/or watchpoints to be dependent on an external condition.

Table 2-1: Signal Description (Continued)

Open Access

Name	Type	Description
EXTERN1 External input 1.	IC	This is an input to the ICEBreaker logic in the ARM7TDMI which allows breakpoints and/or watchpoints to be dependent on an external condition.
HIGHZ	04	This signal denotes that the HIGHZ instruction has been loaded into the TAP controller. See *Chapter 8, Debug Interface* for details.
ICAPCLKBS Intest capture clock	04	This is a **TCK2** wide pulse generated when the TAP controller state machine is in the CAPTURE-DR state, the current instruction is INTEST and scan chain 3 is selected. This is used to capture the macrocell outputs during INTEST. When an external boundary scan chain is not connected, this output should be left unconnected.
IR[3:0] TAP controller Instruction register	04	These 4 bits reflect the current instruction loaded into the TAP controller instruction register. The instruction encoding is as described in *8.8 Public Instructions* on page 8-9. These bits change on the falling edge of **TCK** when the state machine is in the UPDATE-IR state.
ISYNC Synchronous interrupts.	IC	When LOW indicates that the **nIRQ** and **nFIQ** inputs are to be synchronised by the ARM core. When HIGH disables this synchronisation for inputs that are already synchronous.
LOCK Locked operation.	08	When **LOCK** is HIGH, the processor is performing a “locked” memory access, and the memory controller must wait until **LOCK** goes LOW before allowing another device to access the memory. **LOCK** changes while **MCLK** is HIGH, and remains HIGH for the duration of the locked memory accesses. It is active only during the data swap (SWP) instruction. The timing of this signal may be modified by the use of **ALE** and **APE** in a similar way to the address, please refer to the **ALE** and **APE** descriptions. This signal may also be driven to a high impedance state by driving **ABE** LOW.
MAS[1:0] Memory Access Size.	08	These are output signals used by the processor to indicate to the external memory system when a word transfer or a half-word or byte length is required. The signals take the value 10 (binary) for words, 01 for half-words and 00 for bytes. 11 is reserved. These values are valid for both read and write cycles. The signals will normally become valid during phase 2 of the cycle before the one in which the transfer will take place. They will remain stable throughout phase 1 of the transfer cycle. The timing of the signals may be modified by the use of **ALE** and **APE** in a similar way to the address, please refer to the **ALE** and **APE** descriptions. The signals may also be driven to high impedance state by driving **ABE** LOW.

Table 2-1: Signal Description (Continued)

Name	Type	Description
MCLK Memory clock input.	IC	This clock times all ARM7TDMI memory accesses and internal operations. The clock has two distinct phases - *phase 1* in which **MCLK** is LOW and *phase 2* in which **MCLK** (and **nWAIT**) is HIGH. The clock may be stretched indefinitely in either phase to allow access to slow peripherals or memory. Alternatively, the **nWAIT** input may be used with a free running **MCLK** to achieve the same effect.
nCPI Not Coprocessor instruction.	04	When ARM7TDMI executes a coprocessor instruction, it will take this output LOW and wait for a response from the coprocessor. The action taken will depend on this response, which the coprocessor signals on the **CPA** and **CPB** inputs.
nENIN NOT enable input.	IC	This signal may be used in conjunction with **nENOUT** to control the data bus during write cycles. See ➲ *Chapter 6, Memory Interface*.
nENOUT Not enable output.	04	During a data write cycle, this signal is driven LOW during phase 1, and remains LOW for the entire cycle. This may be used to aid arbitration in shared bus applications. See ➲ *Chapter 6, Memory Interface*.
nENOUTI Not enable output.	O	During a coprocessor register transfer C-cycle from the ICEbreaker comms channel coprocessor to the ARM core, this signal goes LOW during phase 1 and stays LOW for the entire cycle. This may be used to aid arbitration in shared bus systems.
nEXEC Not executed.	04	When HIGH indicates that the instruction in the execution unit is not being executed, because for example it has failed its condition code check.
nFIQ Not fast interrupt request.	IC	This is an interrupt request to the processor which causes it to be interrupted if taken LOW when the appropriate enable in the processor is active. The signal is level-sensitive and must be held LOW until a suitable response is received from the processor. **nFIQ** may be synchronous or asynchronous, depending on the state of **ISYNC**.
nHIGHZ Not **HIGHZ**	04	This signal is generated by the TAP controller when the current instruction is HIGHZ. This is used to place the scan cells of that scan chain in the high impedance state. When a external boundary scan chain is not connected, this output should be left unconnected.
nIRQ Not interrupt request.	IC	As **nFIQ**, but with lower priority. May be taken LOW to interrupt the processor when the appropriate enable is active. **nIRQ** may be synchronous or asynchronous, depending on the state of **ISYNC**.
nM[4:0] Not processor mode.	04	These are output signals which are the inverses of the internal status bits indicating the processor operation mode.

Table 2-1: Signal Description (Continued)

Name	Type	Description
nMREQ Not memory request.	04	This signal, when LOW, indicates that the processor requires memory access during the following cycle. The signal becomes valid during phase 1, remaining valid through phase 2 of the cycle preceding that to which it refers.
nOPC Not op-code fetch.	08	When LOW this signal indicates that the processor is fetching an instruction from memory; when HIGH, data (if present) is being transferred. The signal becomes valid during phase 2 of the previous cycle, remaining valid through phase 1 of the referenced cycle. The timing of this signal may be modified by the use of **ALE** and **APE** in a similar way to the address, please refer to the **ALE** and **APE** descriptions. This signal may also be driven to a high impedance state by driving **ABE** LOW.
nRESET Not reset.	IC	This is a level sensitive input signal which is used to start the processor from a known address. A LOW level will cause the instruction being executed to terminate abnormally. When **nRESET** becomes HIGH for at least one clock cycle, the processor will re-start from address 0. **nRESET** must remain LOW (and **nWAIT** must remain HIGH) for at least two clock cycles. During the LOW period the processor will perform dummy instruction fetches with the address incrementing from the point where reset was activated. The address will overflow to zero if **nRESET** is held beyond the maximum address limit.
nRW Not read/write.	08	When HIGH this signal indicates a processor write cycle; when LOW, a read cycle. It becomes valid during phase 2 of the cycle before that to which it refers, and remains valid to the end of phase 1 of the referenced cycle. The timing of this signal may be modified by the use of **ALE** and **APE** in a similar way to the address, please refer to the **ALE** and **APE** descriptions. This signal may also be driven to a high impedance state by driving **ABE** LOW.
nTDOEN Not **TDO** Enable.	04	When LOW, this signal denotes that serial data is being driven out on the **TDO** output. **nTDOEN** would normally be used as an output enable for a **TDO** pin in a packaged part.
nTRANS Not memory translate.	08	When this signal is LOW it indicates that the processor is in user mode. It may be used to tell memory management hardware when translation of the addresses should be turned on, or as an indicator of non-user mode activity. The timing of this signal may be modified by the use of **ALE** and **APE** in a similar way to the address, please refer to the **ALE** and **APE** description. This signal may also be driven to a high impedance state by driving **ABE** LOW.
nTRST Not Test Reset.	IC	Active-low reset signal for the boundary scan logic. This pin must be pulsed or driven LOW to achieve normal device operation, in addition to the normal device reset **(nRESET)**. For more information, see ➲ *Chapter 8, Debug Interface.*

Table 2-1: Signal Description (Continued)

Name	Type	Description
nWAIT Not wait.	IC	When accessing slow peripherals, ARM7TDMI can be made to wait for an integer number of **MCLK** cycles by driving **nWAIT** LOW. Internally, **nWAIT** is ANDed with **MCLK** and must only change when **MCLK** is LOW. If **nWAIT** is not used it must be tied HIGH.
PCLKBS Boundary scan update clock	04	This is a **TCK2** wide pulse generated when the TAP controller state machine is in the UPDATE-DR state and scan chain 3 is selected. This is used by an external boundary scan chain as the update clock. When an external boundary scan chain is not connected, this output should be left unconnected.
RANGEOUT0 ICEbreaker Rangeout0	04	This signal indicates that ICEbreaker watchpoint register 0 has matched the conditions currently present on the address, data and control busses. This signal is independent of the state of the watchpoint's enable control bit. **RANGEOUT0** changes when **ECLK** is LOW.
RANGEOUT1 ICEbreaker Rangeout1	04	As **RANGEOUT0** but corresponds to ICEbreaker's watchpoint register 1.
RSTCLKBS Boundary Scan Reset Clock	O	This signal denotes that either the TAP controller state machine is in the RESET state or that **nTRST** has been asserted. This may be used to reset external boundary scan cells.
SCREG[3:0] Scan Chain Register	O	These 4 bits reflect the ID number of the scan chain currently selected by the TAP controller. These bits change on the falling edge of **TCK** when the TAP state machine is in the UPDATE-DR state.
SDINBS Boundary Scan Serial Input Data	O	This signal contains the serial data to be applied to an external scan chain and is valid around the falling edge of **TCK**.
SDOUTBS Boundary scan serial output data	IC	This control signal is provided to ease the connection of an external boundary scan chain. This is the serial data out of the boundary scan chain. It should be set up to the rising edge of **TCK**. When an external boundary scan chain is not connected, this input should be tied LOW.
SEQ Sequential address.	O4	This output signal will become HIGH when the address of the next memory cycle will be related to that of the last memory access. The new address will either be the same as the previous one or 4 greater in ARM state, or 2 greater in THUMB state. The signal becomes valid during phase 1 and remains so through phase 2 of the cycle before the cycle whose address it anticipates. It may be used, in combination with the low-order address lines, to indicate that the next cycle can use a fast memory mode (for example DRAM page mode) and/or to bypass the address translation system.

Table 2-1: Signal Description (Continued)

Name	Type	Description
SHCLKBS Boundary scan shift clock, phase 1	04	This control signal is provided to ease the connection of an external boundary scan chain. **SHCLKBS** is used to clock the master half of the external scan cells. When in the SHIFT-DR state of the state machine and scan chain 3 is selected, **SHCLKBS** follows **TCK1**. When not in the SHIFT-DR state or when scan chain 3 is not selected, this clock is LOW. When an external boundary scan chain is not connected, this output should be left unconnected.
SHCLK2BS Boundary scan shift clock, phase 2	04	This control signal is provided to ease the connection of an external boundary scan chain. **SHCLK2BS** is used to clock the master half of the external scan cells. When in the SHIFT-DR state of the state machine and scan chain 3 is selected, **SHCLK2BS** follows **TCK2**. When not in the SHIFT-DR state or when scan chain 3 is not selected, this clock is LOW. When an external boundary scan chain is not connected, this output should be left unconnected.
TAPSM[3:0] TAP controller state machine	04	This bus reflects the current state of the TAP controller state machine, as shown in *8.4.2 The JTAG state machine* on page 8-8. These bits change off the rising edge of **TCK**.
TBE Test Bus Enable.	IC	When driven LOW, **TBE** forces the data bus **D[31:0]**, the Address bus **A[31:0]**, plus **LOCK**, **MAS[1:0]**, **nRW**, **nTRANS** and **nOPC** to high impedance. This is as if both **ABE** and **DBE** had both been driven LOW. However, **TBE** does not have an associated scan cell and so allows external signals to be driven high impedance during scan testing. Under normal operating conditions, **TBE** should be held HIGH at all times.
TBIT	O4	When HIGH, this signal denotes that the processor is executing the THUMB instruction set. When LOW, the processor is executing the ARM instruction set. This signal changes in phase 2 in the first execute cycle of a BX instruction.
TCK	IC	Test Clock.
TCK1 TCK, phase 1	04	This clock represents phase 1 of **TCK**. **TCK1** is HIGH when **TCK** is HIGH, although there is a slight phase lag due to the internal clock non-overlap.
TCK2 TCK, phase 2	04	This clock represents phase 2 of **TCK**. **TCK2** is HIGH when **TCK** is LOW, although there is a slight phase lag due to the internal clock non-overlap.**TCK2** is the non-overlapping compliment of **TCK1**.
TDI	IC	Test Data Input.
TDO Test Data Output.	O4	Output from the boundary scan logic.
TMS	IC	Test Mode Select.

Table 2-1: Signal Description (Continued)

Name	Type	Description
VDD Power supply.	P	These connections provide power to the device.
VSS Ground.	P	These connections are the ground reference for all signals.

Table 2-1: Signal Description (Continued)

3 Programmer’s Model

This chapter describes the two operating states of the ARM7TDMI.

3.1 Processor Operating States

From the programmer's point of view, the ARM7TDMI can be in one of two states:

ARM state	which executes 32-bit, word-aligned ARM instructions.
THUMB state	which operates with 16-bit, halfword-aligned THUMB instructions. In this state, the PC uses bit 1 to select between alternate halfwords.

Note *Transition between these two states does not affect the processor mode or the contents of the registers.*

3.2 Switching State

Entering THUMB state

Entry into THUMB state can be achieved by executing a BX instruction with the state bit (bit 0) set in the operand register.

Transition to THUMB state will also occur automatically on return from an exception (IRQ, FIQ, UNDEF, ABORT, SWI etc.), if the exception was entered with the processor in THUMB state.

Entering ARM state

Entry into ARM state happens:

1 On execution of the BX instruction with the state bit clear in the operand register.

2 On the processor taking an exception (IRQ, FIQ, RESET, UNDEF, ABORT, SWI etc.).

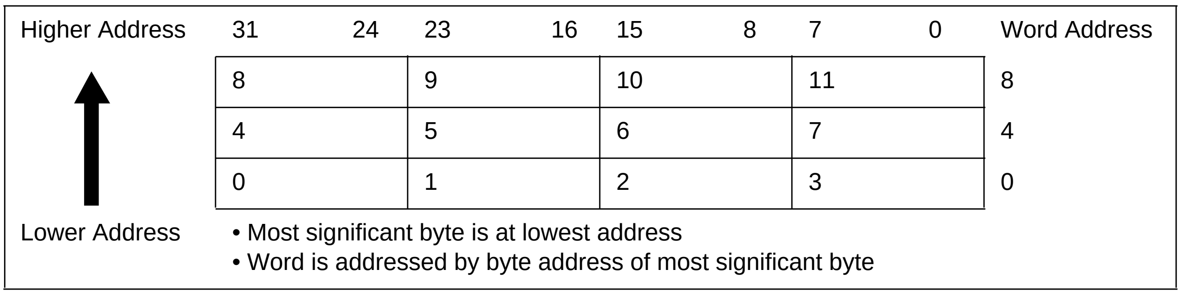
In this case, the PC is placed in the exception mode's link register, and execution commences at the exception's vector address.

3.3 Memory Formats

ARM7TDMI views memory as a linear collection of bytes numbered upwards from zero. Bytes 0 to 3 hold the first stored word, bytes 4 to 7 the second and so on. ARM7TDMI can treat words in memory as being stored either in *Big Endian* or *Little Endian* format.

3.3.1 Big endian format

In Big Endian format, the most significant byte of a word is stored at the lowest numbered byte and the least significant byte at the highest numbered byte. Byte 0 of the memory system is therefore connected to data lines 31 through 24.



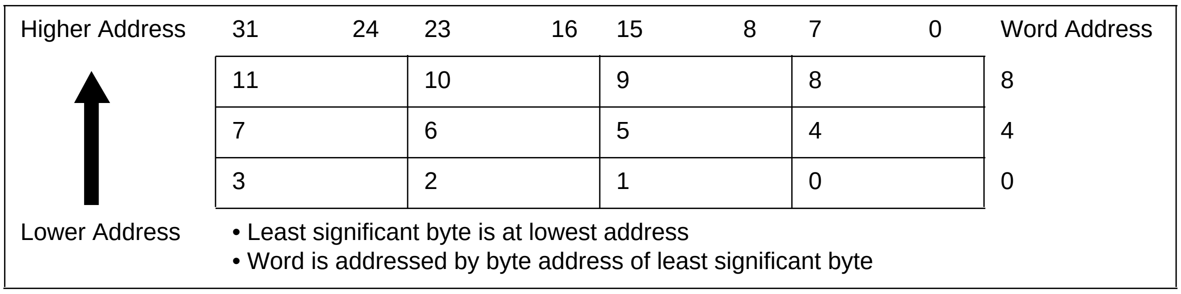
Higher Address	31 24	23 16	15 8	7 0	Word Address
	8	9	10	11	8
	4	5	6	7	4
	0	1	2	3	0
Lower Address					

• Most significant byte is at lowest address
• Word is addressed by byte address of most significant byte

Figure 3-1: Big endian addresses of bytes within words

3.3.2 Little endian format

In Little Endian format, the lowest numbered byte in a word is considered the word's least significant byte, and the highest numbered byte the most significant. Byte 0 of the memory system is therefore connected to data lines 7 through 0.



Higher Address	31 24	23 16	15 8	7 0	Word Address
	11	10	9	8	8
	7	6	5	4	4
	3	2	1	0	0
Lower Address					

• Least significant byte is at lowest address
• Word is addressed by byte address of least significant byte

Figure 3-2: Little endian addresses of bytes within words

3.4 Instruction Length

Instructions are either 32 bits long (in ARM state) or 16 bits long (in THUMB state).

3.5 Data Types

ARM7TDMI supports byte (8-bit), halfword (16-bit) and word (32-bit) data types. Words must be aligned to four-byte boundaries and half words to two-byte boundaries.

3.6 Operating Modes

ARM7TDMI supports seven modes of operation:

User (usr):	The normal ARM program execution state
FIQ (fiq):	Designed to support a data transfer or channel process
IRQ (irq):	Used for general-purpose interrupt handling
Supervisor (svc):	Protected mode for the operating system
Abort mode (abt):	Entered after a data or instruction prefetch abort
System (sys):	A privileged user mode for the operating system
Undefined (und):	Entered when an undefined instruction is executed

Mode changes may be made under software control, or may be brought about by external interrupts or exception processing. Most application programs will execute in User mode. The non-user modes - known as *privileged modes* - are entered in order to service interrupts or exceptions, or to access protected resources.

3.7 Registers

ARM7TDMI has a total of 37 registers - 31 general-purpose 32-bit registers and six status registers - but these cannot all be seen at once. The processor state and operating mode dictate which registers are available to the programmer.

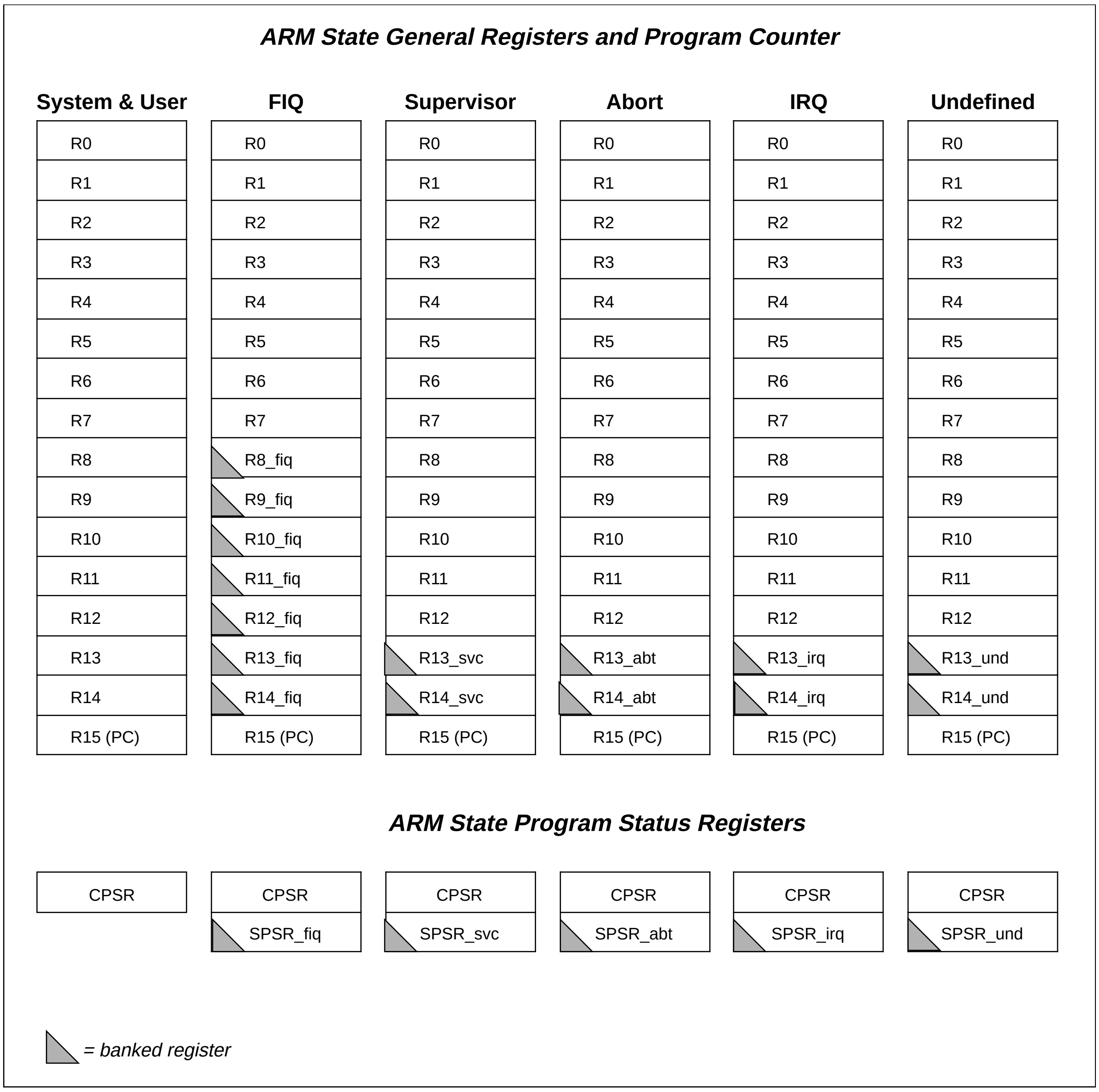
3.7.1 The ARM state register set

In ARM state, 16 general registers and one or two status registers are visible at any one time. In privileged (non-User) modes, mode-specific banked registers are switched in. *Figure 3-3: Register organization in ARM state* shows which registers are available in each mode: the banked registers are marked with a shaded triangle.

The ARM state register set contains 16 directly accessible registers: R0 to R15. All of these except R15 are general-purpose, and may be used to hold either data or address values. In addition to these, there is a seventeenth register used to store status information

Register 14	is used as the subroutine link register. This receives a copy of R15 when a Branch and Link (BL) instruction is executed. At all other times it may be treated as a general-purpose register. The corresponding banked registers R14_svc, R14_irq, R14_fiq, R14_abt and R14_und are similarly used to hold the return values of R15 when interrupts and exceptions arise, or when Branch and Link instructions are executed within interrupt or exception routines.
Register 15	holds the Program Counter (PC). In ARM state, bits [1:0] of R15 are zero and bits [31:2] contain the PC. In THUMB state, bit [0] is zero and bits [31:1] contain the PC.
Register 16	is the CPSR (Current Program Status Register). This contains condition code flags and the current mode bits.

FIQ mode has seven banked registers mapped to R8-14 (R8_fiq-R14_fiq). In ARM state, many FIQ handlers do not need to save any registers. User, IRQ, Supervisor, Abort and Undefined each have two banked registers mapped to R13 and R14, allowing each of these modes to have a private stack pointer and link registers.



ARM State General Registers and Program Counter

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	◣ R8_fiq	R8	R8	R8	R8
R9	◣ R9_fiq	R9	R9	R9	R9
R10	◣ R10_fiq	R10	R10	R10	R10
R11	◣ R11_fiq	R11	R11	R11	R11
R12	◣ R12_fiq	R12	R12	R12	R12
R13	◣ R13_fiq	◣ R13_svc	◣ R13_abt	◣ R13_irq	◣ R13_und
R14	◣ R14_fiq	◣ R14_svc	◣ R14_abt	◣ R14_irq	◣ R14_und
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)

ARM State Program Status Registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	◣ SPSR_fiq	◣ SPSR_svc	◣ SPSR_abt	◣ SPSR_irq	◣ SPSR_und

◣ *= banked register*

Figure 3-3: Register organization in ARM state

3.7.2 The THUMB state register set

The THUMB state register set is a subset of the ARM state set. The programmer has direct access to eight general registers, R0-R7, as well as the Program Counter (PC), a stack pointer register (SP), a link register (LR), and the CPSR. There are banked Stack Pointers, Link Registers and Saved Process Status Registers (SPSRs) for each privileged mode. This is shown in *Figure 3-4: Register organization in THUMB state*.

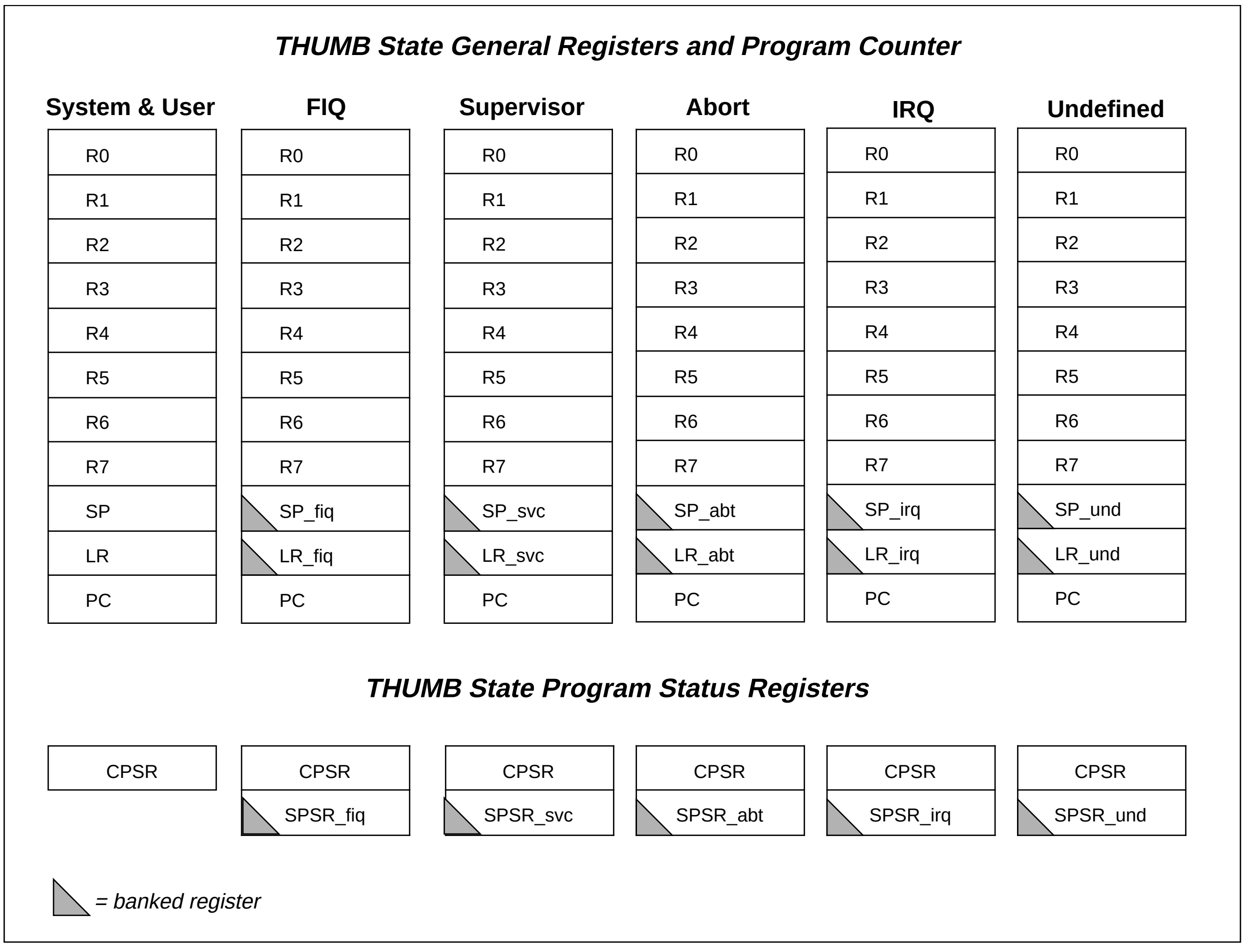


Figure 3-4: Register organization in THUMB state

3.7.3 The relationship between ARM and THUMB state registers

The THUMB state registers relate to the ARM state registers in the following way:

- THUMB state R0-R7 and ARM state R0-R7 are identical
- THUMB state CPSR and SPSRs and ARM state CPSR and SPSRs are identical
- THUMB state SP maps onto ARM state R13

- THUMB state LR maps onto ARM state R14
- The THUMB state Program Counter maps onto the ARM state Program Counter (R15)

This relationship is shown in ➲*Figure 3-5: Mapping of THUMB state registers onto ARM state registers*.

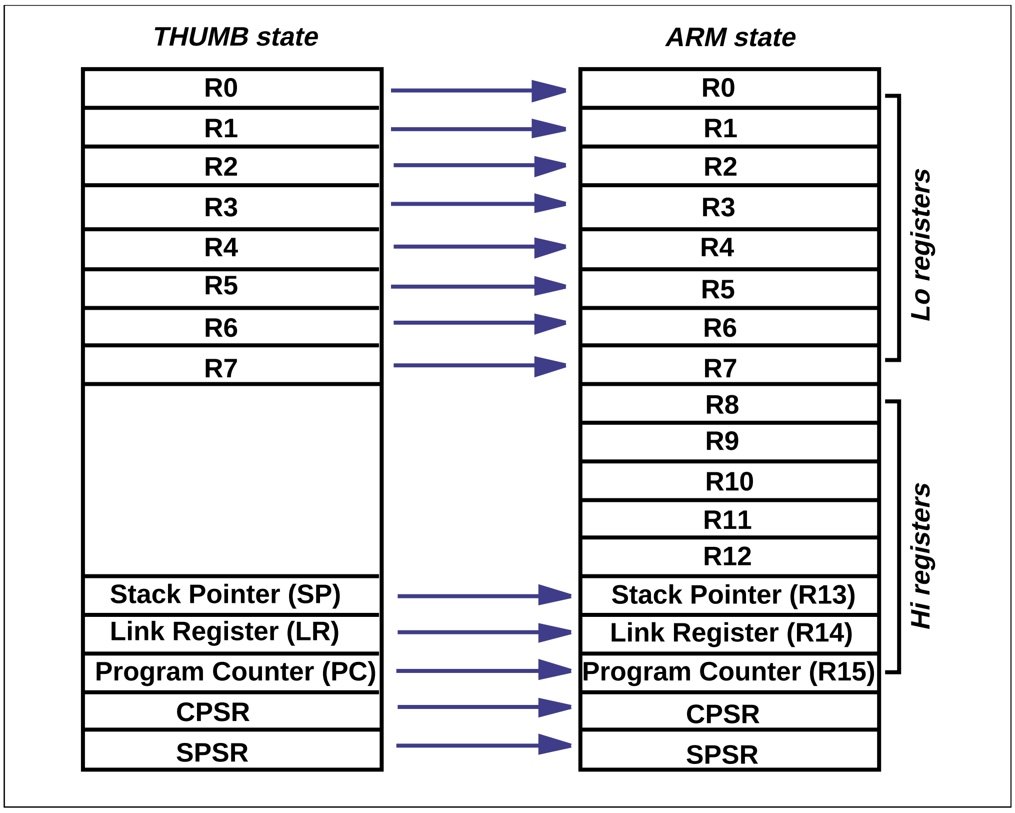


Figure 3-5: Mapping of THUMB state registers onto ARM state registers

3.7.4 Accessing Hi registers in THUMB state

In THUMB state, registers R8-R15 (the *Hi registers*) are not part of the standard register set. However, the assembly language programmer has limited access to them, and can use them for fast temporary storage.

A value may be transferred from a register in the range R0-R7 (a *Lo register*) to a Hi register, and from a Hi register to a Lo register, using special variants of the `MOV` instruction. Hi register values can also be compared against or added to Lo register values with the `CMP` and `ADD` instructions. See ➲*5.5 Format 5: Hi register operations/ branch exchange* on page 5-13.

3.8 The Program Status Registers

The ARM7TDMI contains a Current Program Status Register (CPSR), plus five Saved Program Status Registers (SPSRs) for use by exception handlers. These registers

- hold information about the most recently performed ALU operation
- control the enabling and disabling of interrupts
- set the processor operating mode

The arrangement of bits is shown in *Figure 3-6: Program status register format.*

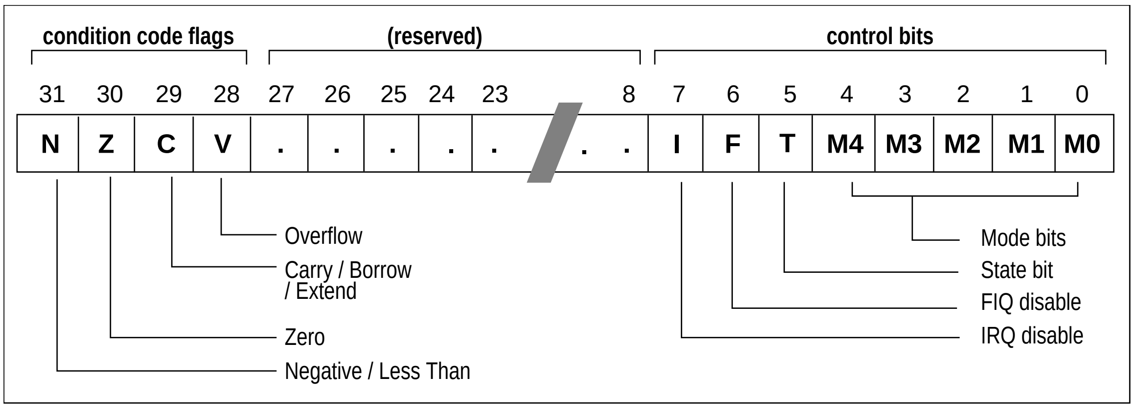


Figure 3-6: Program status register format

3.8.1 The condition code flags

The N, Z, C and V bits are the condition code flags. These may be changed as a result of arithmetic and logical operations, and may be tested to determine whether an instruction should be executed.

In ARM state, all instructions may be executed conditionally: see *4.2 The Condition Field* on page 4-5 for details.

In THUMB state, only the Branch instruction is capable of conditional execution: see *5.17 Format 17: software interrupt* on page 5-38

3.8.2 The control bits

The bottom 8 bits of a PSR (incorporating I, F, T and M[4:0]) are known collectively as the control bits. These will change when an exception arises. If the processor is operating in a privileged mode, they can also be manipulated by software.

The T bit This reflects the operating state. When this bit is set, the processor is executing in THUMB state, otherwise it is executing in ARM state. This is reflected on the **TBIT** external signal.

Note that the software must never change the state of the **TBIT** in the CPSR. If this happens, the processor will enter an unpredictable state.

Interrupt disable bits The I and F bits are the interrupt disable bits. When set, these disable the IRQ and FIQ interrupts respectively.

The mode bits The M4, M3, M2, M1 and M0 bits (M[4:0]) are the mode bits. These determine the processor's operating mode, as shown in *Table 3-1: PSR mode bit values* on page 3-9. Not all combinations of the mode bits define a valid processor mode. Only those explicitly described shall be used. The user should be aware that if any illegal value is programmed into the mode bits, M[4:0], then the processor will enter an unrecoverable state. If this occurs, reset should be applied.

M[4:0]	Mode	Visible THUMB state registers	Visible ARM state registers
10000	User	R7..R0, LR, SP PC, CPSR	R14..R0, PC, CPSR
10001	FIQ	R7..R0, LR_fiq, SP_fiq PC, CPSR, SPSR_fiq	R7..R0, R14_fiq..R8_fiq, PC, CPSR, SPSR_fiq
10010	IRQ	R7..R0, LR_irq, SP_irq PC, CPSR, SPSR_irq	R12..R0, R14_irq..R13_irq, PC, CPSR, SPSR_irq
10011	Supervisor	R7..R0, LR_svc, SP_svc, PC, CPSR, SPSR_svc	R12..R0, R14_svc..R13_svc, PC, CPSR, SPSR_svc
10111	Abort	R7..R0, LR_abt, SP_abt, PC, CPSR, SPSR_abt	R12..R0, R14_abt..R13_abt, PC, CPSR, SPSR_abt
11011	Undefined	R7..R0 LR_und, SP_und, PC, CPSR, SPSR_und	R12..R0, R14_und..R13_und, PC, CPSR
11111	System	R7..R0, LR, SP PC, CPSR	R14..R0, PC, CPSR

Table 3-1: PSR mode bit values

Reserved bits The remaining bits in the PSRs are reserved. When changing a PSR's flag or control bits, you must ensure that these unused bits are not altered. Also, your program should not rely on them containing specific values, since in future processors they may read as one or zero.

3.9 Exceptions

Exceptions arise whenever the normal flow of a program has to be halted temporarily, for example to service an interrupt from a peripheral. Before an exception can be handled, the current processor state must be preserved so that the original program can resume when the handler routine has finished.

It is possible for several exceptions to arise at the same time. If this happens, they are dealt with in a fixed order - see *3.9.10 Exception priorities* on page 3-14.

3.9.1 Action on entering an exception

When handling an exception, the ARM7TDMI:

1. Preserves the address of the next instruction in the appropriate Link Register. If the exception has been entered from ARM state, then the address of the next instruction is copied into the Link Register (that is, current PC + 4 or PC + 8 depending on the exception. See *Table 3-2: Exception entry/exit* on page 3-11 for details). If the exception has been entered from THUMB state, then the value written into the Link Register is the current PC offset by a value such that the program resumes from the correct place on return from the exception. This means that the exception handler need not determine which state the exception was entered from. For example, in the case of SWI, `MOVS PC, R14_svc` will always return to the next instruction regardless of whether the SWI was executed in ARM or THUMB state.
2. Copies the CPSR into the appropriate SPSR
3. Forces the CPSR mode bits to a value which depends on the exception
4. Forces the PC to fetch the next instruction from the relevant exception vector

It may also set the interrupt disable flags to prevent otherwise unmanageable nestings of exceptions.

If the processor is in THUMB state when an exception occurs, it will automatically switch into ARM state when the PC is loaded with the exception vector address.

3.9.2 Action on leaving an exception

On completion, the exception handler:

1. Moves the Link Register, minus an offset where appropriate, to the PC. (The offset will vary depending on the type of exception.)
2. Copies the SPSR back to the CPSR
3. Clears the interrupt disable flags, if they were set on entry

Note *An explicit switch back to THUMB state is never needed, since restoring the CPSR from the SPSR automatically sets the T bit to the value it held immediately prior to the exception.*

3.9.3 Exception entry/exit summary

➲*Table 3-2: Exception entry/exit* summarises the PC value preserved in the relevant R14 on exception entry, and the recommended instruction for exiting the exception handler.

	Return Instruction	**Previous State**		**Notes**
		ARM R14_x	**THUMB R14_x**	
BL	MOV PC, R14	PC + 4	PC + 2	1
SWI	MOVS PC, R14_svc	PC + 4	PC + 2	1
UDEF	MOVS PC, R14_und	PC + 4	PC + 2	1
FIQ	SUBS PC, R14_fiq, #4	PC + 4	PC + 4	2
IRQ	SUBS PC, R14_irq, #4	PC + 4	PC + 4	2
PABT	SUBS PC, R14_abt, #4	PC + 4	PC + 4	1
DABT	SUBS PC, R14_abt, #8	PC + 8	PC + 8	3
RESET	NA	-	-	4

Table 3-2: Exception entry/exit

Notes

1 Where PC is the address of the BL/SWI/Undefined Instruction fetch which had the prefetch abort.

2 Where PC is the address of the instruction which did not get executed since the FIQ or IRQ took priority.

3 Where PC is the address of the Load or Store instruction which generated the data abort.

4 The value saved in R14_svc upon reset is unpredictable.

3.9.4 FIQ

The FIQ (Fast Interrupt Request) exception is designed to support a data transfer or channel process, and in ARM state has sufficient private registers to remove the need for register saving (thus minimising the overhead of context switching).

FIQ is externally generated by taking the **nFIQ** input LOW. This input can except either synchronous or asynchronous transitions, depending on the state of the **ISYNC** input signal. When **ISYNC** is LOW, **nFIQ** and **nIRQ** are considered asynchronous, and a cycle delay for synchronization is incurred before the interrupt can affect the processor flow.

Irrespective of whether the exception was entered from ARM or Thumb state, a FIQ handler should leave the interrupt by executing

```
SUBS PC,R14_fiq,#4
```

FIQ may be disabled by setting the CPSR's F flag (but note that this is not possible from User mode). If the F flag is clear, ARM7TDMI checks for a LOW level on the output of the FIQ synchroniser at the end of each instruction.

3.9.5 IRQ

The IRQ (Interrupt Request) exception is a normal interrupt caused by a LOW level on the **nIRQ** input. IRQ has a lower priority than FIQ and is masked out when a FIQ sequence is entered. It may be disabled at any time by setting the I bit in the CPSR, though this can only be done from a privileged (non-User) mode.

Irrespective of whether the exception was entered from ARM or Thumb state, an IRQ handler should return from the interrupt by executing

```
SUBS PC,R14_irq,#4
```

3.9.6 Abort

An abort indicates that the current memory access cannot be completed. It can be signalled by the external **ABORT** input. ARM7TDMI checks for the abort exception during memory access cycles.

There are two types of abort:

Prefetch abort	occurs during an instruction prefetch.
Data abort	occurs during a data access.

If a prefetch abort occurs, the prefetched instruction is marked as invalid, but the exception will not be taken until the instruction reaches the head of the pipeline. If the instruction is not executed - for example because a branch occurs while it is in the pipeline - the abort does not take place.

If a data abort occurs, the action taken depends on the instruction type:

1. Single data transfer instructions (LDR, STR) write back modified base registers: the Abort handler must be aware of this.
2. The swap instruction (SWP) is aborted as though it had not been executed.
3. Block data transfer instructions (LDM, STM) complete. If write-back is set, the base is updated. If the instruction would have overwritten the base with data (ie it has the base in the transfer list), the overwriting is prevented. All register overwriting is prevented after an abort is indicated, which means in particular that R15 (always the last register to be transferred) is preserved in an aborted LDM instruction.

The abort mechanism allows the implementation of a demand paged virtual memory system. In such a system the processor is allowed to generate arbitrary addresses. When the data at an address is unavailable, the Memory Management Unit (MMU) signals an abort. The abort handler must then work out the cause of the abort, make the requested data available, and retry the aborted instruction. The application program needs no knowledge of the amount of memory available to it, nor is its state in any way affected by the abort.

After fixing the reason for the abort, the handler should execute the following irrespective of the state (ARM or Thumb):

`SUBS PC,R14_abt,#4` for a prefetch abort, or

`SUBS PC,R14_abt,#8` for a data abort

This restores both the PC and the CPSR, and retries the aborted instruction.

3.9.7 Software interrupt

The software interrupt instruction (SWI) is used for entering Supervisor mode, usually to request a particular supervisor function. A SWI handler should return by executing the following irrespective of the state (ARM or Thumb):

```
MOV PC, R14_svc
```

This restores the PC and CPSR, and returns to the instruction following the SWI.

3.9.8 Undefined instruction

When ARM7TDMI comes across an instruction which it cannot handle, it takes the undefined instruction trap. This mechanism may be used to extend either the THUMB or ARM instruction set by software emulation.

After emulating the failed instruction, the trap handler should execute the following irrespective of the state (ARM or Thumb):

```
MOVS PC,R14_und
```

This restores the CPSR and returns to the instruction following the undefined instruction.

3.9.9 Exception vectors

The following table shows the exception vector addresses.

Address	Exception	Mode on entry
0x00000000	Reset	Supervisor
0x00000004	Undefined instruction	Undefined
0x00000008	Software interrupt	Supervisor
0x0000000C	Abort (prefetch)	Abort
0x00000010	Abort (data)	Abort
0x00000014	*Reserved*	*Reserved*
0x00000018	IRQ	IRQ
0x0000001C	FIQ	FIQ

Table 3-3: Exception vectors

3.9.10 Exception priorities

When multiple exceptions arise at the same time, a fixed priority system determines the order in which they are handled:

Highest priority:

1	Reset
2	Data abort
3	FIQ
4	IRQ
5	Prefetch abort

Lowest priority:

6	Undefined Instruction, Software interrupt.

Not all exceptions can occur at once:

Undefined Instruction and Software Interrupt are mutually exclusive, since they each correspond to particular (non-overlapping) decodings of the current instruction.

If a data abort occurs at the same time as a FIQ, and FIQs are enabled (ie the CPSR's F flag is clear), ARM7TDMI enters the data abort handler and then immediately proceeds to the FIQ vector. A normal return from FIQ will cause the data abort handler to resume execution. Placing data abort at a higher priority than FIQ is necessary to ensure that the transfer error does not escape detection. The time for this exception entry should be added to worst-case FIQ latency calculations.

3.10 Interrupt Latencies

The worst case latency for FIQ, assuming that it is enabled, consists of the longest time the request can take to pass through the synchroniser (*Tsyncmax* if asynchronous), plus the time for the longest instruction to complete (*Tldm*, the longest instruction is an LDM which loads all the registers including the PC), plus the time for the data abort entry (*Texc*), plus the time for FIQ entry (*Tfiq*). At the end of this time ARM7TDMI will be executing the instruction at 0x1C.

Tsyncmax is 3 processor cycles, *Tldm* is 20 cycles, *Texc* is 3 cycles, and *Tfiq* is 2 cycles. The total time is therefore 28 processor cycles. This is just over 1.4 microseconds in a system which uses a continuous 20 MHz processor clock. The maximum IRQ latency calculation is similar, but must allow for the fact that FIQ has higher priority and could delay entry into the IRQ handling routine for an arbitrary length of time. The minimum latency for FIQ or IRQ consists of the shortest time the request can take through the synchroniser (*Tsyncmin*) plus *Tfiq*. This is 4 processor cycles.

3.11 Reset

When the **nRESET** signal goes LOW, ARM7TDMI abandons the executing instruction and then continues to fetch instructions from incrementing word addresses.

When **nRESET** goes HIGH again, ARM7TDMI:

1. Overwrites R14_svc and SPSR_svc by copying the current values of the PC and CPSR into them. The value of the saved PC and SPSR is not defined.
2. Forces M[4:0] to 10011 (Supervisor mode), sets the I and F bits in the CPSR, and clears the CPSR's T bit.
3. Forces the PC to fetch the next instruction from address 0x00.
4. Execution resumes in ARM state.