ARM 7100

Preliminary Data Sheet

Document Number: ARM DDI 0035A
Issued: January 1996

Preliminary

Proprietary Notice

ARM and the ARM Powered logo are trademarks of Advanced RISC Machines Ltd.

SPI is a registered trademark of Motorola.
Microwire is a registered trademark of National Semiconductor.

Neither the whole nor any part of the information contained in, or the product described in, this datasheet may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this datasheet is subject to continuous developments and improvements. All particulars of the product and its use contained in this datasheet are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties or merchantability, or fitness for purpose, are excluded.

This datasheet is intended only to assist the reader in the use of the product. ARM Ltd shall not be liable for any loss or damage arising from the use of any information in this datasheet, or any error or omission in such information, or any incorrect use of the product.

Change Log

Issue	Date	By	Change
-01	Jan 95	AW	Created
-02	Feb 95	AW	Changes after an initial preliminary review. Changes to reflect updated specification.
A draft1	Dec 95	AP	Changes to reflect updated specification.
A	Jan 96	AP	Minor edits; addition of timing diagrams.

Preface

The ARM7100 is a high integration microcontroller particularly well-suited for PDAs, smart mobile phones, handheld games, portable instruments and similar applications. Built around the ARM710 microprocessor, the ARM7100 integrates LCD control, glueless DRAM interface, UART with infra-red SIR protocol support and the other peripherals required for handheld computing applications.

In normal operation at 18.4MHz at 3.3V, the ARM7100 consumes an extremely low 70mW and in standby, less than 40µW, which gives excellent battery performance. System power is minimized by the ability to use self-refresh DRAM when in standby.

ARM7100 was implemented using a modular design methodology and the AMBA internal bus architecture. The ARM7100 is the ideal starting point from which to consider further ASSP devices for volume OEM applications.

Applications

High integration and low power consumption makes the ARM7100 ideal for battery-powered portable computing applications:

- PDAs
- Smart Mobile phones
- Handheld games
- Electronic books and organizers
- Handheld instruments and data collection devices
- High specification pagers
- Mobile epos terminals

Its high performance and low cost features make the ARM7100 also suitable for:

- Office automation (photocopiers, faxes)
- Automotive user consoles

Features

- 32-bit ARM710 RISC cached processor
- 8Kb cache, memory management unit and write buffer to deliver strong performance with inexpensive memory
- Very low chip and system power consumption (two low power modes and advanced power management)
- Glueless DRAM interface, which supports self-refresh DRAM to further reduce system power consumption
- 3072Mb total physical address range
- Support for 8-bit, 16-bit or 32-bit wide ROM/SRAM devices
- 36 bits of general purpose I/O
- Flexible LCD controller with DMA support
- Full duplex UART with two 16-byte FIFOs and IrDA industry standard infra-red protocol support
- Synchronous serial interface supporting multiple protocols for peripheral expansion
- Telephony CODEC interface
- Other peripherals include timer/counters, real-time clock, DC-DC converter interface and on-chip clock generators

Contents

1	**Introduction**		**1-1**
	1.1	Introduction	1-2
	1.2	System Description	1-2
	1.3	Block Diagram	1-3
	1.4	CPU Core	1-4
	1.5	Datasheet Notation	1-4
2	**Signal Description**		**2-1**
	2.1	Signal Descriptions	2-2
3	**The ARM Processor Macrocell**		**3-1**
	3.1	Introduction	3-2
	3.2	Instruction set	3-2
	3.3	Memory Interface	3-3
	3.4	Clocking	3-3
	3.5	ARM Processor Block Diagram	3-3
4	**The ARM Processor Programmer's Model**		**4-1**
	4.1	Introduction	4-2
	4.2	Register Configuration	4-2
	4.3	Operating Mode Selection	4-4
	4.4	Registers	4-4
	4.5	Exceptions	4-7
	4.6	Configuration Control Registers	4-11
	4.7	Reset	4-16
5	**ARM Processor Instruction Set**		**5-1**
	5.1	Instruction Set Summary	5-2
	5.2	The Condition Field	5-3
	5.3	Branch and Branch with link (B, BL)	5-4
	5.4	Data Processing	5-6
	5.5	PSR Transfer (MRS, MSR)	5-15
	5.6	Multiply and Multiply-Accumulate (MUL, MLA)	5-19
	5.7	Single Data Transfer (LDR, STR)	5-21

Preliminary

Contents

	5.8	Block Data Transfer (LDM, STM)	5-27
	5.9	Single Data Swap (SWP)	5-34
	5.10	Software Interrupt (SWI)	5-36
	5.11	Coprocessor Instructions	5-38
	5.12	Coprocessor data operations (CDP)	5-39
	5.13	Coprocessor Data Transfers (LDC, STC)	5-41
	5.14	Coprocessor Register Transfers (MRC, MCR)	5-44
	5.15	Undefined Instruction	5-47
	5.16	Instruction Set Examples	5-48
	5.17	Instruction Speed Summary	5-52
6	**Cache, Write Buffer and Coprocessors**		**6-1**
	6.1	Instruction and Data Cache	6-2
	6.2	Read-lock-write	6-3
	6.3	IDC Enable/Disable and Reset	6-3
	6.4	Write Buffer	6-4
	6.5	Coprocessors	6-5
7	**ARM Processor MMU**		**7-1**
	7.1	Introduction	7-2
	7.2	MMU Program Accessible Registers	7-3
	7.3	Address Translation	7-4
	7.4	Translation Process	7-5
	7.5	Translating Section References	7-8
	7.6	Translating Small Page References	7-10
	7.7	Translating Large Page References	7-11
	7.8	MMU Faults and CPU Aborts	7-12
	7.9	Fault Address and Fault Status Registers (FAR and FSR)	7-13
	7.10	Domain Access Control	7-14
	7.11	Fault Checking Sequence	7-15
	7.12	Interaction of the MMU, IDC and Write Buffer	7-18
	7.13	Effect of Reset	7-19
8	**ARM7100 Programmer's Model**		**8-1**
	8.1	Introduction	8-2
	8.2	Summary of Registers	8-3
	8.3	Register Descriptions	8-5
9	**Interrupt Controller**		**9-1**
	9.1	Interrupt Controller	9-2
10	**The Expansion and ROM Interface**		**10-1**
	10.1	The Expansion and ROM Interface	10-2
11	**DRAM controller**		**11-1**
	11.1	DRAM Controller	11-2
12	**CODEC Interface**		**12-1**
	12.1	CODEC Interface	12-2
13	**Synchronous Serial Interface**		**13-1**
	13.1	Synchronous Serial Interface	13-2
14	**LCD Controller**		**14-1**
	14.1	LCD Controller	14-2

15 UART and SiR Encoder 15-1
- 15.1 UART 15-2
- 15.2 SiR Encoder 15-2

16 Timer Counters 16-1
- 16.1 Timer Counters 16-2
- 16.2 Real Time Clock 16-2

17 DC to DC Converters 17-1
- 17.1 DC to DC Converter Interfaces 17-2

18 Power Management and Reset 18-1
- 18.1 State Control 18-2
- 18.2 Reset 18-3

19 Memory Map 19-1
- 19.1 Memory Map 19-2

20 DC and AC Parameters 20-1
- 20.1 Absolute Maximum Ratings 20-2
- 20.2 DC Operating Conditions 20-2
- 20.3 DC Characteristics 20-3
- 20.4 AC Characteristics 20-5

21 Physical Details 21-1
- 21.1 Pin diagrams for the ARM7100 21-2

22 Pinout 22-1
- 22.1 Pin details 22-2

Contents

1 Introduction

This chapter provides an introduction to the ARM7100.

1.1	Introduction	1-2
1.2	System Description	1-2
1.3	Block Diagram	1-3
1.4	CPU Core	1-4
1.5	Datasheet Notation	1-4

Preliminary

1.1 Introduction

The ARM7100 is a highly integrated single chip microcontroller for PDA products, using modular design techniques based on the Advanced Microcontroller Bus Architecture (AMBA) to simplify design and test while optimizing for lowest power (70mW) and low die size. The ARM7100 delivers 18.4 MIPS (peak) at 3.3V and contains an embedded ARM710a core (including 8kByte cache and MMU) with ARM-library peripherals such as an LCD controller, UART and CODEC interface.

1.2 System Description

ARM7100 is based around the ARM710a processor core.

The principle functional blocks in ARM7100 are:

- ARM7 CPU core
- memory management unit
- 8Kb of unified instruction and data cache
- interrupt and fast interrupt controller
- expansion and ROM interface giving 8 x 256 Mb expansion segments with independent wait state control
- DRAM controller supporting fast page mode and self refresh in standby
- 36 bits of general purpose peripheral I/O
- telephony CODEC interface with 16-byte FIFOs
- programmable 4-bit per pixel LCD controller
- full duplex UART and two 16-byte FIFOs plus logic to implement the IrDA SIR protocol; capable of speeds up to 115K bits per second
- two 16-bit general purpose counter timers
- A 32-bit real time clock and comparator
- two DC to DC converter interfaces
- system state control and power management
- synchronous serial interface for Microwire or SPI peripherals such as ADCs
- pin test and device isolation logic
- external tracing support for debug
- a main 3.68MHz oscillator with PLL to create system frequency of 18.432MHz
- a low power 32.768 KHz oscillator

Figure 1-1: ARM7100 block diagram on page 1-3 shows a simplified block diagram of ARM7100.

1.3 Block Diagram

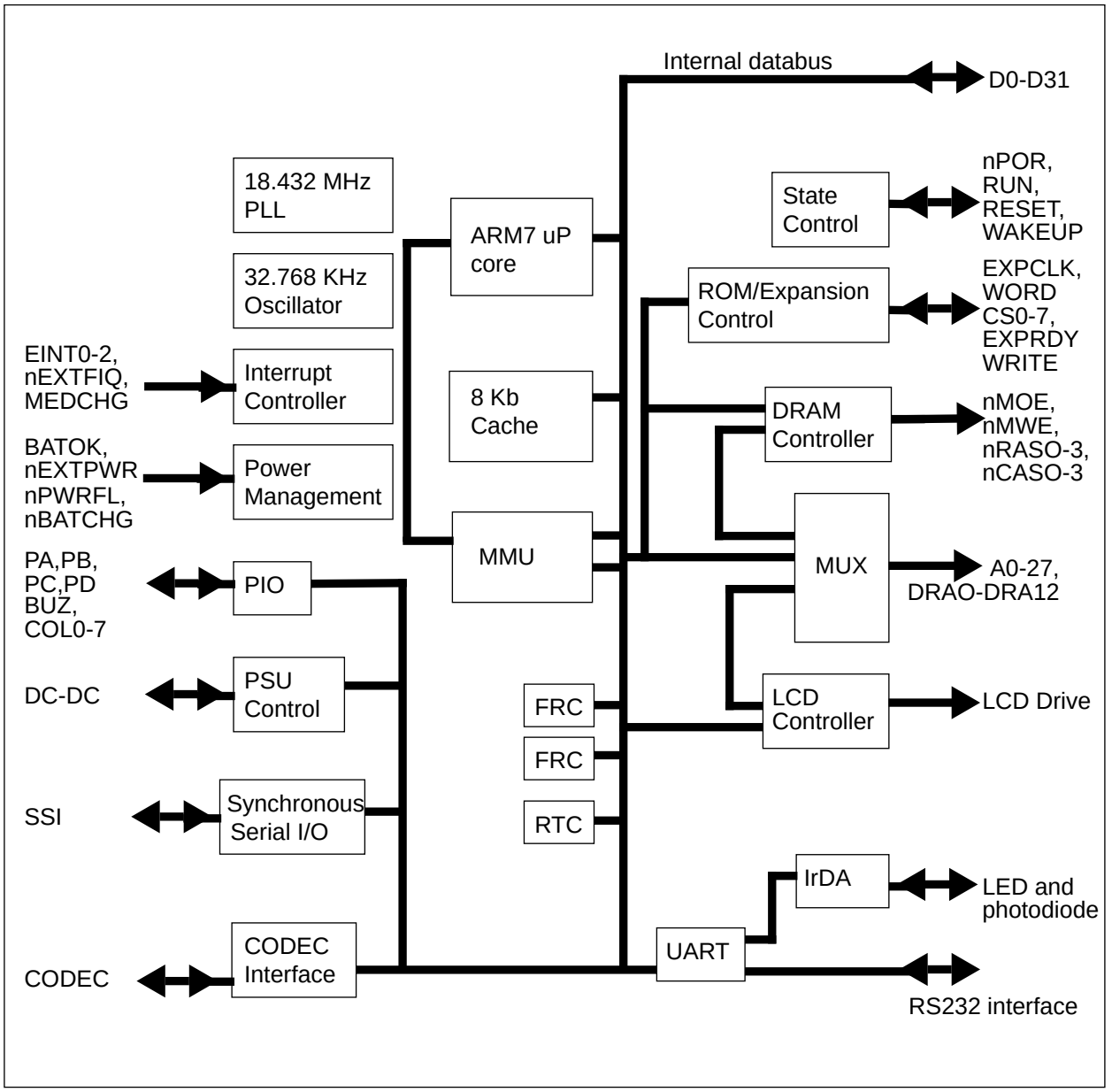


Figure 1-1: ARM7100 block diagram

1.4 CPU Core

The processor macrocell contains the ARM7 processor core with 8Kb of cache, memory management unit and write buffer. See *Chapter 3, The ARM Processor Macrocell* for a description of the ARM processor macrocell.

1.5 Datasheet Notation

0x	marks a Hexadecimal quantity
BOLD	external signals are shown in bold capital letters
binary	where it is not clear that a quantity is binary, it is followed by the word binary

2 Signal Description

This chapter gives the name, type and relevant details of each of the ARM7100 signals.

2.1 Signal Descriptions 2-2

Preliminary

2.1 Signal Descriptions

Name	Type	Description:
D[0:31]	IO	32-bit system data bus for DRAM, ROM, and memory mapped expansion.
A[0:14]	O	Least significant 15 bits of system byte address during ROM and expansion cycles.
A[15] / DRA[12]: A[27] / DRA[0]	O	13-bit multiplexed DRAM word address during DRAM cycles or address bits 16 to 27 of system byte address during ROM and expansion cycles.
nRAS[0:3]	O	DRAM RAS outputs to DRAM banks 0 to 3.
nCAS[0:3]	OM	DRAM CAS outputs for bytes 0 to 3 within 32-bit word.
nMOE	O	DRAM, ROM and expansion output enable.
nMWE	O	DRAM, ROM and expansion write enable.
nCS[0:3]	O	Expansion channel I/O strobes. Active LOW SRAM like chip selects for expansion.
CS[4:7]	O	Expansion channel I/O strobes. Active HIGH SRAM like chip selects for expansion.
EXPRDY	I	Expansion channel ready. External devices drive this LOW to extend expansion bus cycles.
WRITE	O	Transfer direction, LOW during reads, HIGH during writes from ARM7100.
WORD	O	Word access enable. Driven HIGH during word wide cycles, LOW during byte wide cycles.
EXPCLK	O	Expansion clock output. Clock output at the same phase and speed as the CPU clock. Free running or active only during expansion I/O cycles.
MEDCHG	I	The MEDCHG input is intended to be driven by a system sensor indicating that a device connected to an external system port has been physically removed or inserted. It can cause an interrupt to allow software to take appropriate action.
nEXTFIQ	I	External active LOW fast interrupt request input.
EINT3	I	External active HIGH interrupt request input.
nEINT1-2	I	Two general purpose, active LOW interrupt inputs.
nPWRFL	I	Power fail input. Active LOW deglitched input to force system into the standby state automatically.
BATOK	I	Main battery OK input. Falling edge generates a **FIQ**, a low level while in standby inhibits system start up. Deglitched input.

Table 2-1: Signal descriptions

Name	Type	Description:
nEXTPWR	I	External power sense. Must be driven LOW if the system is powered by external source.
nBATCHG	I	New battery sense. Should be driven LOW if battery voltage falls below the no-battery threshold.
nPOR	IS	Power-on reset input. Active LOW input completely resets the system.
RUN	O	System active output. HIGH when system is active or idle, LOW while in the standby state.
WAKEUP	IS	Wake up input signal. Rising edge forces system into operating state.
nURESET	IS	User reset input. Active LOW input.
PCMCK	O	CODEC clock output.
PCMSYNC	O	CODEC synchronisation pulse output.
PCMOUT	O	CODEC serial data output.
PCMIN	I	CODEC serial data input.
ADCCLK	O	Synchronous serial interface ADC clock output.
SMPLCK	O	Synchronous serial interface ADC sample clock, can be disabled.
nADCCS	O	Synchronous serial interface ADC active LOW chip select and synchronisation output.
ADCOUT	O	Synchronous serial interface ADC serial data output.
ADCIN	I	Synchronous serial interface ADC serial data input.
LEDDRV	O	Infra-red LED drive output.
PHDIN	I	Infra-red photo diode input.
TXD	O	RS232 Tx output.
RXD	I	RS232 Rx input.
DSR	I	RS232 DSR input.
DCD	I	RS232 DCD input.
CTS	I	RS232 CTS input.
DD[0:3]	O	LCD display data.
CL1	O	LCD line clock.
CL2	O	LCD pixel clock.
FRM	O	LCD frame synchronisation pulse output.
M	O	LCD AC bias drive.

Table 2-1: Signal descriptions (Continued)

Name	Type	Description:
COL[0:7]	O	Keyboard column drives.
BUZ	O	This output is driven by direct software control, or can be driven by a frequency generated by timer counter interrupts. It is designed to drive a buzzer.
PA[0:7]	IO	Port A I/O.
PB[0:7]	IO	Port B I/O.
PC[0:7]	IO	Port C I/O.
PD[0:3]	IO	Port D I/O.
PD[4:7]	IOH	Port D high drive I/O
PE[0:3]	IO	Port E I/O.
DRIVE[0:1]	IOVH	DC to DC drive outputs.
FB[0:1]	I	DC to DC feedback inputs.
nTEST[0:1]	IP	Test mode select inputs (always HIGH for normal operation).
MOSCIN/OUT	-	Main 3.6864MHz oscillator for 18.432 MHz PLL.
RTCIN/OUT	-	Real time clock 32.768 KHz oscillator.

Table 2-1: Signal descriptions (Continued)

Key to signal types and drive capabilities

I	Input
IS	Schmitt input
IP	Input with internal pull-up
O	Standard drive output
OM	Medium drive output
IOVH	Very high drive I/O
IO	Standard drive I/O
IOH	High drive I/O

See *20.3 DC Characteristics* on page 20-3 for more details.

3 The ARM Processor Macrocell

This chapter introduces the ARM processor 32-bit microprocessor macrocell.

3.1	Introduction	3-2
3.2	Instruction set	3-2
3.3	Memory Interface	3-3
3.4	Clocking	3-3
3.5	ARM Processor Block Diagram	3-3

3.1 Introduction

ARM7100 contains a 32-bit RISC ARM710a processor macrocell. It has a 8Kb cache, write buffer, and a memory management unit (MMU). The ARM processor macrocell offers high-level RISC performance, yet its fully static design ensures minimal power consumption. This makes it ideal for incorporation into ARM7100.

This part of the datasheet describes the features of the ARM processor macrocell which are available to the user in its embedded state within ARM7100. It is not intended that this should be used as a standalone datasheet for a separate ARM processor macrocell.

3.1.1 Architecture

The ARM processor architecture is based on Reduced Instruction Set Computer (RISC) principles, and the instruction set and related decode mechanism are greatly simplified compared with microprogrammed Complex Instruction Set Computers (CISC).

The mixed data and instruction cache together with the write buffer substantially raise the average execution speed and reduce the average amount of memory bandwidth required by the processor.

The MMU supports a conventional two-level page-table structure and a number of extensions which make it ideal for embedded control, UNIX and Object Oriented systems.

3.2 Instruction set

The instruction set comprises ten basic instruction types:

- two of these make use of the on-chip arithmetic logic unit, barrel shifter and multiplier to perform high-speed operations on the data in a bank of 31 registers, each 32 bits wide
- three classes of instruction control data transfer between memory and the registers, one optimized for flexibility of addressing, another for rapid context switching and the third for swapping data
- two instructions control the flow and privilege level of execution
- three types are dedicated to the control of external coprocessors which allow the functionality of the instruction set to be extended in an open and uniform way. However, as for the ARM710, the facility to add external coprocessors to the ARM7100 is not available, and software emulation of coprocessor activity will be required if these instructions are to perform a defined function.

The ARM instruction set is a good target for compilers of many different high-level languages. Where required for critical code segments, assembly code programming is also straightforward, unlike some RISC processors which depend on sophisticated compiler technology to manage complicated instruction interdependencies.

3.3 Memory Interface

The 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 permit ARM7100 to exploit the page mode access offered by industry-standard DRAMs.

3.4 Clocking

ARM7100 uses the ARM processor macrocell in fastbus mode. This means that the core **FCLK** frequency is tied to the main processor input clock (**MCLK**). All references to **FCLK** in this datasheet should be read as **MCLK**.

3.5 ARM Processor Block Diagram

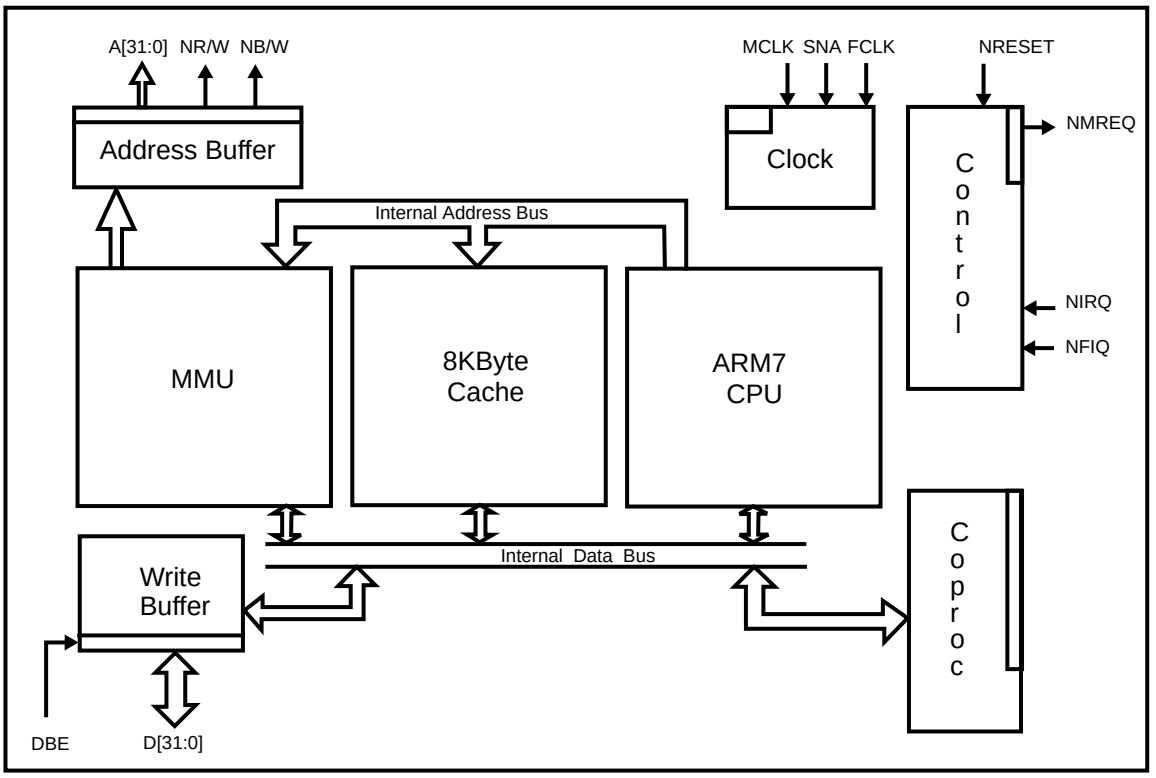


Figure 3-1: ARM processor block diagram

4 The ARM Processor Programmer's Model

This chapter describes the programmer’s model.

4.1	Introduction	4-2
4.2	Register Configuration	4-2
4.3	Operating Mode Selection	4-4
4.4	Registers	4-4
4.5	Exceptions	4-7
4.6	Configuration Control Registers	4-11
4.7	Reset	4-16

4.1 Introduction

The ARM processor supports a variety of operating configurations. Some are controlled by register bits and are known as the *register configurations*. Others may be controlled by software and these are known as *operating modes*.

4.2 Register Configuration

The ARM processor provides 3 register configuration settings which may be changed while the processor is running. These are discussed below.

4.2.1 Big and little-endian (the bigend bit)

The bigend bit in the control register sets whether the ARM7100 treats words in memory as being stored in big-endian or little-endian format. See ➢*Chapter 6, Cache, Write Buffer and Coprocessors* for more information on the Control Register. Memory is viewed 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.

In the little-endian scheme the lowest numbered byte in a word is considered to be the least significant byte of the word and the highest numbered byte is the most significant. Byte 0 of the memory system should be connected to data lines 7 through 0 (**D[7:0]**) in this scheme.

Little-endian

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

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

Figure 4-1: Little-endian addresses of bytes within word

In the big-endian scheme the most significant byte of a word is stored at the lowest numbered byte and the least significant byte is stored at the highest numbered byte. Byte 0 of the memory system should therefore be connected to data lines 31 through 24 (**D[31:24]**). Load and store are the only instructions affected by the endianness: see ➢*5.7 Single Data Transfer (LDR, STR)* on page 5-21 for more details.

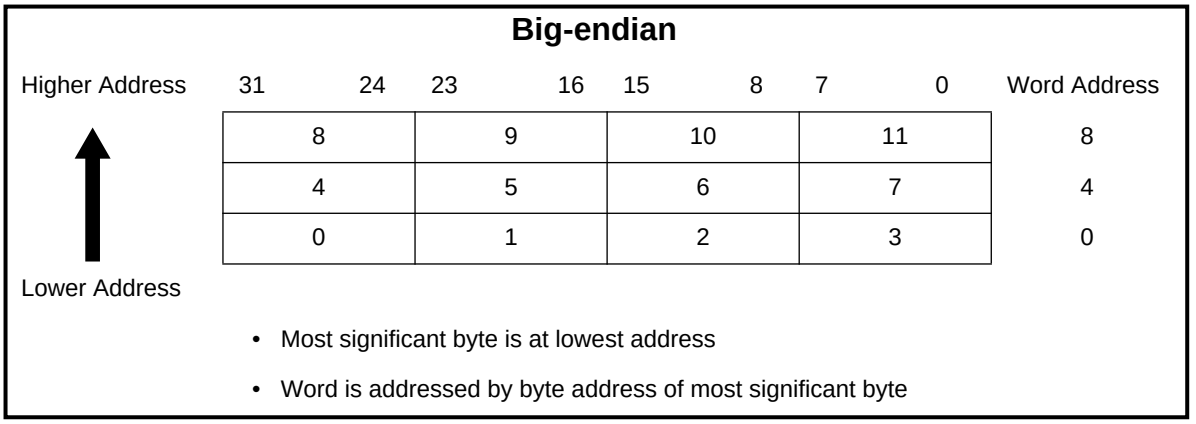


Figure 4-2: Big-endian addresses of bytes within words

4.2.2 Configuration bits for backward compatibility

The other two configuration bits, prog32 and data32, are used for backward compatibility with earlier ARM processors but should normally be set to 1. This mode is recommended for compatibility with future ARM processors and all new code should be written to use only the 32-bit operating modes.

Because the original ARM instruction set has been modified to accommodate 32-bit operation there are certain additional restrictions which programmers must be aware of. These are indicated in the text. Reference should also be made to the ARM Application Notes *Rules for ARM Code Writers* and *Notes for ARM Code Writers,* available from your supplier.

4.3 Operating Mode Selection

The processor has a 32-bit data bus and a 32-bit address bus. The processor supports *byte* (8-bit) and *word* (32-bit) data types, where words must be aligned to four byte boundaries. Instructions are exactly one word, and data operations (eg. ADD) are only performed on word quantities. Load and store operations can transfer either bytes or words.

The processor supports six modes of operation:

1. User mode (usr): the normal program execution state
2. FIQ mode (fiq): fast interrupt for data transfer or channel processes
3. IRQ mode (irq): used for general purpose interrupt handling
4. Supervisor mode (svc): a protected mode for the operating system
5. Abort mode (abt): entered after a data or instruction prefetch abort
6. Undefined mode (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 other modes, known as *privileged modes*, will be entered to service interrupts or exceptions or to access protected resources.

4.4 Registers

The processor has a total of 37 registers made up of 31 general 32-bit registers and 6 status registers. At any one time 16 general registers (R0 to R15) and one or two status registers are visible to the programmer. The visible registers depend on the processor mode. The other registers, known as the *banked registers*, are switched in to support IRQ, FIQ, Supervisor, Abort and Undefined mode processing. *Figure 4-3: Register organisation* on page 4-5 shows how the registers are arranged, with the banked registers shaded.

In all modes 16 registers, R0 to R15, are directly accessible. All registers except R15 are general purpose and may be used to hold data or address values. Register R15 holds the Program Counter (PC). When R15 is read, bits [1:0] are zero and bits [31:2] contain the PC. A seventeenth register (the CPSR - Current Program Status Register) is also accessible. It contains condition code flags and the current mode bits and may be thought of as an extension to the PC.

R14 is used as the subroutine link register and receives a copy of R15 when a Branch and Link instruction is executed. It may be treated as a general purpose register at all other times. R14_svc, R14_irq, R14_fiq, R14_abt and R14_und are used similarly 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.

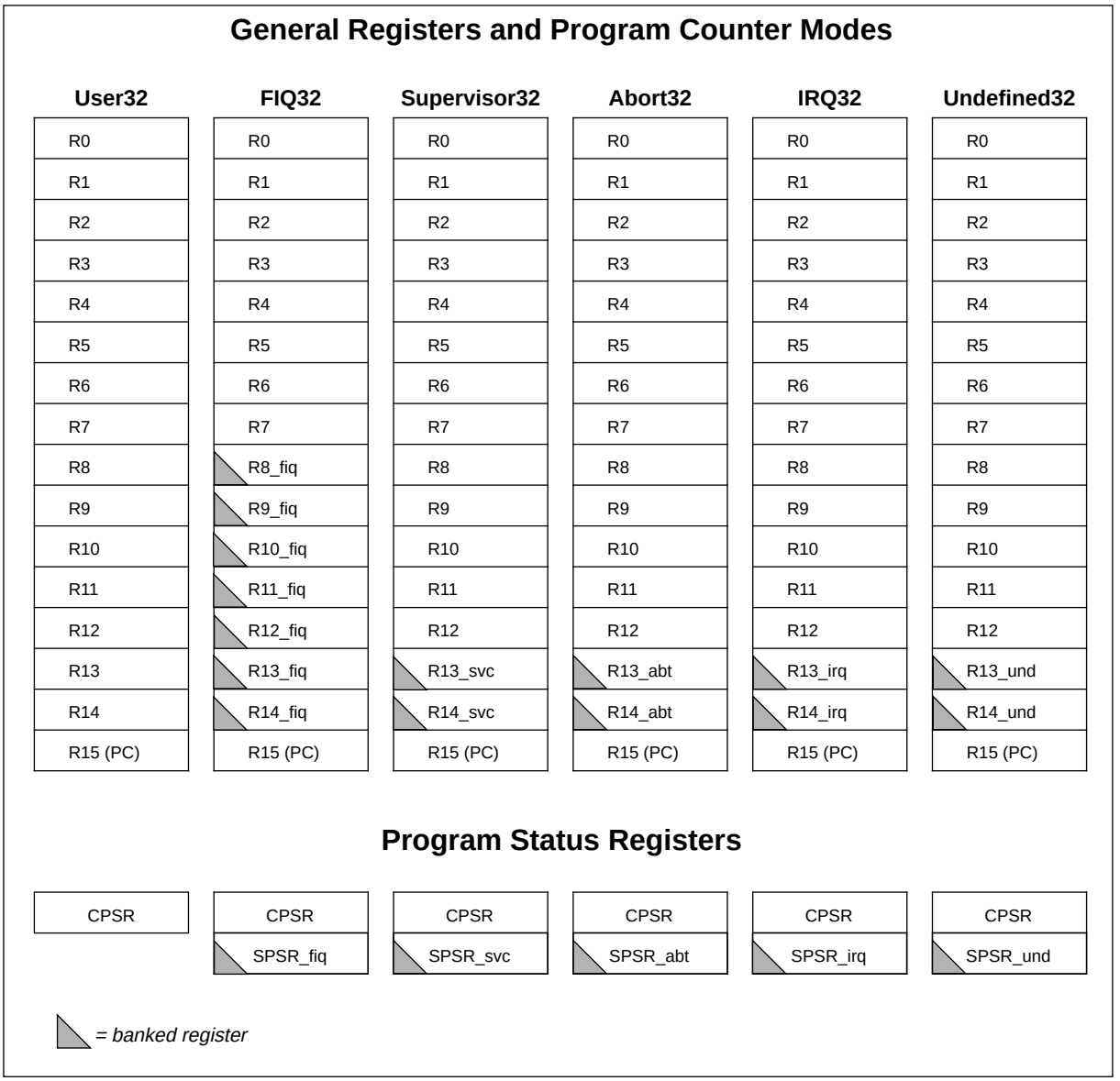


Figure 4-3: Register organisation

FIQ mode has seven banked registers mapped to R8-14 (R8_fiq-R14_fiq). Many FIQ programs will not need to save any registers. User mode, IRQ mode, Supervisor mode, Abort mode and Undefined mode each have two banked registers mapped to R13 and R14. The two banked registers allow these modes to each have a private stack pointer and link register. Supervisor, IRQ, Abort and Undefined mode programs which require more than these two banked registers are expected to save some or all of the caller's registers (R0 to R12) on their respective stacks. They are then free to use these registers which they will restore before returning to the caller. In addition there are also five SPSRs (Saved Program Status Registers) which are loaded with the CPSR when an exception occurs. There is one SPSR for each privileged mode.

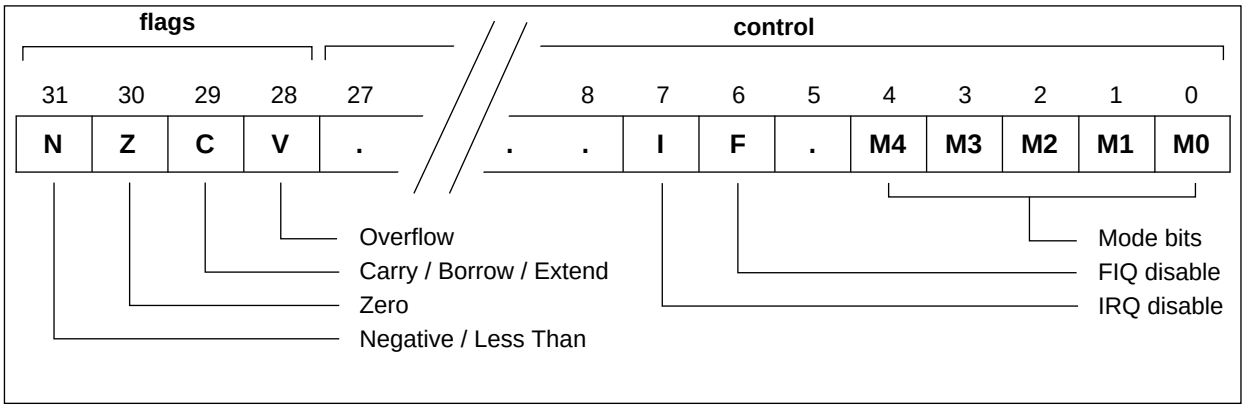


Figure 4-4: Format of the program status registers (PSRs)

The format of the Program Status Registers is shown in ➢*Figure 4-4: Format of the program status registers (PSRs)*. The N, Z, C and V bits are the *condition code flags*. The condition code flags in the CPSR may be changed as a result of arithmetic and logical operations in the processor and may be tested by all instructions to determine if the instruction is to be executed.

The I and F bits are the *interrupt disable bits.* The I bit disables IRQ interrupts when it is set and the F bit disables FIQ interrupts when it is set. The M0, M1, M2, M3 and M4 bits (M[4:0]) are the *mode bits*, and these determine the mode in which the processor operates. The interpretation of the mode bits is shown in ➢*Table 4-1: The Mode Bits*. Not all bit combinations define a valid processor mode. Only those explicitly described should be used. The user should be aware that if any illegal value is programmed into the mode bits, M[4:0], the processor will enter an unrecoverable state. If this occurs, reset should be applied.

The bottom 28 bits of a PSR (incorporating I, F and M[4:0]) are known collectively as the *control bits*. These will change when an exception arises and in addition can be manipulated by software when the processor is in a privileged mode. Unused bits in the PSRs are reserved and their state must be preserved when changing the flag or control bits. Programs must not rely on specific values from the reserved bits when checking the PSR status, since they may read as one or zero in future processors.

M[4:0]	Mode	Accessible Register Set	
10000	User	PC, R14..R0	CPSR
10001	FIQ	PC, R14_fiq..R8_fiq, R7..R0	CPSR, SPSR_fiq
10010	IRQ	PC, R14_irq..R13_irq, R12..R0	CPSR, SPSR_irq
10011	Supervisor	PC, R14_svc..R13_svc, R12..R0	CPSR, SPSR_svc
10111	Abort	PC, R14_abt..R13_abt, R12..R0	CPSR, SPSR_abt
11011	Undefined	PC, R14_und..R13_und, R12..R0	CPSR, SPSR_und

Table 4-1: The Mode Bits

4.5 Exceptions

Exceptions arise whenever there is a need for the normal flow of program execution to be broken, so that (for example) the processor can be diverted to handle an interrupt from a peripheral. The processor state just prior to handling the exception must be preserved so that the original program can be resumed when the exception routine has completed. Many exceptions may arise at the same time.

The ARM processor handles exceptions by making use of the banked registers to save state. The old PC and CPSR contents are copied into the appropriate R14 and SPSR and the PC and mode bits in the CPSR bits are forced to a value which depends on the exception. Interrupt disable flags are set where required to prevent otherwise unmanageable nestings of exceptions. In the case of a re-entrant interrupt handler, R14 and the SPSR should be saved onto a stack in main memory before re-enabling the interrupt; when transferring the SPSR register to and from a stack, it is important to transfer the whole 32-bit value, and not just the flag or control fields. When multiple exceptions arise simultaneously, a fixed priority determines the order in which they are handled. This is listed later in *4.5.7 Exception priorities* on page 4-10.

4.5.1 FIQ

The FIQ (Fast Interrupt reQuest) exception is externally generated in response to a FIQ interrupt source becoming active, causing the **nFIQ** input to the macrocell to be taken LOW. This input can except asynchronous transitions, and is delayed by one clock cycle for synchronisation before it can affect the processor execution flow. FIQ is designed to support a data transfer or channel process, and has sufficient private registers to remove the need for register saving in such applications (thus minimising the overhead of context switching). The FIQ exception may be disabled by setting the F flag in the CPSR (but note that this is not possible from User mode). If the F flag is clear, the processor checks for a LOW level on the output of the FIQ synchroniser at the end of each instruction.

When a FIQ is detected, the processor:

1. Saves the address of the next instruction to be executed plus 4 in R14_fiq; saves CPSR in SPSR_fiq
2. Forces M[4:0]=10001 (FIQ mode) and sets the F and I bits in the CPSR
3. Forces the PC to fetch the next instruction from address 0x1C

To return normally from FIQ, use SUBS PC, R14_fiq,#4 which will restore both the PC (from R14) and the CPSR (from SPSR_fiq) and resume execution of the interrupted code.

4.5.2 IRQ

The IRQ (Interrupt ReQuest) exception is a normal interrupt caused by a LOW level on the **nIRQ** input to the macrocell. It has a lower priority than FIQ, and is masked out when a FIQ sequence is entered. Its effect may be masked out at any time by setting the I bit in the CPSR (but note that this is not possible from User mode). If the I flag is clear, the processor checks for a LOW level on the output of the IRQ synchroniser at the end of each instruction. When an IRQ is detected, the processor:

1 Saves the address of the next instruction to be executed plus 4 in R14_irq; saves CPSR in SPSR_irq

2 Forces M[4:0]=10010 (IRQ mode) and sets the I bit in the CPSR

3 Forces the PC to fetch the next instruction from address 0x18

To return normally from **IRQ**, use SUBS PC,R14_irq,#4 which will restore both the PC and the CPSR and resume execution of the interrupted code.

4.5.3 Abort

An abort is signalled by the internal Memory Management Unit. An abort indicates that the current memory access cannot be completed. For instance, in a virtual memory system the data corresponding to the current address may have been moved out of memory onto a disc, and considerable processor activity may be required to recover the data before the access can be performed successfully. The ARM processor checks for aborts during memory access cycles. When successfully aborted ARM processor will respond in one of two ways:

1 If the abort occurred during an instruction prefetch (a *Prefetch Abort*), the prefetched instruction is marked as invalid but the abort exception does not occur immediately. If the instruction is not executed, for example as a result of a branch being taken while it is in the pipeline, no abort will occur. An abort will take place if the instruction reaches the head of the pipeline and is about to be executed.

2 If the abort occurred during a data access (a *Data Abort*), the action depends on the instruction type.

   a) Single data transfer instructions (LDR, STR) will write back modified base registers and the Abort handler must be aware of this.

   b) The swap instruction (SWP) is aborted as though it had not executed, though externally the read access may take place.

   c) Block data transfer instructions (LDM, STM) complete, and if write-back is set, the base is updated. If the instruction would normally have overwritten the base with data (ie. LDM with the base in the transfer list), this overwriting is prevented. All register overwriting is prevented after the Abort is indicated, which means in particular that R15 (which is always last to be transferred) is preserved in an aborted LDM instruction.

When either a prefetch or data abort occurs, the processor:

1 Saves the address of the aborted instruction plus 4 (for prefetch aborts) or 8 (for data aborts) in R14_abt; saves CPSR in SPSR_abt.

2 Forces M[4:0]=10111 (Abort mode) and sets the I bit in the CPSR.

3 Forces the PC to fetch the next instruction from either address 0x0C (prefetch abort) or address 0x10 (data abort).

To return after fixing the reason for the abort, use SUBS PC,R14_abt,#4 (for a prefetch abort) or SUBS PC,R14_abt,#8 (for a data abort). This will restore both the PC and the CPSR and retry the aborted instruction.

The abort mechanism allows a *demand paged virtual memory system* to be implemented when suitable memory management software is available. The processor is allowed to generate arbitrary addresses, and when the data at an address is unavailable the MMU signals an abort. The processor traps into system software which must 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.

4.5.4 Software interrupt

The software interrupt instruction (SWI) is used for getting into Supervisor mode, usually to request a particular supervisor function. When a SWI is executed, the processor:

1 Saves the address of the SWI instruction plus 4 in R14_svc; saves CPSR in SPSR_svc

2 Forces M[4:0]=10011 (Supervisor mode) and sets the I bit in the CPSR

3 Forces the PC to fetch the next instruction from address 0x08

To return from a SWI, use MOVS PC,R14_svc. This will restore the PC and CPSR and return to the instruction following the SWI.

4.5.5 Undefined instruction trap

When the ARM processor comes across an instruction which it cannot handle (see ◆*Chapter 5, ARM Processor Instruction Set*), it will take the undefined instruction trap. This includes all coprocessor instructions, except MCR and MRC operations which access the internal control coprocessor.

The trap may be used for software emulation of a coprocessor in a system which does not have the coprocessor hardware, or for general purpose instruction set extension by software emulation.

When the ARM processor takes the undefined instruction trap it:

1 Saves the address of the Undefined or coprocessor instruction plus 4 in R14_und; saves CPSR in SPSR_und.

2 Forces M[4:0]=11011 (Undefined mode) and sets the I bit in the CPSR

3 Forces the PC to fetch the next instruction from address 0x04

To return from this trap after emulating the failed instruction, use MOVS PC,R14_und. This will restore the CPSR and return to the instruction following the undefined instruction.

4.5.6 Vector summary

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 --	--
0x00000018	IRQ	IRQ
0x0000001C	FIQ	FIQ

Table 4-2: Vector summary

These are byte addresses, and will normally contain a branch instruction pointing to the relevant routine.

The FIQ routine might reside at 0x1C onwards, and thereby avoid the need for (and execution time of) a branch instruction.

4.5.7 Exception priorities

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

1 Reset (highest priority)

2 Data abort

3 FIQ

4 IRQ

5 Prefetch abort

6 Undefined Instruction, Software interrupt (lowest priority)

Note that 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 F flag in the CPSR is clear), ARM processor will enter the data abort handler and then immediately proceed 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.

4.6 Configuration Control Registers

The operation and configuration of the ARM processor is controlled both directly via coprocessor instructions and indirectly via the Memory Management Page tables. The coprocessor instructions manipulate a number of on-chip registers which control the configuration of the Cache, write buffer, MMU and a number of other configuration options.

To ensure backwards compatibility of future CPUs, all reserved or unused bits in registers and coprocessor instructions should be programmed to '0'. Invalid registers must not be read/written. The following bits must be programmed to '0':

Register 1 bits[31:11]

Register 2 bits[13:0]

Register 5 bits[31:0]

Register 6 bits[11:0]

Register 7 bits[31:0]

Note *The grey areas in the register and translation diagrams are reserved and should be programmed to 0 for future compatibility.*

4.6.1 Internal coprocessor instructions

These registers may be read using MRC instructions and written using MCR instructions. These operations are only allowed in non-user modes and the undefined instruction trap will be taken if accesses are attempted in user mode.

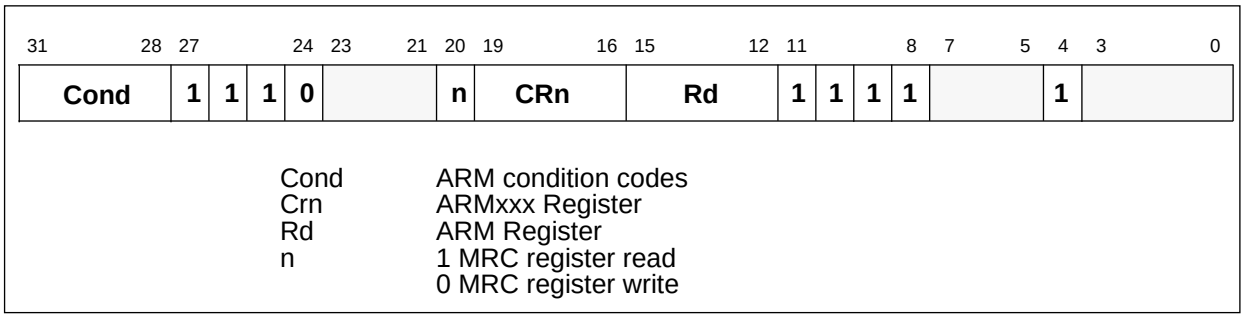


Figure 4-5: Format of internal coprocessor instructions MRC and MCR

4.6.2 Registers

The ARM processor contains registers which control the cache and MMU operation. These registers are accessed using CPRT instructions to Coprocessor #15 with the processor in a privileged mode. Only some of registers 0-7 are valid: an access to an

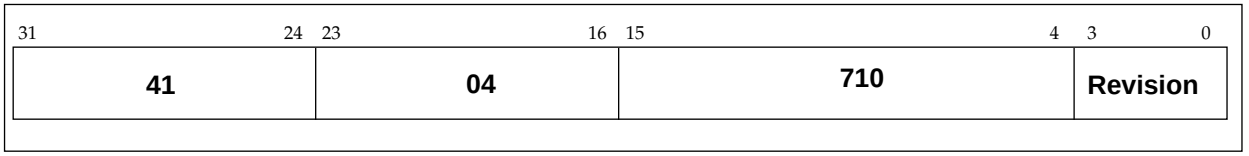
invalid register will cause neither the access nor an undefined instruction trap, and therefore should never be carried out; an access to any of the registers 8-15 will cause the undefined instruction trap to be taken.

Register	Register Reads	Register Writes
0	ID Register	Reserved
1	Reserved	Control
2	Reserved	Translation Table Base
3	Reserved	Domain Access Control
4	Reserved	Reserved
5	Fault Status	Flush TLB
6	Fault Address	Purge TLB
7	Reserved	Flush IDC
8-15	Reserved	Reserved

Table 4-3: Cache and MMU control register

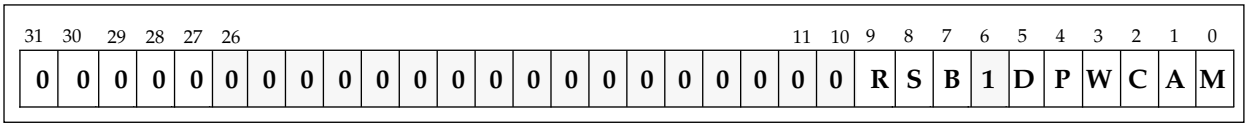
Register 0 ID

Register 0 is a read-only identity register that returns the ARM Ltd code for this chip: 0x4104710x.



Register 1 Control

Register 1 is write only and contains control bits. All bits in this register are forced LOW by reset.



M Bit 0 Enable/disable

0 - on-chip Memory Management Unit turned off

1 - on-chip Memory Management Unit turned on.

A Bit 1 Address Fault Enable/Disable

0 - alignment fault disabled

1 - alignment fault enabled

C Bit 2 Cache Enable/Disable

0 - Instruction / data cache turned off

1 - Instruction / data cache turned on

W Bit 3 Write buffer Enable/Disable

0 - Write buffer turned off

1 - Write buffer turned on

P Bit 4 ARM 32/26 Bit Program Space

0 - 26-bit Program Space selected

1 - 32-bit Program Space selected

D Bit 5 ARM 32/26 Bit Data Space

0 - 26-bit Data Space selected

1 - 32-bit Data Space selected

B Bit 7 Big/little-endian

0 - Little-endian operation

1 - Big-endian operation

S Bit 8 System

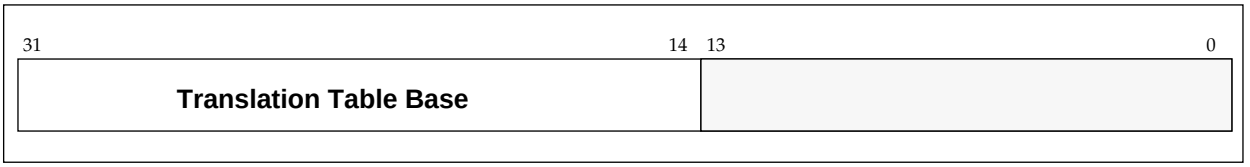
This bit controls the ARM processor permission system.

R Bit 9 ROM

This bit controls the ARM processor permission system.

Register 2 Translation Table Base

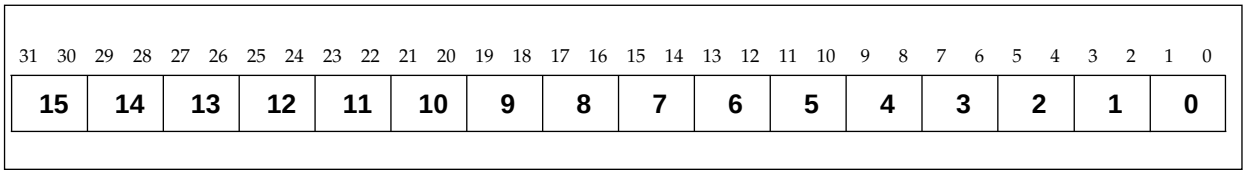
Register 2 is a write-only register which holds the base of the currently active Level One page table.



Register 3 Domain Access Control

Register 3 is a write-only register which holds the current access control for domains 0 to 15.

.



Register 4 Reserved

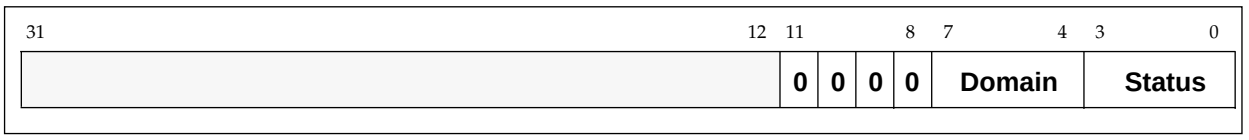
Register 4 is Reserved. Accessing this register has no effect, but should never be attempted.

Register 5

Read: Fault Status

Reading register 5 returns the status of the last data fault. It is not updated for a prefetch fault. Note that only the bottom 12 bits are returned. The upper 20 bits will be the last value on the internal data bus, and therefore will have no meaning. Bits 11:8 are always returned as zero

.



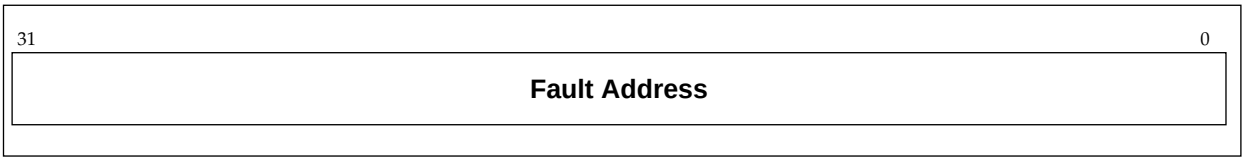
Write: Translation Lookaside Buffer Flush

Writing Register 5 flushes the TLB. (The data written is discarded).

Register 6

Read: Fault Address

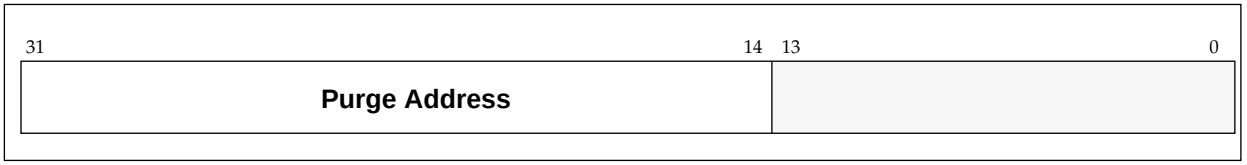
Reading register 6 returns the virtual address of the last data fault.



Write: TLB Purge

Writing Register 6 purges the TLB; the data is treated as an address and the TLB is searched for a corresponding page table descriptor. If a match is found, the corresponding entry is marked as invalid. This allows the page table descriptors in main memory to be updated and invalid entries in the on-chip TLB to be purged without requiring the entire TLB to be flushed

.



Register 7 IDC Flush

Register 7 is a write-only register. The data written to this register is discarded and the IDC is flushed.

Registers 8 - 15 Reserved

Accessing any of these registers will cause the undefined instruction trap to be taken.

4.7 Reset

When the **nRESET** input to the processor macrocell goes LOW, the ARM processor abandons the executing instruction and then performs idle cycles from incrementing word addresses.

When the **nRESET** macrocell input goes HIGH again, the ARM processor does the following:

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 CPSR is not defined.
2. Forces M[4:0]=10011 (Supervisor mode) and sets the I and F bits in the CPSR.
3. Forces the PC to fetch the next instruction from address 0x00

At the end of the reset sequence, the MMU is disabled and the TLB is flushed, so forces “flat” translation (ie. the physical address is the virtual address, and there is no permission checking); alignment faults are also disabled; the cache is disabled and flushed; the write buffer is disabled and flushed; the ARM7 CPU core is put into 26-bit data and address mode and little-endian mode.

5 ARM Processor Instruction Set

This chapter describes the ARM Processor instruction set.

5.1	Instruction Set Summary	5-2
5.2	The Condition Field	5-3
5.3	Branch and Branch with link (B, BL)	5-4
5.4	Data Processing	5-6
5.5	PSR Transfer (MRS, MSR)	5-15
5.6	Multiply and Multiply-Accumulate (MUL, MLA)	5-19
5.7	Single Data Transfer (LDR, STR)	5-21
5.8	Block Data Transfer (LDM, STM)	5-27
5.9	Single Data Swap (SWP)	5-34
5.10	Software Interrupt (SWI)	5-36
5.11	Coprocessor Instructions	5-38
5.12	Coprocessor data operations (CDP)	5-39
5.13	Coprocessor Data Transfers (LDC, STC)	5-41
5.14	Coprocessor Register Transfers (MRC, MCR)	5-44
5.15	Undefined Instruction	5-47
5.16	Instruction Set Examples	5-48
5.17	Instruction Speed Summary	5-52

5.1 Instruction Set Summary

A summary of the ARM Processor instruction set is shown in ◆*Figure 5-1: Instruction set summary*.

Note Some instruction codes are not defined but do not cause the Undefined instruction trap to be taken, for instance a Multiply instruction with bit 6 changed to a 1. These instructions must not be used, as their action may change in future ARM implementations.

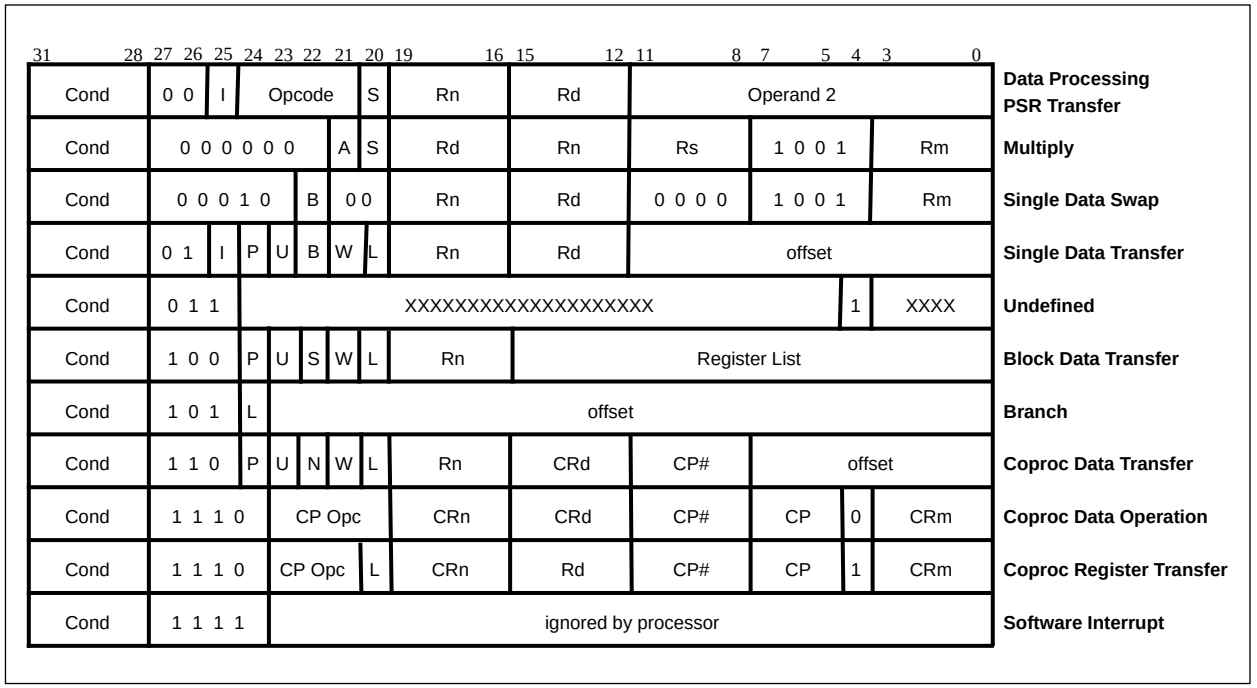


Figure 5-1: Instruction set summary

5.2 The Condition Field

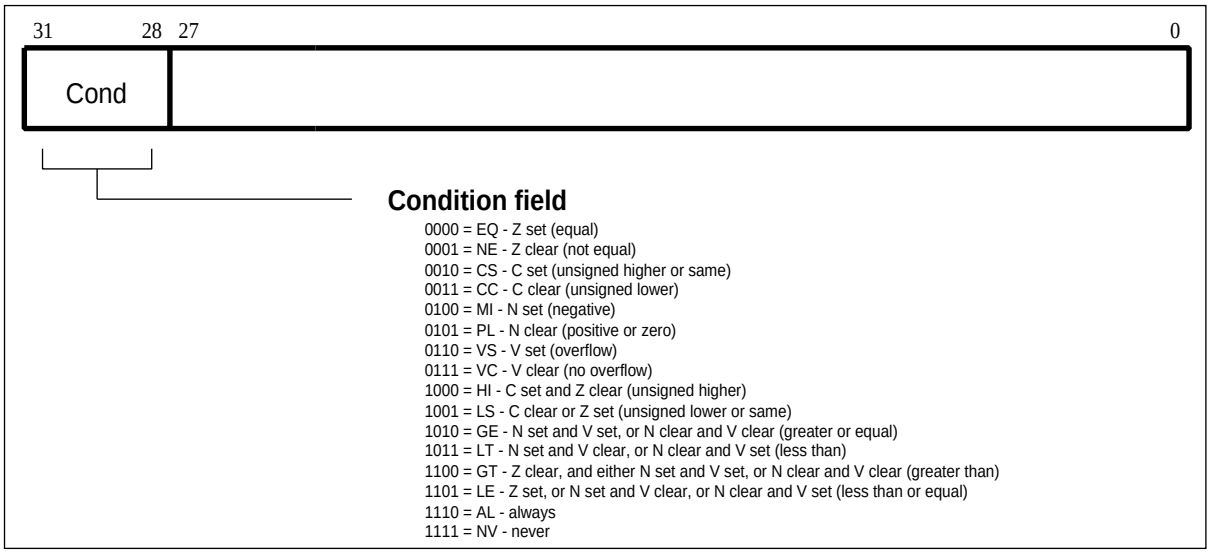


Figure 5-2: Condition codes

All ARM Processor instructions are conditionally executed, which means that their execution may or may not take place depending on the values of the N, Z, C and V flags in the CPSR. The condition encoding is shown in ◆*Figure 5-2: Condition codes*.

If the *always* (AL) condition is specified, the instruction will be executed irrespective of the flags. The *never* (NV) class of condition codes should not be used as they will be redefined in future variants of the ARM architecture. If a NOP is required, MOV R0,R0 should be used. The assembler treats the absence of a condition code as though *always* had been specified.

The other condition codes have meanings as detailed in ◆*Figure 5-2: Condition codes*, for instance code 0000 (EQual) causes the instruction to be executed only if the Z flag is set. This would correspond to the case where a compare (CMP) instruction had found the two operands to be equal. If the two operands were different, the compare instruction would have cleared the Z flag and the instruction will not be executed.

5.3 Branch and Branch with link (B, BL)

The instruction is only executed if the condition is true. The various conditions are defined at the beginning of this chapter. The instruction encoding is shown in *Figure 5-3: Branch instructions* on page 5-4.

Branch instructions contain a signed 2's complement 24-bit offset. This is shifted left two bits, sign extended to 32-bits, and added to the PC. The instruction can therefore specify a branch of +/- 32Mbytes. The branch offset must take account of the prefetch operation, which causes the PC to be 2 words (8 bytes) ahead of the current instruction.

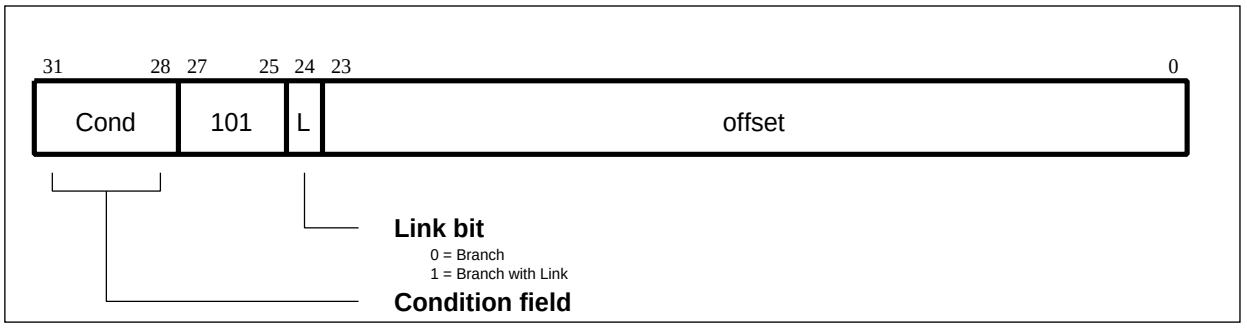


Figure 5-3: Branch instructions

Branches beyond +/- 32Mbytes must use an offset or absolute destination which has been previously loaded into a register. In this case the PC should be manually saved in R14 if a Branch with Link type operation is required.

5.3.1 The link bit

Branch with Link (BL) writes the old PC into the link register (R14) of the current bank. The PC value written into R14 is adjusted to allow for the prefetch, and contains the address of the instruction following the branch and link instruction. Note that the CPSR is not saved with the PC.

To return from a routine called by Branch with Link use `MOV PC,R14` if the link register is still valid or `LDM Rn!,{..PC}` if the link register has been saved onto a stack pointed to by Rn.

5.3.2 Instruction cycle times

Branch and Branch with Link instructions take 3 instruction fetches. For more information see *5.17 Instruction Speed Summary* on page 5-52.

5.3.3 Assembler syntax

```
B{L}{cond} <expression>
```

`{L}` is used to request the Branch with Link form of the instruction. If absent, R14 will not be affected by the instruction.

`{cond}` is a two-character mnemonic as shown in ◆*Figure 5-2: Condition codes* on page 5-3 (EQ, NE, VS etc). If absent then AL (ALways) will be used.

`<expression>` is the destination. The assembler calculates the offset.

Items in {} are optional. Items in <> must be present.

5.3.4 Examples

```
here  BAL   here     ; assembles to 0xEAFFFFFE (note effect
      B     there      of PC offset) ALways condition used as
                       default

      CMP   R1,#0    ; compare R1 with zero and branch to fred
      BEQ   fred       if R1 was zero otherwise continue to
                     ; next instruction

      BL    sub+ROM  ; call subroutine at computed address

      ADDS  R1,#1    ; add 1 to register 1, setting CPSR flags
      BLCC  sub      ; on the result then call subroutine if
                     ; the C flag is clear,which will be the
                     ; case unless R1 held 0xFFFFFFFF
```

5.4 Data Processing

The instruction is only executed if the condition is true, defined at the beginning of this chapter. The instruction encoding is shown in *Figure 5-4: Data processing instructions* on page 5-7.

The instruction produces a result by performing a specified arithmetic or logical operation on one or two operands. The first operand is always a register (Rn). The second operand may be a shifted register (Rm) or a rotated 8-bit immediate value (Imm) according to the value of the I bit in the instruction. The condition codes in the CPSR may be preserved or updated as a result of this instruction, according to the value of the S bit in the instruction. Certain operations (TST, TEQ, CMP, CMN) do not write the result to Rd. They are used only to perform tests and to set the condition codes on the result and always have the S bit set. The instructions and their effects are listed in *Table 5-1: ARM data processing instructions* on page 5-8

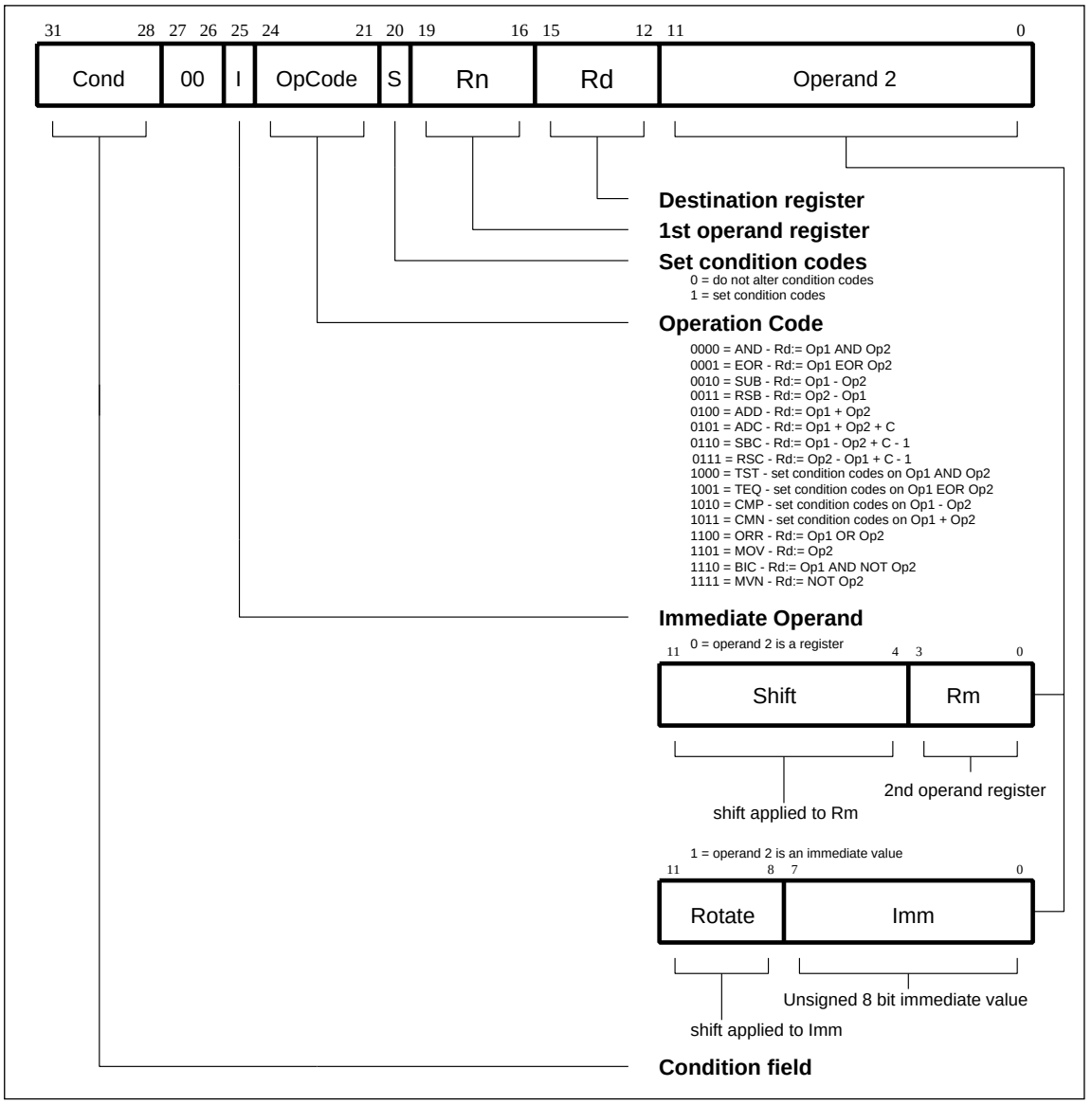


Figure 5-4: Data processing instructions

5.4.1 CPSR flags

The data processing operations may be classified as logical or arithmetic. The logical operations (AND, EOR, TST, TEQ, ORR, MOV, BIC, MVN) perform the logical action on all corresponding bits of the operand or operands to produce the result. If the S bit is set (and Rd is not R15, see below) the V flag in the CPSR will be unaffected, the C flag will be set to the carry out from the barrel shifter (or preserved when the shift operation is LSL #0), the Z flag will be set if and only if the result is all zeros, and the N flag will be set to the logical value of bit 31 of the result.

Assembler Mnemonic	OpCode	Action
AND	0000	operand1 AND operand2
EOR	0001	operand1 EOR operand2
SUB	0010	operand1 - operand2
RSB	0011	operand2 - operand1
ADD	0100	operand1 + operand2
ADC	0101	operand1 + operand2 + carry
SBC	0110	operand1 - operand2 + carry - 1
RSC	0111	operand2 - operand1 + carry - 1
TST	1000	as AND, but result is not written
TEQ	1001	as EOR, but result is not written
CMP	1010	as SUB, but result is not written
CMN	1011	as ADD, but result is not written
ORR	1100	operand1 OR operand2
MOV	1101	operand2 (operand1 is ignored)
BIC	1110	operand1 AND NOT operand2 (Bit clear)
MVN	1111	NOT operand2 (operand1 is ignored)

Table 5-1: ARM data processing instructions

The arithmetic operations (SUB, RSB, ADD, ADC, SBC, RSC, CMP, CMN) treat each operand as a 32-bit integer (either unsigned or 2's complement signed, the two are equivalent). If the S bit is set (and Rd is not R15) the V flag in the CPSR will be set if an overflow occurs into bit 31 of the result; this may be ignored if the operands were considered unsigned, but warns of a possible error if the operands were 2's complement signed. The C flag will be set to the carry out of bit 31 of the ALU, the Z flag will be set if and only if the result was zero, and the N flag will be set to the value of bit 31 of the result (indicating a negative result if the operands are considered to be 2's complement signed).

5.4.2 Shifts

When the second operand is specified to be a shifted register, the operation of the barrel shifter is controlled by the Shift field in the instruction. This field indicates the type of shift to be performed (logical left or right, arithmetic right or rotate right). The amount by which the register should be shifted may be contained in an immediate field in the instruction, or in the bottom byte of another register (other than R15). The encoding for the different shift types is shown in ➢*Figure 5-5: ARM shift operations*.

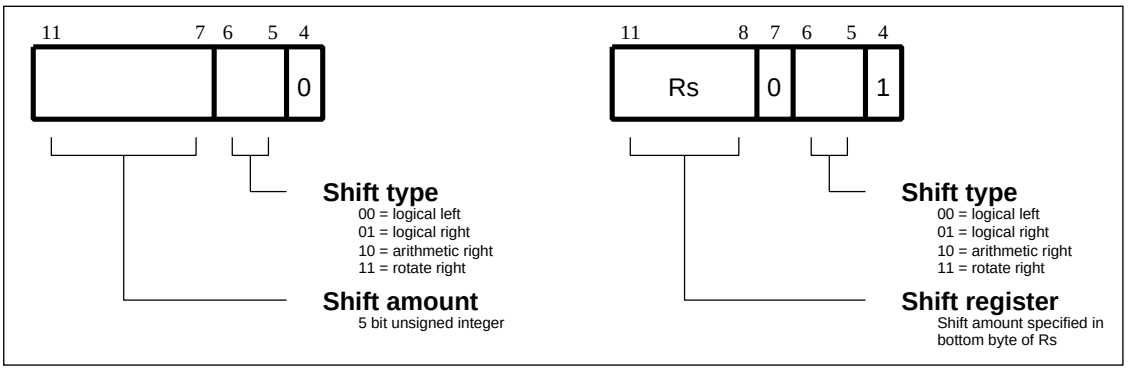


Figure 5-5: ARM shift operations

Instruction specified shift amount

When the shift amount is specified in the instruction, it is contained in a 5-bit field which may take any value from 0 to 31. A logical shift left (LSL) takes the contents of Rm and moves each bit by the specified amount to a more significant position. The least significant bits of the result are filled with zeros, and the high bits of Rm which do not map into the result are discarded, except that the least significant discarded bit becomes the shifter carry output which may be latched into the C bit of the CPSR when the ALU operation is in the logical class (see above). For example, the effect of LSL #5 is shown in ➢*Figure 5-6: Logical shift left*.

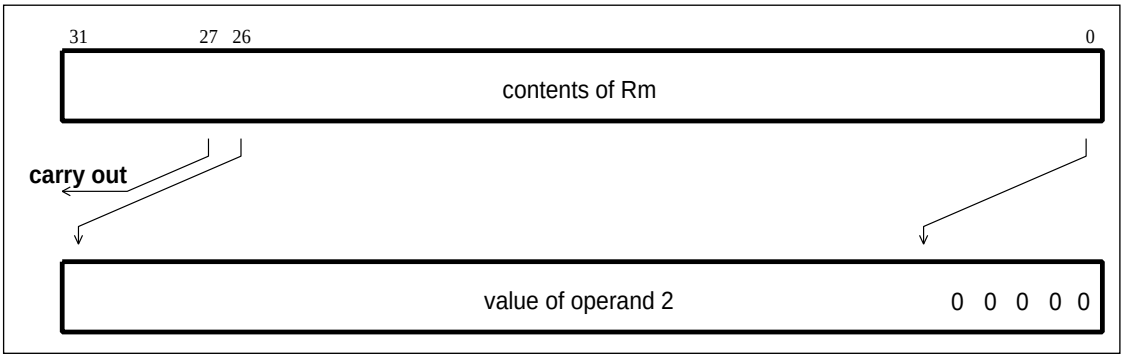


Figure 5-6: Logical shift left

Note that LSL #0 is a special case, where the shifter carry out is the old value of the CPSR C flag. The contents of Rm are used directly as the second operand.

A logical shift right (LSR) is similar, but the contents of Rm are moved to less significant positions in the result. LSR #5 has the effect shown in *Figure 5-7: Logical shift right* on page 5-10.

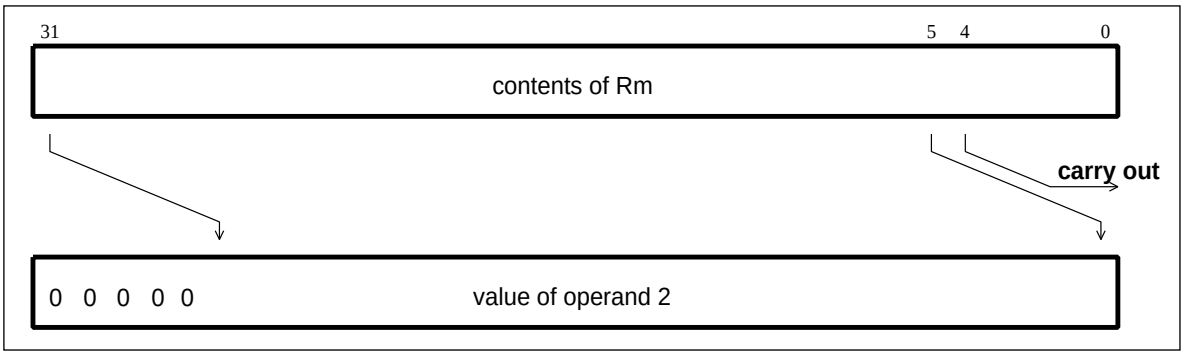


Figure 5-7: Logical shift right

The form of the shift field which might be expected to correspond to LSR #0 is used to encode LSR #32, which has a zero result with bit 31 of Rm as the carry output. Logical shift right zero is redundant as it is the same as logical shift left zero, so the assembler will convert LSR #0 (and ASR #0 and ROR #0) into LSL #0, and allow LSR #32 to be specified.

An arithmetic shift right (ASR) is similar to logical shift right, except that the high bits are filled with bit 31 of Rm instead of zeros. This preserves the sign in 2's complement notation. For example, ASR #5 is shown in *Figure 5-8: Arithmetic shift right*.

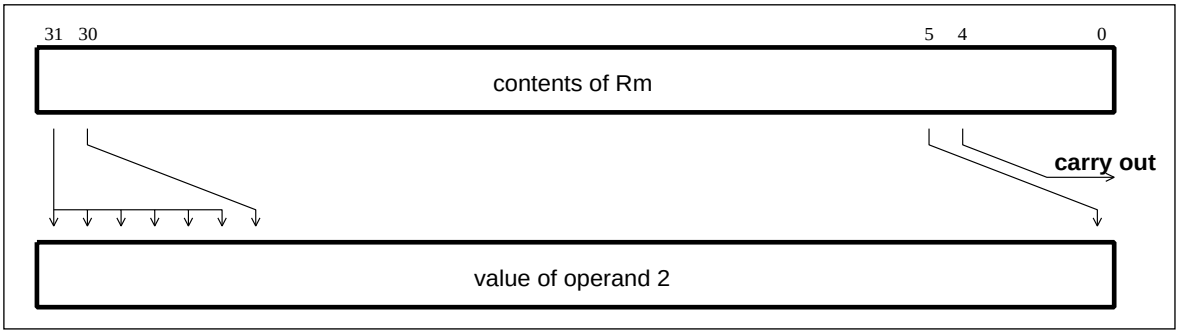


Figure 5-8: Arithmetic shift right

The form of the shift field which might be expected to give ASR #0 is used to encode ASR #32. Bit 31 of Rm is again used as the carry output, and each bit of operand 2 is also equal to bit 31 of Rm. The result is therefore all ones or all zeros, according to the value of bit 31 of Rm.

Rotate right (ROR) operations reuse the bits which 'overshoot' in a logical shift right operation by reintroducing them at the high end of the result, in place of the zeros used to fill the high end in logical right operations. For example, ROR #5 is shown in *Figure 5-9: Rotate right*.

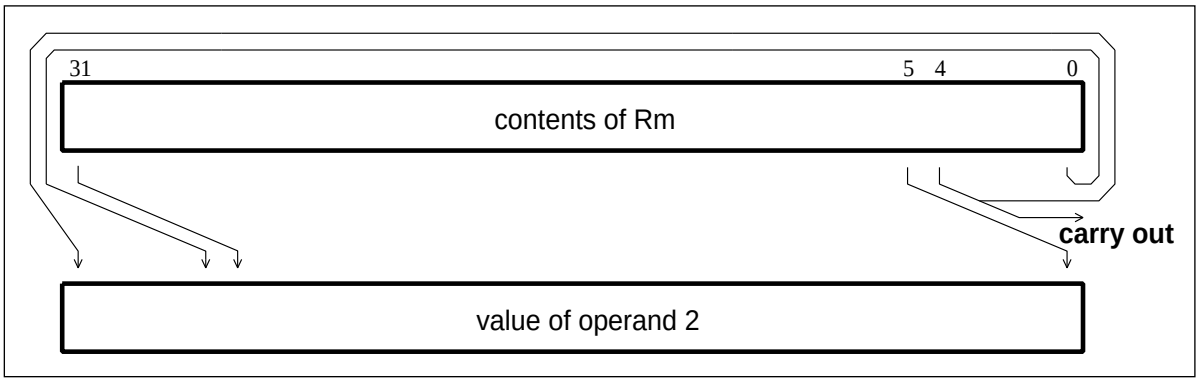


Figure 5-9: Rotate right

The form of the shift field which might be expected to give ROR #0 is used to encode a special function of the barrel shifter, rotate right extended (RRX). This is a rotate right by one bit position of the 33-bit quantity formed by appending the CPSR C flag to the most significant end of the contents of Rm as shown in ➲*Figure 5-10: Rotate right extended*.

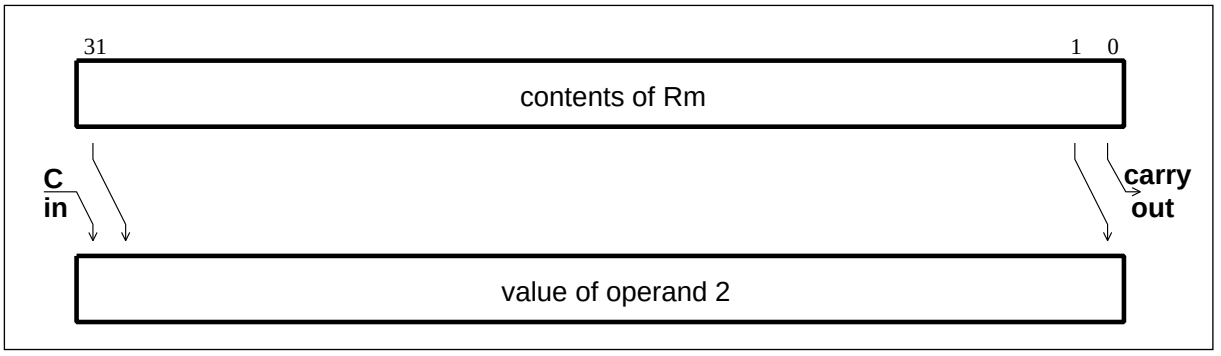


Figure 5-10: Rotate right extended

Register specified shift amount

Only the least significant byte of the contents of Rs is used to determine the shift amount. Rs can be any general register other than R15.

If this byte is zero, the unchanged contents of Rm will be used as the second operand, and the old value of the CPSR C flag will be passed on as the shifter carry output.

If the byte has a value between 1 and 31, the shifted result will exactly match that of an instruction specified shift with the same value and shift operation.

If the value in the byte is 32 or more, the result will be a logical extension of the shift described above:

1 LSL by 32 has result zero, carry out equal to bit 0 of Rm.

2 LSL by more than 32 has result zero, carry out zero.

3 LSR by 32 has result zero, carry out equal to bit 31 of Rm.

4 LSR by more than 32 has result zero, carry out zero.

5 ASR by 32 or more has result filled with and carry out equal to bit 31 of Rm.

6 ROR by 32 has result equal to Rm, carry out equal to bit 31 of Rm.

7 ROR by n where n is greater than 32 will give the same result and carry out as ROR by n-32; therefore repeatedly subtract 32 from n until the amount is in the range 1 to 32 and see above.

Note that the zero in bit 7 of an instruction with a register controlled shift is compulsory; a one in this bit will cause the instruction to be a multiply or undefined instruction.

5.4.3 Immediate operand rotates

The immediate operand rotate field is a 4-bit unsigned integer which specifies a shift operation on the 8-bit immediate value. This value is zero extended to 32 bits, and then subject to a rotate right by twice the value in the rotate field. This enables many common constants to be generated, for example all powers of 2.

5.4.4 Writing to R15

When Rd is a register other than R15, the condition code flags in the CPSR may be updated from the ALU flags as described above.

When Rd is R15 and the S flag in the instruction is not set the result of the operation is placed in R15 and the CPSR is unaffected.

When Rd is R15 and the S flag is set the result of the operation is placed in R15 and the SPSR corresponding to the current mode is moved to the CPSR. This allows state changes which atomically restore both PC and CPSR. This form of instruction must not be used in User mode.

5.4.5 Using R15 as an operand

If R15 (the PC) is used as an operand in a data processing instruction the register is used directly.

The PC value will be the address of the instruction, plus 8 or 12 bytes due to instruction prefetching. If the shift amount is specified in the instruction, the PC will be 8 bytes ahead. If a register is used to specify the shift amount the PC will be 12 bytes ahead.

5.4.6 TEQ, TST, CMP and CMN opcodes

These instructions do not write the result of their operation but do set flags in the CPSR. An assembler must always set the S flag for these instructions even if it is not specified in the mnemonic.

The TEQP form of the instruction used in earlier processors must not be used in the 32-bit modes, the PSR transfer operations should be used instead. If used in these modes, its effect is to move SPSR_<mode> to CPSR if the processor is in a privileged mode and to do nothing if in User mode.

5.4.7 Instruction cycle times

Data Processing instructions vary in the number of incremental cycles taken as follows:

Normal Data Processing	1instruction fetch
Data Processing with register specified shift	1 instruction fetch + 1 internal cycle
Data Processing with PC written	3 instruction fetches
Data Processing with register specified shift and PC written	3 instruction fetches and 1 internal cycle

See *5.17 Instruction Speed Summary* on page 5-52 for more information.

5.4.8 Assembler syntax

1 MOV,MVN - single operand instructions
`<opcode>{cond}{S} Rd,<Op2>`

2 CMP,CMN,TEQ,TST - instructions which do not produce a result.
`<opcode>{cond} Rn,<Op2>`

3 AND,EOR,SUB,RSB,ADD,ADC,SBC,RSC,ORR,BIC
`<opcode>{cond}{S} Rd,Rn,<Op2>`

where

`<Op2>` is Rm{,<shift>} or,<#expression>

`{cond}` is a two-character condition mnemonic, see *Figure 5-2: Condition codes* on page 5-3

`{S}` sets condition codes if S present (implied for CMP, CMN, TEQ, TST).

`Rd,Rn,Rm` are expressions evaluating to a register number.

If `<#expression>` is used, the assembler will attempt to generate a shifted immediate 8-bit field to match the expression. If this is impossible, it will give an error.

`<shift>` is `<shiftname> <register>` or `<shiftname> #expression`, or RRX (rotate right one bit with extend).

`<shiftname>`s are: ASL, LSL, LSR, ASR, ROR. (ASL is a synonym for LSL, they assemble to the same code.)

5.4.9 Examples

```
ADDEQ R2,R4,R5          ; if the Z flag is set make R2:=R4+R5

TEQS  R4,#3             ; test R4 for equality with 3
                        ; (the S is in fact redundant as the
                        ; assembler inserts it automatically)

SUB   R4,R5,R7,LSR R2   ; logical right shift R7 by the number
                        ; in the bottom byte of R2, subtract
                        ; result from R5, and put the answer
                        ; into R4

MOV   PC,R14            ; return from subroutine

MOVS  PC,R14            ; return from exception and restore
                        ; CPSR from SPSR_mode
```

5.5 PSR Transfer (MRS, MSR)

The instruction is only executed if the condition is true. The various conditions are defined at the beginning of this chapter.

The MRS and MSR instructions are formed from a subset of the Data Processing operations and are implemented using the TEQ, TST, CMN and CMP instructions without the S flag set. The encoding is shown in *Figure 5-11: PSR Transfer* on page 5-16.

These instructions allow access to the CPSR and SPSR registers. The MRS instruction allows the contents of the CPSR or SPSR_<mode> to be moved to a general register. The MSR instruction allows the contents of a general register to be moved to the CPSR or SPSR_<mode> register.

The MSR instruction also allows an immediate value or register contents to be transferred to the condition code flags (N,Z,C and V) of CPSR or SPSR_<mode> without affecting the control bits. In this case, the top four bits of the specified register contents or 32-bit immediate value are written to the top four bits of the relevant PSR.

5.5.1 Operand restrictions

In User mode, the control bits of the CPSR are protected from change, so only the condition code flags of the CPSR can be changed. In other (privileged) modes the entire CPSR can be changed.

The SPSR register which is accessed depends on the mode at the time of execution. For example, only SPSR_fiq is accessible when the processor is in FIQ mode.

R15 must not be specified as the source or destination register.

A further restriction is that no attempt should be made to access an SPSR in User mode, since no such register exists.

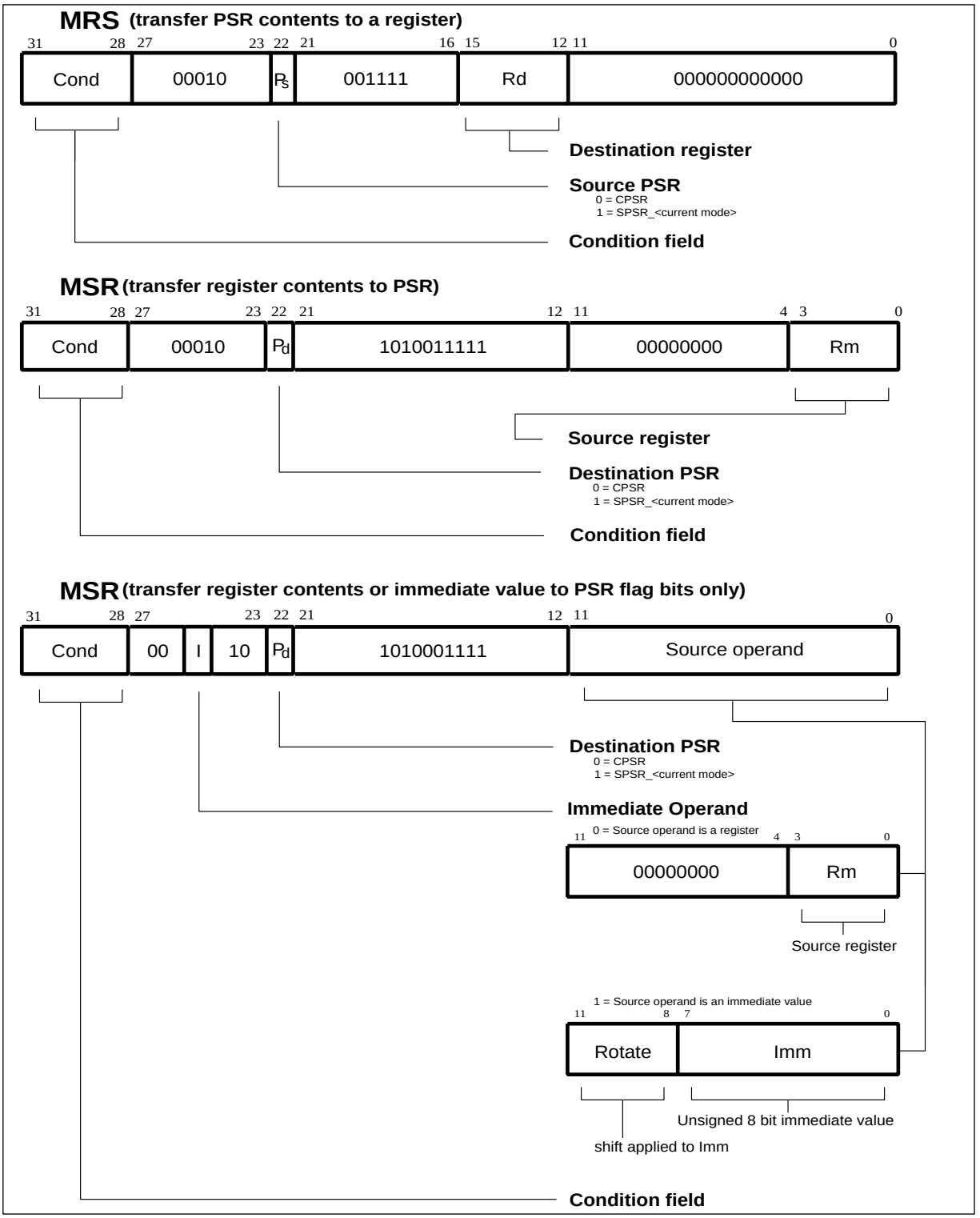


Figure 5-11: PSR Transfer

5.5.2 Reserved bits

Only eleven bits of the PSR are defined in ARM processor (N,Z,C,V,I,F and M[4:0]); the remaining bits (PSR[27:8,5]) are reserved for use in future versions of the processor. To ensure the maximum compatibility between ARM processor programs and future processors, the following rules should be observed:

1. The reserved bits must be preserved when changing the value in a PSR.
2. Programs must not rely on specific values from the reserved bits when checking the PSR status, since they may read as one or zero in future processors.

A read-modify-write strategy should therefore be used when altering the control bits of any PSR register; this involves transferring the appropriate PSR register to a general register using the MRS instruction, changing only the relevant bits and then transferring the modified value back to the PSR register using the MSR instruction.

For example, the following sequence performs a mode change:

```
MRS  R0,CPSR         ; take a copy of the CPSR
BIC  R0,R0,#0x1F     ; clear the mode bits
ORR  R0,R0,#new_mode ; select new mode
MSR  CPSR,R0         ; write back the modified CPSR
```

When the aim is simply to change the condition code flags in a PSR, a value can be written directly to the flag bits without disturbing the control bits. For example, the following instruction sets the N,Z,C and V flags:

```
MSR  CPSR_flg,#0xF0000000; set all the flags regardless of
                          ; their previous state (does not
                          ; affect any control bits)
```

No attempt should be made to write an 8-bit immediate value into the whole PSR since such an operation cannot preserve the reserved bits.

5.5.3 Instruction cycle times

PSR Transfers take 1 instruction fetch. For more information see *5.17 Instruction Speed Summary* on page 5-52.

5.5.4 Assembler syntax

1. MRS - transfer PSR contents to a register

   `MRS{cond} Rd,<psr>`
2. MSR - transfer register contents to PSR

   `MSR{cond} <psr>,Rm`
3. MSR - transfer register contents to PSR flag bits only

   `MSR{cond} <psrf>,Rm`

   The most significant four bits of the register contents are written to the N,Z,C and V flags respectively.

4 MSR - transfer immediate value to PSR flag bits only

`MSR{cond} <psrf>,<#expression>`

The expression should symbolise a 32-bit value of which the most significant four bits are written to the N,Z,C and V flags respectively.

`{cond}` two-character condition mnemonic, see *Figure 5-2: Condition codes* on page 5-3

`Rd, Rm` expressions evaluating to a register number other than R15

`<psr>` CPSR, CPSR_all, SPSR or SPSR_all. (CPSR and CPSR_all are synonyms as are SPSR and SPSR_all)

`<psrf>` CPSR_flg or SPSR_flg

Where <#expression> is used, the assembler will attempt to generate a shifted immediate 8-bit field to match the expression. If this is impossible, it will give an error.

5.5.5 Examples

In User mode the instructions behave as follows:

```
MSR    CPSR_all,Rm          ; CPSR[31:28] <- Rm[31:28]
MSR    CPSR_flg,Rm          ; CPSR[31:28] <- Rm[31:28]

MSR    CPSR_flg,#0xA0000000; CPSR[31:28] <- 0xA
                            ; (i.e. set N,C; clear Z,V)

MRS    Rd,CPSR              ; Rd[31:0] <- CPSR[31:0]
```

In privileged modes the instructions behave as follows:

```
MSR    CPSR_all,Rm          ; CPSR[31:0]  <- Rm[31:0]
MSR    CPSR_flg,Rm          ; CPSR[31:28] <- Rm[31:28]

MSR    CPSR_flg,#0x50000000; CPSR[31:28] <- 0x5
                            ; (i.e. set Z,V; clear N,C)

MRS    Rd,CPSR              ; Rd[31:0] <- CPSR[31:0]

MSR    SPSR_all,Rm          ; SPSR_<mode>[31:0]  <- Rm[31:0]
MSR    SPSR_flg,Rm          ; SPSR_<mode>[31:28] <- Rm[31:28]

MSR    SPSR_flg,#0xC0000000; SPSR_<mode>[31:28] <- 0xC
                            ; (i.e. set N,Z; clear C,V)

MRS    Rd,SPSR              ; Rd[31:0] <- SPSR_<mode>[31:0]
```

5.6 Multiply and Multiply-Accumulate (MUL, MLA)

The instruction is only executed if the condition is true. The various conditions are defined at the beginning of this chapter. The instruction encoding is shown in ↻*Figure 5-12: Multiply instructions*.

The multiply and multiply-accumulate instructions use an 2-bit Booth's algorithm to perform integer multiplication. They give the least significant 32 bits of the product of two 32-bit operands, and may be used to synthesize higher precision multiplications.

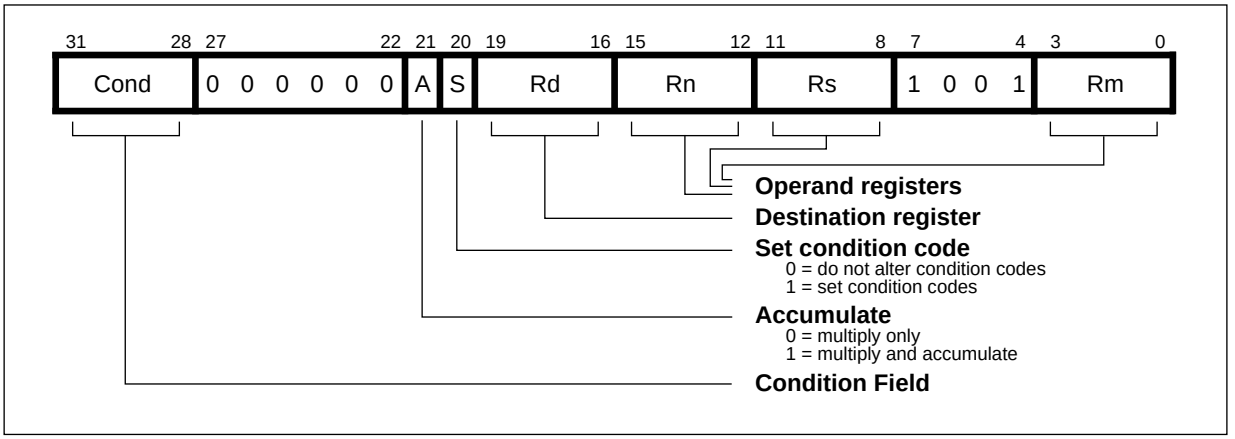


Figure 5-12: Multiply instructions

The multiply form of the instruction gives Rd:=Rm*Rs. Rn is ignored, and should be set to zero for compatibility with possible future upgrades to the instruction set.

The multiply-accumulate form gives Rd:=Rm*Rs+Rn, which can save an explicit ADD instruction in some circumstances.

Both forms of the instruction work on operands which may be considered as signed (2's complement) or unsigned integers.

5.6.1 Operand restrictions

Due to the way multiplication was implemented, certain combinations of operand registers should be avoided. (The assembler will issue a warning if these restrictions are overlooked.)

The destination register (Rd) should not be the same as the operand register (Rm), as Rd is used to hold intermediate values and Rm is used repeatedly during multiply. A MUL will give a zero result if Rm=Rd, and an MLA will give a meaningless result. R15 must not be used as an operand or as the destination register.

All other register combinations will give correct results, and Rd, Rn and Rs may use the same register when required.

5.6.2 CPSR flags

Setting the CPSR flags is optional, and is controlled by the S bit in the instruction. The N (Negative) and Z (Zero) flags are set correctly on the result (N is made equal to bit 31 of the result, and Z is set if and only if the result is zero). The C (Carry) flag is set to a meaningless value and the V (oVerflow) flag is unaffected.

5.6.3 Instruction cycle times

The Multiply instructions take 1 instruction fetch and m internal cycles. For more information see ➢*5.17 Instruction Speed Summary* on page 5-52.

m is the number of cycles required by the multiply algorithm, which is determined by the contents of Rs. Multiplication by any number between *2^(2m-3)* and *2^(2m-1)-1* takes 1S+mI cycles for *1<m>16*. Multiplication by 0 or 1 takes 1S+1I cycles, and multiplication by any number greater than or equal to 2^(29) takes 1S+16I cycles. The maximum time for any multiply is thus 1S+16I cycles.

5.6.4 Assembler syntax

```
MUL{cond}{S} Rd,Rm,Rs
MLA{cond}{S} Rd,Rm,Rs,Rn
```

`{cond}` two-character condition mnemonic, see ➢*Figure 5-2: Condition codes* on page 5-3

`{S}` set condition codes if S present

`Rd, Rm, Rs` and `Rn`

expressions evaluating to a register number other than R15.

5.6.5 Examples

```
MUL         R1,R2,R3    ; R1:=R2*R3
MLAEQS      R1,R2,R3,R4 ; conditionally R1:=R2*R3+R4,
                        ; setting condition codes
```

5.7 Single Data Transfer (LDR, STR)

The instruction is only executed if the condition is true. The various conditions are defined at the beginning of this chapter. The instruction encoding is shown in *Figure 5-13: Single data transfer instructions* on page 5-21.

The single data transfer instructions are used to load or store single bytes or words of data. The memory address used in the transfer is calculated by adding an offset to or subtracting an offset from a base register. The result of this calculation may be written back into the base register if `auto-indexing' is required.

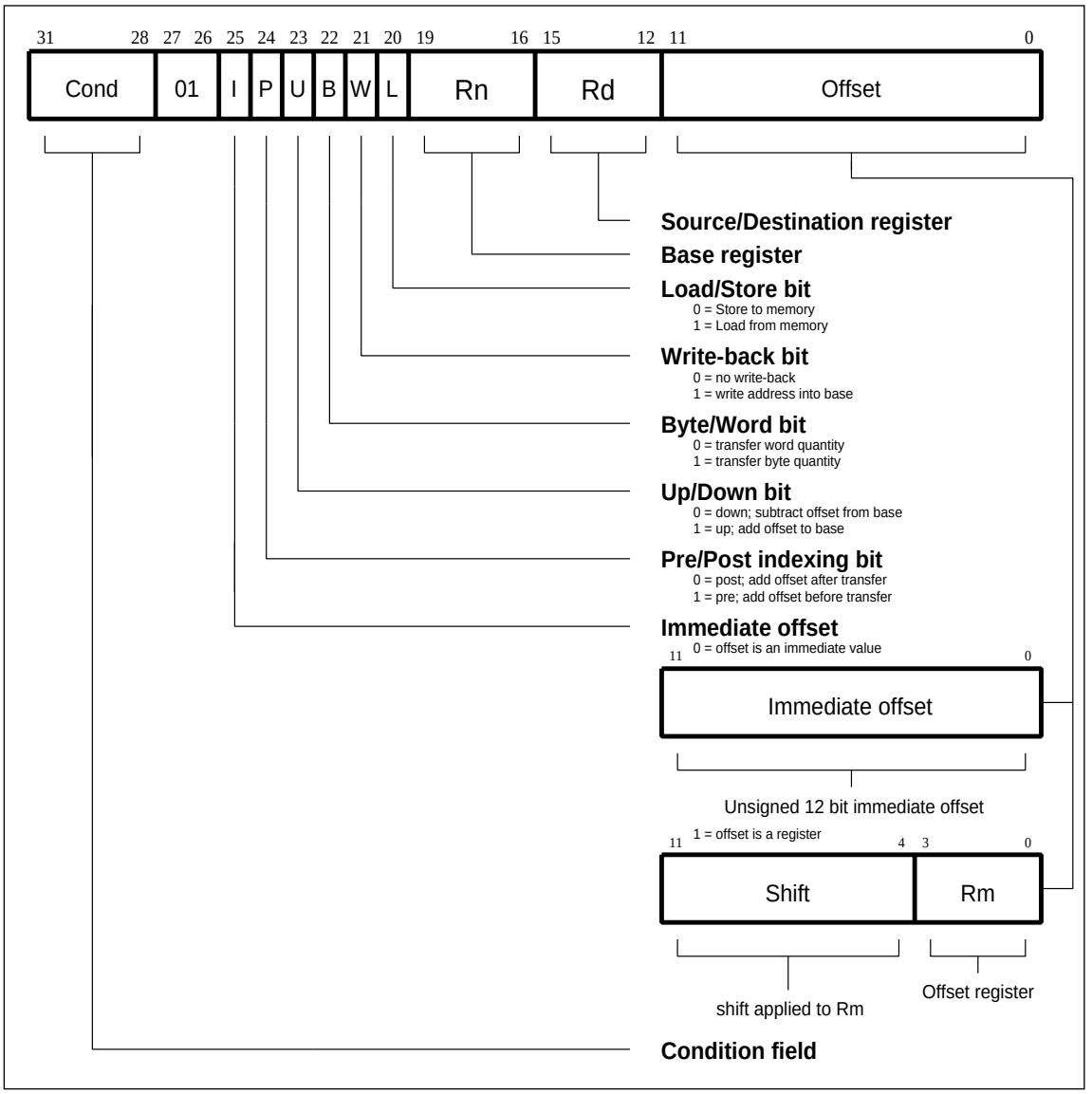


Figure 5-13: Single data transfer instructions

5.7.1 Offsets and auto-indexing

The offset from the base may be either a 12-bit unsigned binary immediate value in the instruction, or a second register (possibly shifted in some way). The offset may be added to (U=1) or subtracted from (U=0) the base register Rn. The offset modification may be performed either before (pre-indexed, P=1) or after (post-indexed, P=0) the base is used as the transfer address.

The W bit gives optional auto increment and decrement addressing modes. The modified base value may be written back into the base (W=1), or the old base value may be kept (W=0). In the case of post-indexed addressing, the write back bit is redundant and is always set to zero, since the old base value can be retained by setting the offset to zero. Therefore post-indexed data transfers always write back the modified base. The only use of the W bit in a post-indexed data transfer is in privileged mode code, where setting the W bit forces non-privileged mode for the transfer, allowing the operating system to generate a user address in a system where the memory management hardware makes suitable use of this hardware.

5.7.2 Shifted register offset

The 8 shift control bits are described in the data processing instructions section. However, the register specified shift amounts are not available in this instruction class. See *5.4.2 Shifts* on page 5-9.

5.7.3 Bytes and words

This instruction class may be used to transfer a byte (B=1) or a word (B=0) between an ARM processor register and memory.

The action of LDR(B) and STR(B) instructions is influenced by the 3 instruction fetches. For more information see *5.17 Instruction Speed Summary* on page 5-52. The two possible configurations are described below.

Little-endian configuration

A byte load (LDRB) expects the data on data bus inputs 7 through 0 if the supplied address is on a word boundary, on data bus inputs 15 through 8 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bits of the destination register, and the remaining bits of the register are filled with zeros. Please see *Figure 4-2: Big-endian addresses of bytes within words* on page 4-3.

A byte store (STRB) repeats the bottom 8 bits of the source register four times across data bus outputs 31 through 0. The external memory system should activate the appropriate byte subsystem to store the data.

A word load (LDR) will normally use a word aligned address. However, an address offset from a word boundary will cause the data to be rotated into the register so that the addressed byte occupies bits 0 to 7. This means that half-words accessed at offsets 0 and 2 from the word boundary will be correctly loaded into

bits 0 through 15 of the register. Two shift operations are then required to clear or to sign extend the upper 16 bits. This is illustrated in *Figure 5-14: Little-endian offset addressing* on page 5-23.

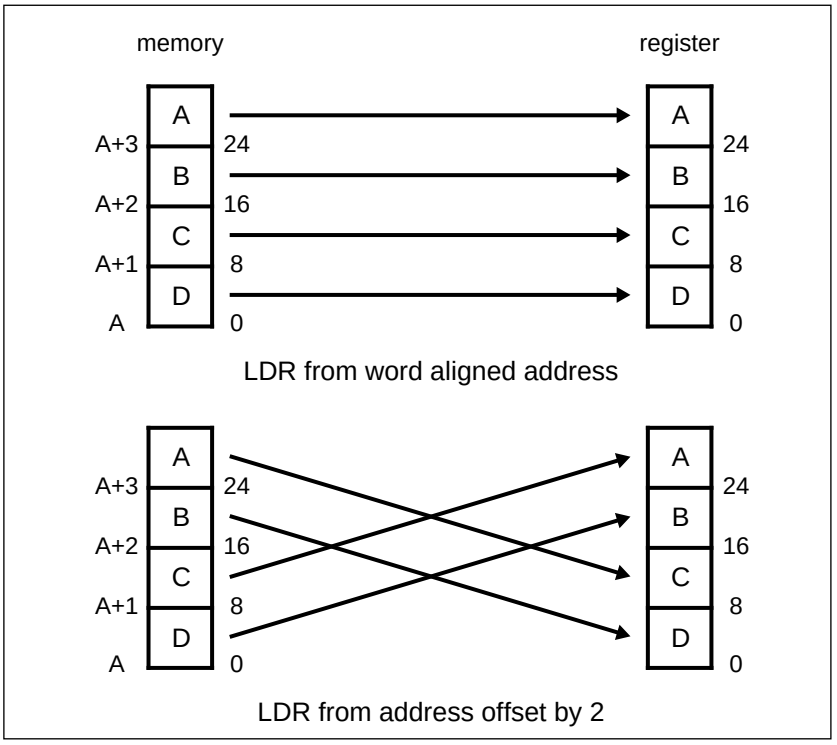


Figure 5-14: Little-endian offset addressing

A word store (STR) should generate a word aligned address. The word presented to the data bus is not affected if the address is not word aligned. That is, bit 31 of the register being stored always appears on data bus output 31.

Big-endian configuration

A byte load (LDRB) expects the data on data bus inputs 31 through 24 if the supplied address is on a word boundary, on data bus inputs 23 through 16 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bits of the destination register and the remaining bits of the register are filled with zeros. Please see *Figure 4-2: Big-endian addresses of bytes within words* on page 4-3.

A byte store (STRB) repeats the bottom 8 bits of the source register four times across data bus outputs 31 through 0. The external memory system should activate the appropriate byte subsystem to store the data.

A word load (LDR) should generate a word-aligned address. An address offset of 0 or 2 from a word boundary will cause the data to be

rotated into the register so that the addressed byte occupies bits 31 through 24. This means that half-words accessed at these offsets will be correctly loaded into bits 16 through 31 of the register. A shift operation is then required to move (and optionally sign extend) the data into the bottom 16 bits. An address offset of 1 or 3 from a word boundary will cause the data to be rotated into the register so that the addressed byte occupies bits 15 through 8.

A word store (STR) should generate a word aligned address. The word presented to the data bus is not affected if the address is not word aligned. That is, bit 31 of the register being stored always appears on data bus output 31.

5.7.4 Use of R15

Writeback must not be specified if R15 is specified as the base register (Rn). When using R15 as the base register you must remember it contains an address 8 bytes on from the address of the current instruction.

R15 must not be specified as the register offset (Rm).

When R15 is the source register (Rd) of a register store (STR) instruction, the stored value will be address of the instruction plus 12.

5.7.5 Restriction on the use of base register

When configured for late aborts, the following example code is difficult to unwind as the base register, Rn, gets updated before the abort handler starts. Sometimes it may be impossible to calculate the initial value.

For example:

```
LDR   R0,[R1],R1
```

So a post-indexed LDR/STR where Rm is the same register as Rn must not be used.

5.7.6 Data aborts

A transfer to or from a legal address may cause problems for a memory management system. For instance, in a system which uses virtual memory the required data may be absent from main memory. The memory manager can signal a problem by taking the processor abort input HIGH whereupon the Data Abort trap will be taken. It is up to the system software to resolve the cause of the problem, then the instruction can be restarted and the original program continued.

5.7.7 Instruction cycle times

Normal LDR instructions take 1 instruction fetch, 1 data read and 1 internal cycle and LDR PC take 3 instruction fetches, 1 data read and 1 internal cycle. For more information see *5.17 Instruction Speed Summary* on page 5-52.

STR instructions take 1 instruction fetch and 1 data write incremental cycles to execute.

5.7.8 Assembler syntax

```
<LDR|STR>{cond}{B}{T} Rd,<Address>
```

`LDR` load from memory into a register

`STR` store from a register into memory

`{cond}` two-character condition mnemonic, see *Figure 5-2: Condition codes* on page 5-3

`{B}` if B is present then byte transfer, otherwise word transfer

`{T}` if T is present the W bit will be set in a post-indexed instruction, forcing non-privileged mode for the transfer cycle. T is not allowed when a pre-indexed addressing mode is specified or implied.

`Rd` an expression evaluating to a valid register number.

<Address> can be:

1 An expression which generates an address:

`<expression>`

The assembler will attempt to generate an instruction using the PC as a base and a corrected immediate offset to address the location given by evaluating the expression. This will be a PC relative, pre-indexed address. If the address is out of range, an error will be generated.

2 A pre-indexed addressing specification:

`[Rn]` offset of zero

`[Rn,<#expression>]{!}` offset of <expression> bytes

`[Rn,{+/-}Rm{,<shift>}]{!}` offset of +/- contents of index register, shifted by <shift>

3 A post-indexed addressing specification:

`[Rn],<#expression>` offset of <expression> bytes

`[Rn],{+/-}Rm{,<shift>}` offset of +/- contents of index register, shifted as by <shift>.

`Rn, Rm` expressions evaluating to a register number. If Rn is R15 then the assembler will subtract 8 from the offset value to allow for ARM7100 pipelining. In this case base writeback must not be specified.

`<shift>` a general shift operation (see section on data processing instructions) but note that the shift amount may not be specified by a register.

`{!}` writes back the base register (set the W bit) if ! is present.

5.7.9 Examples

```
STR    R1,[R2,R4]!; store R1 at R2+R4 (both of which are
              ; registers) and write back address to R2

STR    R1,[R2],R4; store R1 at R2 and write back
              ; R2+R4 to R2

LDR    R1,[R2,#16]; load R1 from contents of R2+16
              ;  Don't write back

LDR    R1,[R2,R3,LSL#2]; load R1 from contents of R2+R3*4

LDREQB R1,[R6,#5]; conditionally load byte at R6+5 into
              ;  R1 bits 0 to 7, filling bits 8 to 31
              ;  with zeros

STR    R1,PLACE; generate PC relative offset to address
       •      ;  PLACE
       •
PLACE
```

5.8 Block Data Transfer (LDM, STM)

The instruction is only executed if the condition is true. The various conditions are defined at the beginning of this chapter. The instruction encoding is shown in *Figure 5-15: Block data transfer instructions* on page 5-27.

Block data transfer instructions are used to load (LDM) or store (STM) any subset of the currently visible registers. They support all possible stacking modes, maintaining full or empty stacks which can grow up or down memory, and are very efficient instructions for saving or restoring context, or for moving large blocks of data around main memory.

5.8.1 The register list

The instruction can cause the transfer of any registers in the current bank (and non-user mode programs can also transfer to and from the user bank, see below). The register list is a 16-bit field in the instruction, with each bit corresponding to a register. A 1 in bit 0 of the register field will cause R0 to be transferred, a 0 will cause it not to be transferred; similarly bit 1 controls the transfer of R1, and so on.

Any subset of the registers, or all the registers, may be specified. The only restriction is that the register list should not be empty.

Whenever R15 is stored to memory the stored value is the address of the STM instruction plus 12.

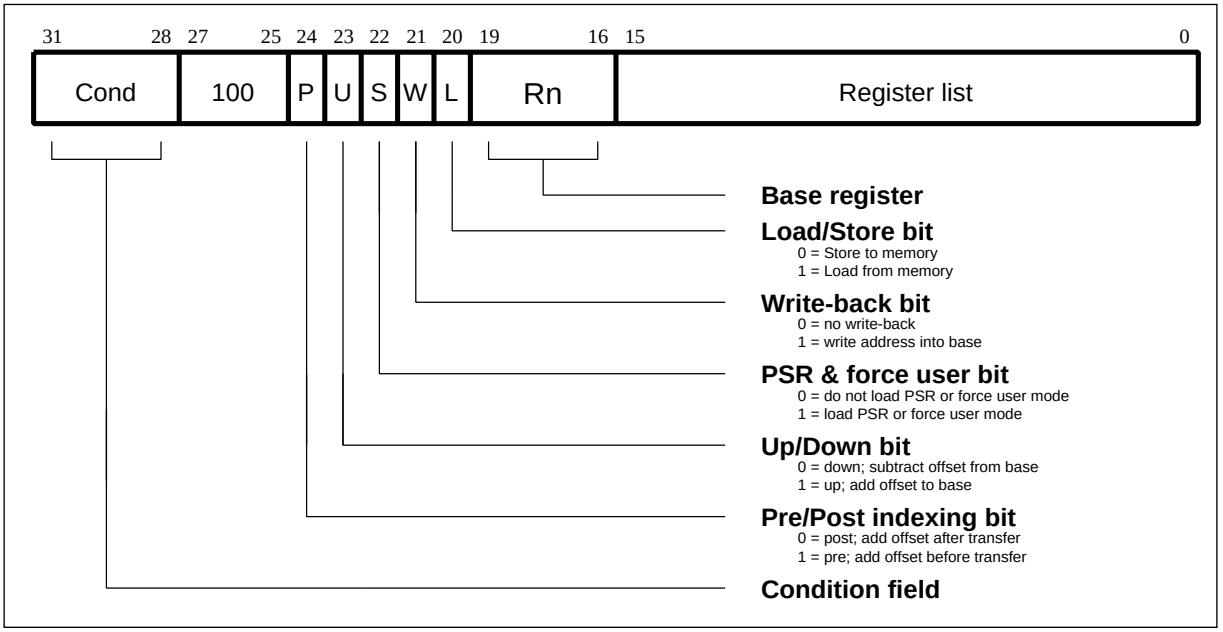


Figure 5-15: Block data transfer instructions

5.8.2 Addressing modes

The transfer addresses are determined by the contents of the base register (Rn), the pre/post bit (P) and the up/down bit (U). The registers are transferred in the order lowest to highest, so R15 (if in the list) will always be transferred last. The lowest register also gets transferred to/from the lowest memory address. By way of illustration, consider the transfer of R1, R5 and R7 in the case where Rn=0x1000 and write back of the modified base is required (W=1). *Figure 5-16: Post-increment addressing* on page 5-28 to *Figure 5-19: Pre-decrement addressing* on page 5-30 show the sequence of register transfers, the addresses used, and the value of Rn after the instruction has completed.

In all cases, had write back of the modified base not been required (W=0), Rn would have retained its initial value of 0x1000 unless it was also in the transfer list of a load multiple register instruction, when it would have been overwritten with the loaded value.

5.8.3 Address alignment

The address should normally be a word aligned quantity and non-word aligned addresses do not affect the instruction. The bottom 2 address bits are ignored by the LDM instruction. No rotating of data will occur for an LDM from a non-aligned address. If this is required then a series of LDRs should be used instead. However, the bottom 2 bits of the address will appear on bits [1:0] at the address and might be interpreted by the memory system.

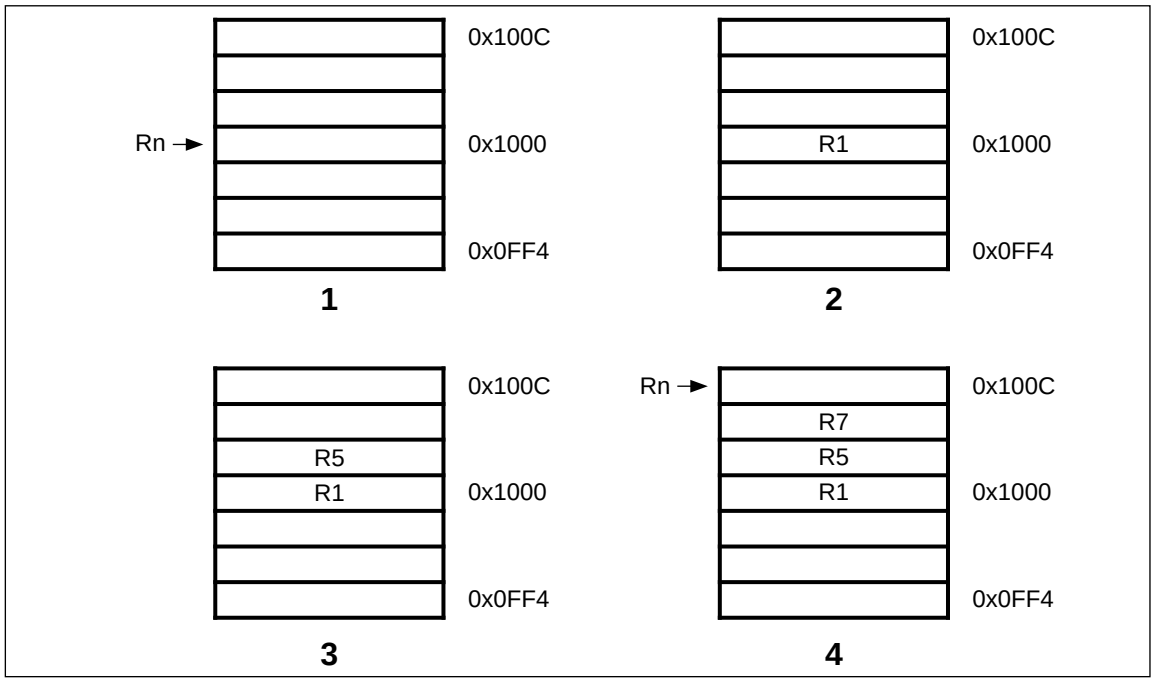


Figure 5-16: Post-increment addressing

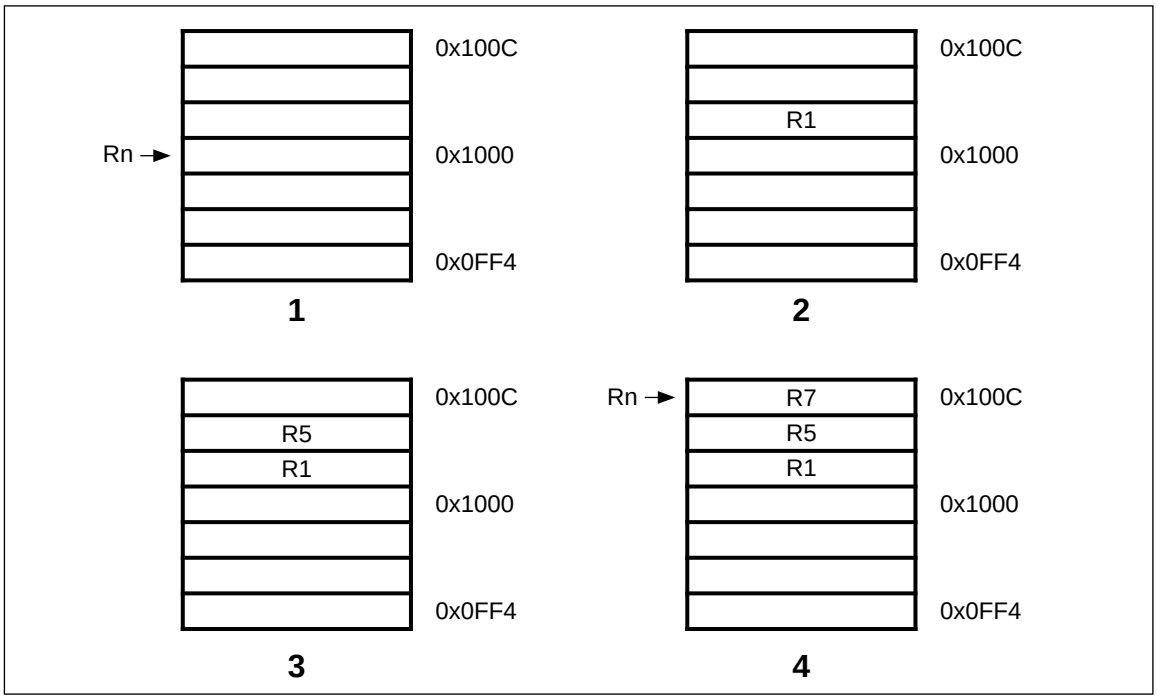


Figure 5-17: Pre-increment addressing

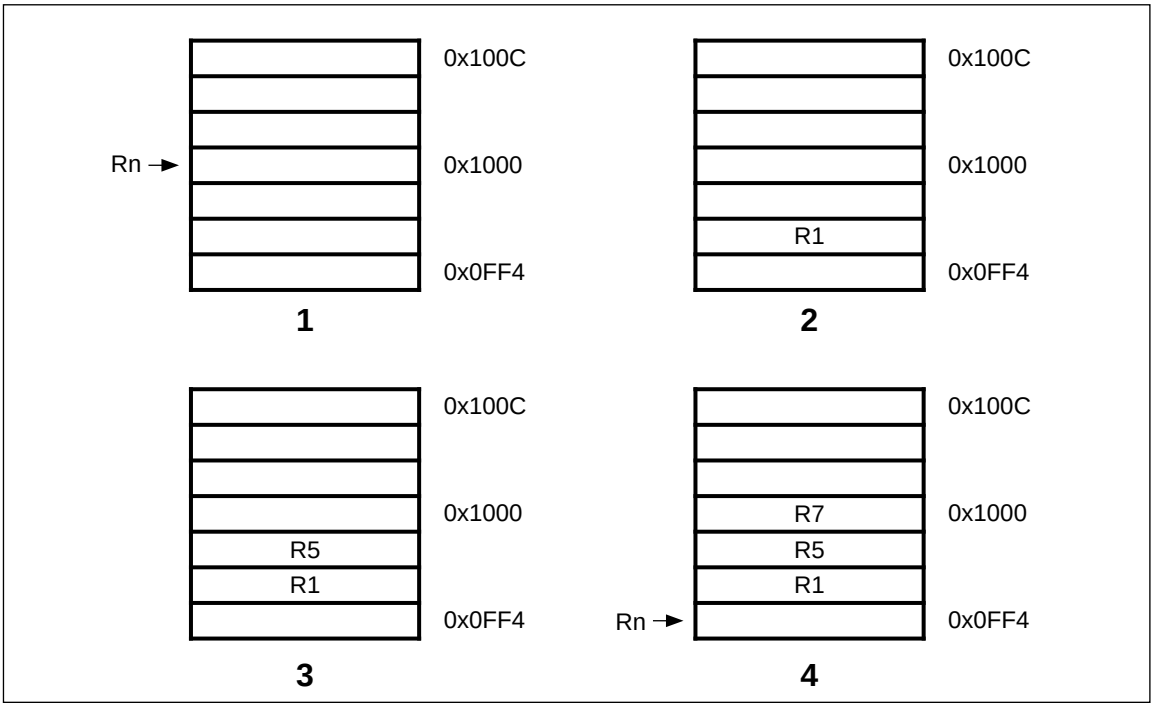


Figure 5-18: Post-decrement addressing

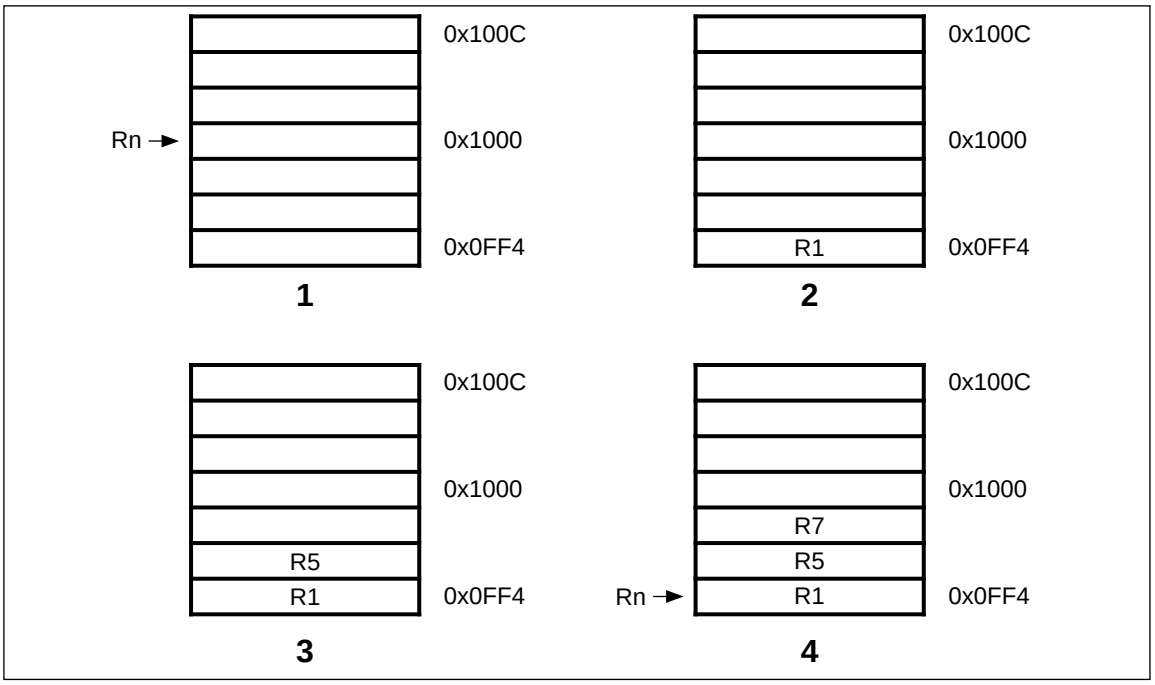


Figure 5-19: Pre-decrement addressing

5.8.4 Use of the S bit

When the S bit is set in a LDM/STM instruction its meaning depends on whether or not R15 is in the transfer list and on the type of instruction. The S bit should only be set if the instruction is to execute in a privileged mode.

LDM with R15 in transfer list and S bit set (Mode changes)

If the instruction is a LDM then SPSR_<mode> is transferred to CPSR at the same time as R15 is loaded.

STM with R15 in transfer list and S bit set (User bank transfer)

The registers transferred are taken from the User bank rather than the bank corresponding to the current mode. This is useful for saving the user state on process switches. Base writeback must not be used when this mechanism is employed.

R15 not in list and S bit set (User bank transfer)

For both LDM and STM instructions, the User bank registers are transferred rather than the register bank corresponding to the current mode. This is useful for saving the user state on process switches. Base writeback must not be used when this mechanism is employed.

When the instruction is LDM, care must be taken not to read from a banked register during the following cycle (inserting a dummy instruction such as MOV R0, R0 after the LDM will ensure safety).

5.8.5 Use of R15 as the base

R15 must not be used as the base register in any LDM or STM instruction.

5.8.6 Inclusion of the base in the register list

When writeback is specified, the base is written back at the end of the second cycle of the instruction. During a STM, the first register is written out at the start of the second cycle. A STM which includes storing the base, with the base as the first register to be stored, will therefore store the unchanged value, whereas with the base second or later in the transfer order, will store the modified value. A LDM will always overwrite the updated base if the base is in the list.

5.8.7 Data aborts

Some legal addresses may be unacceptable to a memory management system, and the memory manager can indicate a problem with an address by taking the abort input to the ARM7 CPU HIGH. This can happen on any transfer during a multiple register load or store, and must be recoverable if the processor is to be used in a virtual memory system.

Aborts during STM instructions

If the abort occurs during a store multiple instruction, the ARM processor takes little action until the instruction completes, whereupon it enters the data abort trap. The memory manager is responsible for preventing erroneous writes to the memory. The only change to the internal state of the processor will be the modification of the base register if writeback was specified, and this must be reversed by software (and the cause of the abort resolved) before the instruction may be retried.

Aborts during LDM instructions

When ARM processor detects a data abort during a load multiple instruction, it modifies the operation of the instruction to ensure that recovery is possible.

1. Overwriting of registers stops when the abort happens. The aborting load will not take place but earlier ones may have overwritten registers. The PC is always the last register to be written and so will always be preserved.
2. The base register is restored, to its modified value if writeback was requested. This ensures recoverability in the case where the base register is also in the transfer list, and may have been overwritten before the abort occurred.

The data abort trap is taken when the load multiple has completed, and the system software must undo any base modification (and resolve the cause of the abort) before restarting the instruction.

5.8.8 Instruction cycle times

Normal LDM instructions take 1 instruction fetch, n data reads and 1 internal cycle and LDM PC takes 3 instruction fetches, n data reads and 1 internal cycle. For more information see *5.17 Instruction Speed Summary* on page 5-52.

STM instructions take 1 instruction fetch, n data reads and 1 internal cycle to execute.

n is the number of words transferred.

5.8.9 Assembler syntax

```
<LDM|STM>{cond}<FD|ED|FA|EA|IA|IB|DA|DB> Rn{!},<Rlist>{^}
```

`{cond}` two character condition mnemonic, see *Figure 5-2: Condition codes* on page 5-3

`Rn` an expression evaluating to a valid register number

`<Rlist>` a list of registers and register ranges enclosed in {} (eg {R0,R2-R7,R10}).

`{!}` if present requests writeback (W=1), otherwise W=0

`{^}` if present set S bit to load the CPSR along with the PC, or force transfer of user bank when in privileged mode

Addressing mode names

There are different assembler mnemonics for each of the addressing modes, depending on whether the instruction is being used to support stacks or for other purposes. The equivalences between the names and the values of the bits in the instruction are shown in the following table.

name	stack	other	L bit	P bit	U bit
pre-increment load	LDMED	LDMIB	1	1	1
post-increment load	LDMFD	LDMIA	1	0	1
pre-decrement load	LDMEA	LDMDB	1	1	0
post-decrement load	LDMFA	LDMDA	1	0	0
pre-increment store	STMFA	STMIB	0	1	1
post-increment store	STMEA	STMIA	0	0	1
pre-decrement store	STMFD	STMDB	0	1	0
post-decrement store	STMED	STMDA	0	0	0

Table 5-2: Addressing mode names

FD, ED, FA, EA define pre/post indexing and the up/down bit by reference to the form of stack required. The F and E refer to a “full” or “empty” stack, ie. whether a pre-index has to be done (full) before storing to the stack. The A and D refer to whether the stack is ascending or descending. If ascending, a STM will go up and LDM down, if descending, vice-versa.

IA, IB, DA, DB allow control when LDM/STM are not being used for stacks and simply mean Increment After, Increment Before, Decrement After, Decrement Before.

5.8.10 Examples

```
LDMFD SP!,{R0,R1,R2} ; unstack 3 registers

STMIA R0,{R0-R15}    ; save all registers

LDMFD SP!,{R15}      ; R15 <- (SP),CPSR unchanged
LDMFD SP!,{R15}^     ; R15 <- (SP), CPSR <- SPSR_mode (allowed
                     ;  only in privileged modes)
STMFD R13,{R0-R14}^  ; Save user mode regs on stack (allowed
                     ;  only in privileged modes)
```

These instructions may be used to save state on subroutine entry, and restore it efficiently on return to the calling routine:

```
STMED  SP!,{R0-R3,R14} ; save R0 to R3 to use as workspace
                       ;  and R14 for returning

BL     somewhere       ; this nested call will overwrite R14

LDMED  SP!,{R0-R3,R15} ; restore workspace and return
```

5.9 Single Data Swap (SWP)

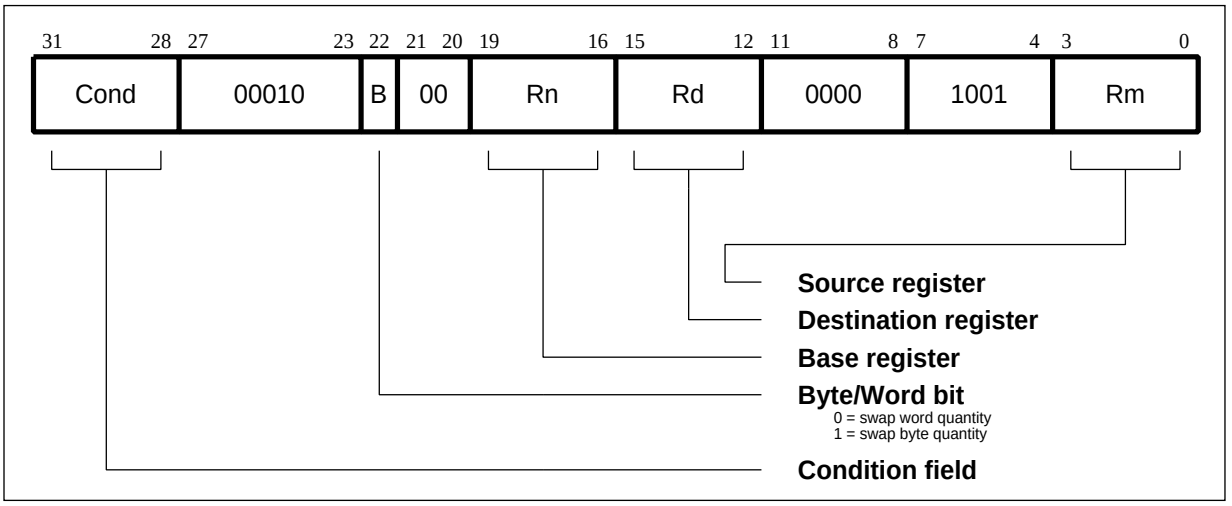


Figure 5-20: Swap instruction

The instruction is only executed if the condition is true. The various conditions are defined at the beginning of this chapter. The instruction encoding is shown in ◆*Figure 5-20: Swap instruction*.

The data swap instruction is used to swap a byte or word quantity between a register and external memory. This instruction is implemented as a memory read followed by a memory write which are “locked” together (the processor cannot be interrupted until both operations have completed, and the memory manager is warned to treat them as inseparable). This class of instruction is particularly useful for implementing software semaphores.

The swap address is determined by the contents of the base register (Rn). The processor first reads the contents of the swap address. Then it writes the contents of the source register (Rm) to the swap address, and stores the old memory contents in the destination register (Rd). The same register may be specified as both the source and destination.

The ARM7 CPU’s lock output goes HIGH for the duration of the read and write operations to signal to the external memory manager that they are locked together, and should be allowed to complete without interruption. This is important in multi-processor systems where the swap instruction is the only indivisible instruction which may be used to implement semaphores; control of the memory must not be removed from a processor while it is performing a locked operation.

5.9.1 Bytes and words

This instruction class may be used to swap a byte (B=1) or a word (B=0) between an ARM processor register and memory. The SWP instruction is implemented as a LDR followed by a STR and the action of these is as described in the section on single data transfers. In particular, the description of big and little-endian configuration applies to the SWP instruction.

5.9.2 Use of R15

R15 must not be used as an operand (Rd, Rn or Rs) in a SWP instruction.

5.9.3 Data aborts

If the address used for the swap is unacceptable to a memory management system, the memory manager can flag the problem by driving the aRM7 CPU's abort input HIGH. This can happen on either the read or the write cycle (or both), and in either case, the Data Abort trap will be taken. It is up to the system software to resolve the cause of the problem, then the instruction can be restarted and the original program continued.

5.9.4 Instruction cycle times

Swap instructions take 1 instruction fetch, 1 data read, 1 data write and 1 internal cycle. For more information see *5.17 Instruction Speed Summary* on page 5-52..

5.9.5 Assembler syntax

```
<SWP>{cond}{B} Rd,Rm,[Rn]
```

`{cond}`	two-character condition mnemonic, see *Figure 5-2: Condition codes* on page 5-3
`{B}`	if B is present then byte transfer, otherwise word transfer
`Rd,Rm,Rn`	are expressions evaluating to valid register numbers

5.9.6 Examples

```
SWP   R0,R1,[R2]     ; load R0 with the word addressed by R2, and
                     ; store R1 at R2

SWPB  R2,R3,[R4]     ; load R2 with the byte addressed by R4, and
                     ; store bits 0 to 7 of R3 at R4

SWPEQ R0,R0,[R1]     ; conditionally swap the contents of the
                     ; Software interrupt (SWI)
```

5.10 Software Interrupt (SWI)

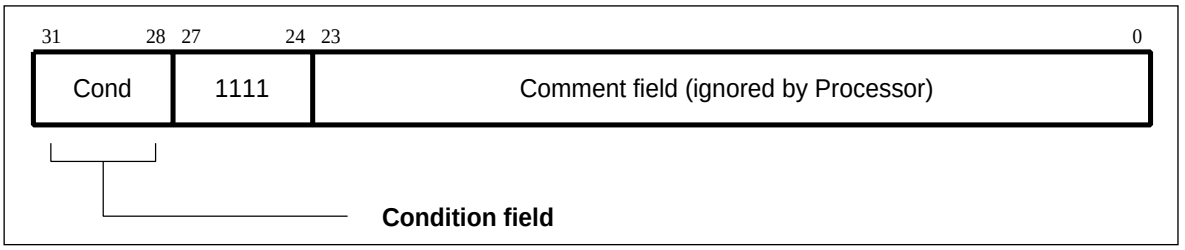


Figure 5-21: Software interrupt instruction

The instruction is only executed if the condition is true. The various conditions are defined at the beginning of this chapter. The instruction encoding is shown in *Figure 5-21: Software interrupt instruction* on page 5-36.

The software interrupt instruction is used to enter Supervisor mode in a controlled manner. The instruction causes the software interrupt trap to be taken, which effects the mode change. The PC is then forced to a fixed value (0x08) and the CPSR is saved in SPSR_svc. If the SWI vector address is suitably protected (by external memory management hardware) from modification by the user, a fully protected operating system may be constructed.

5.10.1 Return from the supervisor

The PC is saved in R14_svc upon entering the software interrupt trap, with the PC adjusted to point to the word after the SWI instruction. MOVS PC,R14_svc will return to the calling program and restore the CPSR.

Note that the link mechanism is not re-entrant, so if the supervisor code wishes to use software interrupts within itself it must first save a copy of the return address and SPSR.

5.10.2 Comment field

The bottom 24 bits of the instruction are ignored by the processor, and may be used to communicate information to the supervisor code. For instance, the supervisor may look at this field and use it to index into an array of entry points for routines which perform the various supervisor functions.

5.10.3 Instruction cycle times

Software interrupt instructions take 3 instruction fetches. For more information see *5.17 Instruction Speed Summary* on page 5-52.

5.10.4 Assembler syntax

```
SWI{cond} <expression>
```

`{cond}` two character condition mnemonic, see *Figure 5-2: Condition codes* on page 5-3

`<expression>` is evaluated and placed in the comment field (which is ignored by ARM processor).

5.10.5 Examples

```
SWI   ReadC             ; get next character from read stream
SWI   WriteI+”k”        ; output a “k” to the write stream
SWINE 0                 ; conditionally call supervisor
                        ; with 0 in comment field
```

The above examples assume that suitable supervisor code exists, for instance:

```
      0x08 B Supervisor ; SWI entry point

      EntryTable        ; addresses of supervisor routines
            DCD ZeroRtn
            DCD ReadCRtn
            DCD WriteIRtn
                . . .

      Zero  EQU  0
      ReadC EQU  256
      WriteIEQU  512

Supervisor

; SWI has routine required in bits 8-23 and data (if any) in
; bits 0-7.
; Assumes R13_svc points to a suitable stack

      STMFD R13,{R0-R2,R14}; save work registers and return
address
      LDR   R0,[R14,#-4]; get SWI instruction
      BIC   R0,R0,#0xFF000000; clear top 8 bits
      MOV   R1,R0,LSR#8 ; get routine offset
      ADR   R2,EntryTable; get start address of entry table
      LDR   R15,[R2,R1,LSL#2]; branch to appropriate routine

      WriteIRtn         ; enter with character in R0 bits 0-7
          .  .  .  .  .  .
      LDMFD R13,{R0-R2,R15}^; restore workspace and return
                        ; restoring processor mode and flags
```

5.11 Coprocessor Instructions

The ARM processor macrocell does not have an external coprocessor interface. This implementation of the ARM processor only supports a single on chip coprocessor, #15, which is used to program the on-chip control registers. This only supports the Coprocessor Register Transfer instructions (MRC and MCR).

All other coprocessor instructions will cause the ARM processor to take the undefined instruction trap. These coprocessor instructions can be emulated in software by the undefined trap handler. Even though external coprocessors cannot be connected to the ARM processor, the coprocessor instructions are still described here in full for completeness. Any external coprocessor referred to will be a software emulation.

5.12 Coprocessor data operations (CDP)

Use of the CDP instruction on the ARM processor will cause an undefined instruction trap to be taken, which may be used to emulate the coprocessor instruction.

The instruction is only executed if the condition is true. The various conditions are defined at the beginning of this chapter. The instruction encoding is shown in ◔*Figure 5-22: Coprocessor data operation instruction*.

This class of instruction is used to tell a coprocessor to perform some internal operation. No result is communicated back to the procesor, and it will not wait for the operation to complete. The coprocessor could contain a queue of such instructions awaiting execution, and their execution can overlap other activity, allowing the coprocessor and the processor to perform independent tasks in parallel.

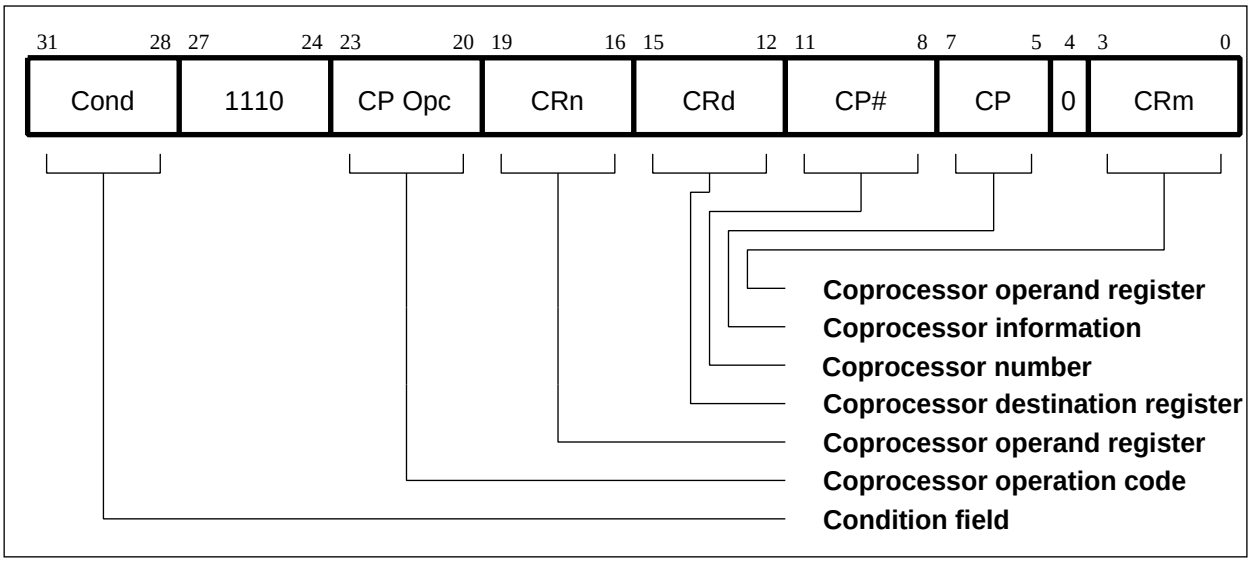


Figure 5-22: Coprocessor data operation instruction

5.12.1 The coprocessor fields

Only bit 4 and bits 24 to 31 are significant to the processor. The remaining bits are used by coprocessors. The above field names are used by convention, and particular coprocessors may redefine the use of all fields except CP# as appropriate. The CP# field is used to contain an identifying number (in the range 0 to 15) for each coprocessor, and a coprocessor will ignore any instruction which does not contain its number in the CP# field.

The conventional interpretation of the instruction is that the coprocessor should perform an operation specified in the CP Opc field (and possibly in the CP field) on the contents of CRn and CRm, and place the result in CRd.

5.12.2 Instruction cycle times

All CDP instructions are emulated in software: the number of cycles taken will depend on the coprocessor support software.

5.12.3 Assembler syntax

```
CDP{cond} p#,<expression1>,cd,cn,cm{,<expression2>}
```

`{cond}` two character condition mnemonic, see *Figure 5-2: Condition codes* on page 5-3

`p#` the unique number of the required coprocessor

`<expression1>` evaluated to a constant and placed in the CP Opc field

`cd, cn` and `cm` evaluate to the valid coprocessor register numbers CRd, CRn and CRm respectively

`<expression2>` where present is evaluated to a constant and placed in the CP field

5.12.4 Examples

```
CDP    p1,10,c1,c2,c3  ; request coproc 1 to do operation 10
                       ; on CR2 and CR3, and put the result in CR1
CDPEQ  p2,5,c1,c2,c3,2 ; if Z flag is set request coproc 2 to do
                       ; operation 5 (type 2) on CR2 and CR3,
                       ; and put the result in CR1
```

5.13 Coprocessor Data Transfers (LDC, STC)

Use of the LDC or STC instruction on the ARM processor will cause an undefined instruction trap to be taken, which may be used to emulate the coprocessor instruction.

The instruction is only executed if the condition is true. The various conditions are defined at the beginning of this chapter. The instruction encoding is shown in *Figure 5-23: Coprocessor data transfer instructions* on page 5-41.

This class of instruction is used to load (LDC) or store (STC) a subset of a coprocessors's registers directly to memory. The processor is responsible for supplying the memory address, and the coprocessor supplies or accepts the data and controls the number of words transferred.

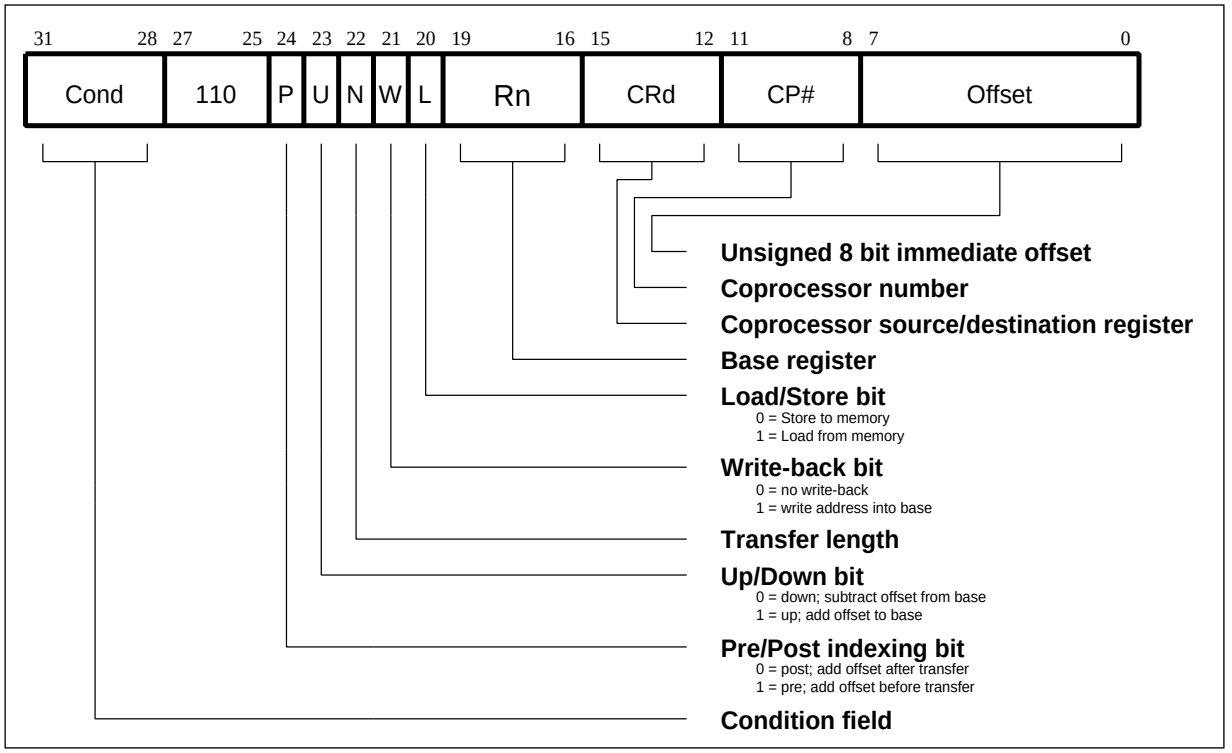


Figure 5-23: Coprocessor data transfer instructions

5.13.1 The coprocessor fields

The CP# field is used to identify the coprocessor which is required to supply or accept the data, and a coprocessor will only respond if its number matches the contents of this field.

The CRd field and the N bit contain information for the coprocessor which may be interpreted in different ways by different coprocessors, but by convention CRd is the register to be transferred (or the first register where more than one is to be

transferred), and the N bit is used to choose one of two transfer length options. For instance N=0 could select the transfer of a single register, and N=1 could select the transfer of all the registers for context switching.

5.13.2 Addressing modes

The processor is responsible for providing the address used by the memory system for the transfer, and the addressing modes available are a subset of those used in single data transfer instructions. Note, however, that for coprocessor data transfers the immediate offsets are 8 bits wide and specify *word* offsets, whereas for single data transfers they are 12 bits wide and specify *byte* offsets.

The 8-bit unsigned immediate offset is shifted left 2 bits and either added to (U=1) or subtracted from (U=0) the base register (Rn); this calculation may be performed either before (P=1) or after (P=0) the base is used as the transfer address. The modified base value may be overwritten back into the base register (if W=1), or the old value of the base may be preserved (W=0). Note that post-indexed addressing modes require explicit setting of the W bit, unlike LDR and STR which always write back when post-indexed.

The value of the base register, modified by the offset in a pre-indexed instruction, is used as the address for the transfer of the first word. The second word (if more than one is transferred) will go to or come from an address one word (4 bytes) higher than the first transfer, and the address will be incremented by one word for each subsequent transfer.

5.13.3 Address alignment

The base address should normally be a word aligned quantity. The bottom 2 bits of the address will appear on bits [1:0] at the address and might be interpreted by the memory system.

5.13.4 Use of R15

If Rn is R15, the value used will be the address of the instruction plus 8 bytes. Base writeback to R15 must not be specified.

5.13.5 Data aborts

If the address is legal but the memory manager generates an abort, the data trap will be taken. The writeback of the modified base will take place, but all other processor state will be preserved. The coprocessor is partly responsible for ensuring that the data transfer can be restarted after the cause of the abort has been resolved, and must ensure that any subsequent actions it undertakes can be repeated when the instruction is retried.

5.13.6 Instruction cycle times

All LDC instructions are emulated in software: the number of cycles taken will depend on the coprocessor support software.

5.13.7 Assembler syntax

```
<LDC|STC>{cond}{L} p#,cd,<Address>
```

LDC	load from memory to coprocessor
STC	store from coprocessor to memory
{L}	when present perform long transfer (N=1), otherwise perform short transfer (N=0)
{cond}	two character condition mnemonic, see *Figure 5-2: Condition codes* on page 5-3
p#	the unique number of the required coprocessor
cd	an expression evaluating to a valid coprocessor register number that is placed in the CRd field

<Address> can be:

- An expression which generates an address:
  <expression>
  The assembler will attempt to generate an instruction using the PC as a base and a corrected immediate offset to address the location given by evaluating the expression. This will be a PC relative, pre-indexed address. If the address is out of range, an error will be generated.
- A pre-indexed addressing specification:
  [Rn] offset of zero
  [Rn,<#expression>]{!} offset of <expression> bytes
- A post-indexed addressing specification:
  [Rn],<#expression> offset of <expression> bytes

Rn is an expression evaluating to a valid processor register number. Note, if Rn is R15 then the assembler will subtract 8 from the offset value to allow for processor pipelining.

{!} write back the base register (set the W bit) if ! is present

5.13.8 Examples

```
LDC    p1,c2,table      ; load c2 of coproc 1 from address
                        ; table, using a PC relative address.
STCEQL p2,c3,[R5,#24]!  ; conditionally store c3 of coproc 2
                        ; into an address 24 bytes up from R5,
                        ; write this address back to R5, and use
                        ; long transfer option (probably to
                        ; store multiple words)
```

Note that though the address offset is expressed in bytes, the instruction offset field is in words. The assembler will adjust the offset appropriately.

5.14 Coprocessor Register Transfers (MRC, MCR)

Use of the MRC or MCR instruction on the ARM processor to a coprocessor other than number 15 will cause an undefined instruction trap to be taken, which may be used to emulate the coprocessor instruction.

The instruction is only executed if the condition is true. The various conditions are defined at the beginning of this chapter. The instruction encoding is shown in *Figure 5-24: Coprocessor register transfer instructions* on page 5-44.

This class of instruction is used to communicate information directly between ARM processor and a coprocessor. An example of a coprocessor to processor register transfer (MRC) instruction would be a FIX of a floating point value held in a coprocessor, where the floating point number is converted into a 32-bit integer within the coprocessor, and the result is then transferred to a processor register. A FLOAT of a 32-bit value in a processor register into a floating point value within the coprocessor illustrates the use of a processor register to coprocessor transfer (MCR).

An important use of this instruction is to communicate control information directly from the coprocessor into the processor CPSR flags. As an example, the result of a comparison of two floating point values within a coprocessor can be moved to the CPSR to control the subsequent flow of execution.

Note The ARM processor has an internal coprocessor (#15) for control of on-chip functions. Accesses to this coprocessor are performed by coprocessor register transfers.

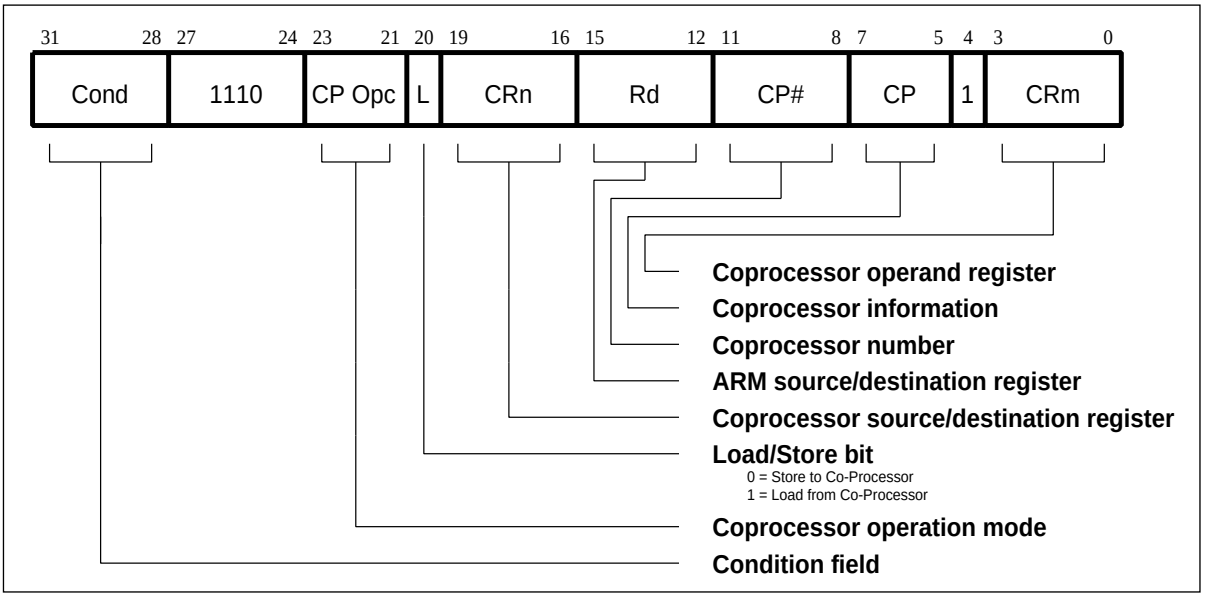


Figure 5-24: Coprocessor register transfer instructions

5.14.1 The coprocessor fields

The CP# field is used, as for all coprocessor instructions, to specify which coprocessor is being called upon.

The CP Opc, CRn, CP and CRm fields are used only by the coprocessor, and the interpretation presented here is derived from convention only. Other interpretations are allowed where the coprocessor functionality is incompatible with this one. The conventional interpretation is that the CP Opc and CP fields specify the operation the coprocessor is required to perform, CRn is the coprocessor register which is the source or destination of the transferred information, and CRm is a second coprocessor register which may be involved in depending on the particular operation specified.

5.14.2 Transfers to R15

When a coprocessor register transfer to ARM processor has R15 as the destination, bits 31, 30, 29 and 28 of the transferred word are copied into the N, Z, C and V flags respectively. The other bits of the transferred word are ignored, and the PC and other CPSR bits are unaffected by the transfer.

5.14.3 Transfers from R15

A coprocessor register transfer from ARM processor with R15 as the source register will store the PC+12.

5.14.4 Instruction cycle times

Access to the internal configuration register takes 1 instruction fetch cycle and 3 internal cycles. All other MRC instructions default to software emulation, and the number of cycles taken will depend on the coprocessor support software.

5.14.5 Assembler syntax

`<MCR|MRC>{cond} p#,<expression1>,Rd,cn,cm{,<expression2>}`

`MRC`	move from coprocessor to ARM processor register (L=1)
`MCR`	move from ARM processor register to coprocessor (L=0)
`{cond}`	two character condition mnemonic, see *Figure 5-2: Condition codes* on page 5-3
`p#`	the unique number of the required coprocessor
`<expression1>`	evaluated to a constant and placed in the CP Opc field
`Rd`	an expression evaluating to a valid ARM processor register number
`cn` and `cm`	expressions evaluating to the valid coprocessor register numbers CRn and CRm respectively
`<expression2>`	where present is evaluated to a constant and placed in the CP field

5.14.6 Examples

```
MRC     p2,5,R3,c5,c6   ; request coproc 2 to perform operation 5
                        ; on c5 and c6, and transfer the (single
                        ; 32 bit word) result back to R3

MCR     p6,0,R4,c6      ; request coproc 6 to perform operation 0
                        ; on R4 and place the result in c6

MRCEQ   p3,9,R3,c5,c6,2 ; conditionally request coproc 3 to
                        ; perform operation 9 (type 2) on c5 and
                        ; c6, and transfer the result back to R3
```

5.15 Undefined Instruction

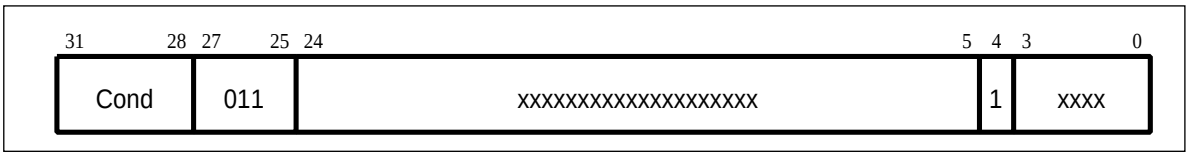


Figure 5-25: Undefined instruction

The instruction is only executed if the condition is true. The various conditions are defined at the beginning of this chapter. The instruction format is shown in *Figure 5-25: Undefined instruction*.

If the condition is true, the undefined instruction trap will be taken.

5.15.1 Assembler syntax

At present the assembler has no mnemonics for generating this instruction. If it is adopted in the future for some specified use, suitable mnemonics will be added to the assembler. Until such time, this instruction must not be used.

5.16 Instruction Set Examples

The following examples show ways in which the basic ARM processor instructions can combine to give efficient code. None of these methods saves a great deal of execution time (although they may save some), mostly they just save code.

5.16.1 Using the conditional instructions

1 using conditionals for logical OR

```
CMP   Rn,#p    ; if Rn=p OR Rm=q THEN GOTO Label
BEQ   Label
CMP   Rm,#q
BEQ   Label
```

can be replaced by

```
CMP   Rn,#p
CMPNE Rm,#q    ; if condition not satisfied try other test
BEQ   Label
```

2 absolute value

```
TEQ   Rn,#0       ; test sign
RSBMI Rn,Rn,#0    ; and 2's complement if necessary
```

3 multiplication by 4, 5 or 6 (run time)

```
MOV   Rc,Ra,LSL#2 ; multiply by 4
CMP   Rb,#5       ; test value
ADDCS Rc,Rc,Ra    ; complete multiply by 5
ADDHI Rc,Rc,Ra    ; complete multiply by 6
```

4 combining discrete and range tests

```
TEQ   Rc,#127     ; discrete test
CMPNE Rc,#" "-1   ; range test
MOVLS Rc,#"."     ; IF   Rc<=" " OR Rc=ASCII(127)
                  ; THEN Rc:="."
```

5 division and remainder

A number of divide routines for specific applications are provided in source form as part of the ANSI C library provided with the ARM Cross Development Toolkit, available from your supplier. A short general purpose divide routine follows.

```
                  ; enter with numbers in Ra and Rb
                  ;
      MOV   Rcnt,#1  ; bit to control the division
Div1  CMP   Rb,#0x80000000; move Rb until greater than Ra
      CMPCC Rb,Ra
      MOVCC Rb,Rb,ASL#1
      MOVCC Rcnt,Rcnt,ASL#1
      BCC   Div1
      MOV   Rc,#0
Div2  CMP   Ra,Rb    ; test for possible subtraction
      SUBCS Ra,Ra,Rb ; subtract if ok
```

```
ADDCS Rc,Rc,Rcnt; put relevant bit into result
MOVS  Rcnt,Rcnt,LSR#1; shift control bit
MOVNE Rb,Rb,LSR#1; halve unless finished
BNE   Div2
                ;
                ; divide result in Rc
                ; remainder in Ra
```

5.16.2 Pseudo random binary sequence generator

It is often necessary to generate (pseudo-) random numbers and the most efficient algorithms are based on shift generators with exclusive-OR feedback rather like a cyclic redundancy check generator. Unfortunately the sequence of a 32-bit generator needs more than one feedback tap to be maximal length (ie. 2^32-1 cycles before repetition), so this example uses a 33-bit register with taps at bits 33 and 20. The basic algorithm is newbit:=bit 33 eor bit 20, shift left the 33-bit number and put in newbit at the bottom; this operation is performed for all the newbits needed (ie. 32 bits). The entire operation can be done in 5 S cycles:

```
                    ; enter with seed in Ra (32 bits),
                      Rb (1 bit in Rb lsb), uses Rc
                    ;
TST   Rb,Rb,LSR#1   ; top bit into carry
MOVS  Rc,Ra,RRX     ; 33 bit rotate right
ADC   Rb,Rb,Rb      ; carry into lsb of Rb
EOR   Rc,Rc,Ra,LSL#12; (involved!)
EOR   Ra,Rc,Rc,LSR#20; (similarly involved!)
                    ;
                    ; new seed in Ra, Rb as before
```

5.16.3 Multiplication by constant using the barrel shifter

1 Multiplication by 2^n (1,2,4,8,16,32..)

```
MOV   Ra, Rb, LSL #n
```

2 Multiplication by 2^n+1 (3,5,9,17..)

```
ADD   Ra,Ra,Ra,LSL #n
```

3 Multiplication by 2^n-1 (3,7,15..)

```
RSB   Ra,Ra,Ra,LSL #n
```

4 Multiplication by 6

```
ADD   Ra,Ra,Ra,LSL #1; multiply by 3
MOV   Ra,Ra,LSL#1 ; and then by 2
```

5 Multiply by 10 and add in extra number

```
ADD   Ra,Ra,Ra,LSL#2; multiply by 5
ADD   Ra,Rc,Ra,LSL#1; multiply by 2 and add in next digit
```

6 General recursive method for Rb := Ra*C, C a constant:

a) If C even, say C = 2^n*D, D odd:

```
D=1:    MOV   Rb,Ra,LSL #n
D<>1:   {Rb := Ra*D}
        MOV Rb,Rb,LSL #n
```

b) If C MOD 4 = 1, say C = 2^n*D+1, D odd, n>1:

```
D=1:    ADD   Rb,Ra,Ra,LSL #n
D<>1:   {Rb := Ra*D}
        ADD Rb,Ra,Rb,LSL #n
```

c) If C MOD 4 = 3, say C = 2^n*D-1, D odd, n>1:

```
D=1:    RSB   Rb,Ra,Ra,LSL #n
D<>1:   {Rb := Ra*D}
        RSB Rb,Ra,Rb,LSL #n
```

This is not quite optimal, but close. An example of its non-optimality is multiply by 45 which is done by:

```
RSB   Rb,Ra,Ra,LSL#2; multiply by 3
RSB   Rb,Ra,Rb,LSL#2; multiply by 4*3-1 = 11
ADD   Rb,Ra,Rb,LSL# 2; multiply by 4*11+1 = 45
```

rather than by:

```
ADD   Rb,Ra,Ra,LSL#3; multiply by 9
ADD   Rb,Rb,Rb,LSL#2; multiply by 5*9 = 45
```

5.16.4 Loading a word from an unknown alignment

```
                    ; enter with address in Ra (32 bits)
                    ; uses Rb, Rc; result in Rd.
                    ; Note d must be less than c e.g. 0,1
                    ;
BIC   Rb,Ra,#3      ; get word aligned address
LDMIA Rb,{Rd,Rc}    ; get 64 bits containing answer
AND   Rb,Ra,#3      ; correction factor in bytes
MOVS  Rb,Rb,LSL#3   ; ...now in bits and test if aligned
MOVNE Rd,Rd,LSR Rb  ; produce bottom of result word
                    ; (if not aligned)
RSBNE Rb,Rb,#32     ; get other shift amount
ORRNE Rd,Rd,Rc,LSL Rb; combine two halves to get result
```

5.16.5 Loading a halfword (Little-endian)

```
LDR   Ra, [Rb,#2]    ; Get halfword to bits 15:0
MOV   Ra,Ra,LSL #16  ; move to top
MOV   Ra,Ra,LSR #16  ; and back to bottom
                     ; use ASR to get sign extended version
```

5.16.6 Loading a halfword (Big-endian)

```
LDR   Ra, [Rb,#2]    ; Get halfword to bits 31:16
MOV   Ra,Ra,LSR #16  ; and back to bottom
                     ; use ASR to get sign extended version
```

5.17 Instruction Speed Summary

Due to the pipelined architecture of the CPU, instructions overlap considerably. In a typical cycle one instruction may be using the data path while the next is being decoded and the one after that is being fetched. For this reason the following table presents the incremental number of cycles required by an instruction, rather than the total number of cycles for which the instruction uses part of the processor. Elapsed time (in cycles) for a routine may be calculated from these figures which are shown in *Table 5-3: ARM instruction speed summary* on page 5-52. These figures assume that the instruction is actually executed. Unexecuted instructions take one instruction fetch cycle.

Instruction	Cycle count
Data Processing - normal with register specified shift with PC written with register specified shift and PC written	1 instruction fetch 1 instruction fetch and 1 internal cycle 3 instruction fetches 3 instruction fetches and 1 internal cycle
MSR, MRS	1 instruction fetch
LDR - normal if the destination is the PC	1 instruction fetch, 1 data read and 1 internal cycle 3 instruction fetches, 1 data read and 1 internal cycle
STR	1 instruction fetch and 1 data write
LDM - normal if the destination is the PC	1 instruction fetch, n data reads and 1 internal cycle 3 instruction fetches, n data reads and 1 internal cycle
STM	1 instruction fetch and n data writes
SWP	1 instruction fetch, 1 data read, 1 data write and 1 internal cycle
B,BL	3 instruction fetches
SWI, trap	3 instruction fetches
MUL,MLA	1 instruction fetch and m internal cycles
CDP	the undefined instruction trap will be taken
LDC	the undefined instruction trap will be taken
STC	the undefined instruction trap will be taken
MCR	1 instruction fetch and 3 internal cycles for coproc 15
MRC	1 instruction fetch and 3 internal cycles for coproc 15

Table 5-3: ARM instruction speed summary

Where:

n is the number of words transferred.

m is the number of cycles required by the multiply algorithm, which is determined by the contents of Rs. Multiplication by any number between *2^(2m-3)* and *2^(2m-1)-1* takes 1S+mI cycles for *1<m>16*. Multiplication by 0 or 1 takes 1S+1I cycles, and multiplication by any number greater than or equal to 2^(29) takes 1S+16I cycles. The maximum time for any multiply is thus 1S+16I cycles.

The time taken for:

- an internal cycle - will always be one FCLK cycle
- an instruction fetch and data read - will be FCLK if a cache hit occurs, otherwise a full memory access is performed.
- a data write - will be FCLK if the write buffer (if enabled) has available space, otherwise the write will be delayed until the write buffer has free space. If the write buffer is not enabled a full memory access is always performed.
- Co-processor cycles - all coprocessor operations except MCR or MRC to registers 0 to 7 on coprocessor #15 (used for internal control) will cause the undefined instruction trap to be taken.

6 Cache, Write Buffer and Coprocessors

This chapter describes the ARM Processor configuration, instruction and data cache and the write buffer.

6.1	Instruction and Data Cache	6-2
6.2	Read-lock-write	6-3
6.3	IDC Enable/Disable and Reset	6-3
6.4	Write Buffer	6-4
6.5	Coprocessors	6-5

6.1 Instruction and Data Cache

The ARM Processor contains a 8kByte mixed instruction and data cache. The IDC has 512 lines of 16 bytes (4 words), arranged as a 4 way set associative cache, and uses the virtual addresses generated by the processor core. The IDC is always reloaded a line at a time (4 words). It may be enabled or disabled via the ARM Processor Control Register and is disabled on the internal nRESET. The operation of the cache is further controlled by the *Cacheable,* or C, bit stored in the Memory Management Page Table. For this reason, in order to use the IDC, the MMU must be enabled. The two functions may however be enabled simultaneously, with a single write to the Control Register.

6.1.1 Cacheable bit

The **Cacheable** bit determines whether data being read may be placed in the IDC and used for subsequent read operations. Typically main memory will be marked as Cacheable to improve system performance, and I/O space as Non-cacheable to stop the data being stored in ARM7100's cache. For example if the processor is polling a hardware flag in I/O space, it is important that the processor is forced to read data from the external peripheral, and not a copy of initial data held in the cache. The *Cacheable* bit can be configured for both pages and sections.

6.1.2 IDC operation

In the ARM Processor the cache will be searched regardless of the state of the C bit, only reads that miss the cache will be affected. The only effect of setting the cacheable bit to 0 is to inhibit cache replacement from occurring.

Cacheable reads C = 1

A linefetch of 4 words will be performed when a cache miss occurs in a cacheable area of memory and it will be randomly placed in a cache bank.

Uncacheable reads C = 0

An external memory access will be performed and the cache will not be written.

6.1.3 IDC validity

The IDC operates with virtual addresses, so care must be taken to ensure that its contents remain consistent with the virtual to physical mappings performed by the Memory Management Unit. If the Memory Mappings are changed, the IDC validity must be ensured.

Software IDC flush

The entire IDC may be marked as invalid by writing to the ARM Processor IDC Flush Register (Register 7). The cache will be flushed immediately the register is written, but note that the following two instruction fetches may come from the cache before the register is written.

6.1.4 Doubly mapped space

Since the cache works with virtual addresses, it is assumed that every virtual address maps to a different physical address. If the same physical location is accessed by more than one virtual address, the cache cannot maintain consistency, since each virtual address will have a separate entry in the cache, and only one entry will be updated on a processor write operation. To avoid any cache inconsistencies, both doubly-mapped virtual addresses should be marked as uncacheable.

6.2 Read-lock-write

The IDC treats the Read-Locked-Write instruction as a special case. The read phase always forces a read of external memory, regardless of whether the data is contained in the cache. The write phase is treated as a normal write operation (and if the data is already in the cache, the cache will be updated).The two phases are flagged as indivisible by asserting the lock signal from the ARM7 core.

6.3 IDC Enable/Disable and Reset

The IDC is automatically disabled and flushed on the internal nRESET. Once enabled, cacheable read accesses will cause lines to be placed in the cache.

To enable the IDC

To enable the IDC, make sure that the MMU is enabled first by setting bit 0 in Control Register, then enable the IDC by setting bit 2 in Control Register. The MMU and IDC may be enabled simultaneously with a single control register write.

To disable the IDC

To disable the IDC clear bit 2 in the Control Register and perform a flush by writing to the flush register.

6.4 Write Buffer

The ARM Processor write buffer is provided to improve system performance. It can buffer up to 8 words of data, and 4 independent addresses. It may be enabled or disabled via the W bit (bit 3) in the ARM Processor Control Register and the buffer is disabled and flushed on reset. The operation of the write buffer is further controlled by one bit, B, or Bufferable, which is stored in the Memory Management Page Tables. For this reason, in order to use the write buffer, the MMU must be enabled. The two functions may however be enabled simultaneously, with a single write to the Control Register. For a write to use the write buffer, both the W bit in the Control Register, and the B bit in the corresponding page table must be set.

6.4.1 Bufferable bit

This bit controls whether a write operation may or may not use the write buffer. Typically main memory will be bufferable and I/O space unbufferable. The Bufferable bit can be configured for both pages and sections.

6.4.2 Write buffer operation

When the CPU performs a write operation, the translation entry for that address is inspected and the state of the B bit determines the subsequent action. If the write buffer is disabled via the ARM Processor Control Register, bufferable writes are treated in the same way as unbuffered writes.

Bufferable write

If the write buffer is enabled and the processor performs a write to a bufferable area, the data is placed in the write buffer at MCLK speeds and the CPU continues execution. The write buffer then performs the external write in parallel. If however the write buffer is full (either because there are already 8 words of data in the buffer, or because there is no slot for the new address) then the processor is stalled until there is sufficient space in the buffer.

Unbufferable writes

If the write buffer is disabled or the CPU performs a write to an unbufferable area, the processor is stalled until the write buffer empties and the write completes externally, which may require synchronisation and several external clock cycles.

Read-lock-write

The write phase of a read-lock-write sequence is treated as an Unbuffered write, even if it is marked as buffered.

6.5 Coprocessors

The ARM7100 has no external coprocessor bus, so it is not possible to add external coprocessors to this device.

The ARM Processor still has an internal coprocessor designated #15 for internal control of the device. All coprocessor operations except MCR or MRC to registers 0 to 7 on coprocessor #15 will cause the undefined instruction trap to be taken.

7 ARM Processor MMU

This chapter describes the ARM Processor Memory Management Unit.

7.1	Introduction	7-2
7.2	MMU Program Accessible Registers	7-3
7.3	Address Translation	7-4
7.4	Translation Process	7-5
7.5	Translating Section References	7-8
7.6	Translating Small Page References	7-10
7.7	Translating Large Page References	7-11
7.8	MMU Faults and CPU Aborts	7-12
7.9	Fault Address and Fault Status Registers (FAR and FSR)	7-13
7.10	Domain Access Control	7-14
7.11	Fault Checking Sequence	7-15
7.12	Interaction of the MMU, IDC and Write Buffer	7-18
7.13	Effect of Reset	7-19

7.1 Introduction

The Memory Management MMU performs two primary functions: it translates virtual addresses into physical addresses, and it controls memory access permissions. The MMU hardware required to perform these functions consists of a Translation Look-aside Buffer (TLB), access control logic, and translation table walking logic.

The MMU supports memory accesses based on Sections or Pages. Sections are comprised of 1MB blocks of memory. Two different page sizes are supported: Small Pages consist of 4kB blocks of memory and Large Pages consist of 64kB blocks of memory. (Large Pages are supported to allow mapping of a large region of memory while using only a single entry in the TLB). Additional access control mechanisms are extended within Small Pages to 1kB Sub-Pages and within Large Pages to 16kB Sub-Pages.

The MMU also supports the concept of domains - areas of memory that can be defined to possess individual access rights. The Domain Access Control Register is used to specify access rights for up to 16 separate domains.

The TLB caches 64 translated entries. During most memory accesses, the TLB provides the translation information to the access control logic.

If the TLB contains a translated entry for the virtual address, the access control logic determines whether access is permitted. If access is permitted, the MMU outputs the appropriate physical address corresponding to the virtual address. If access is not permitted, the MMU signals the CPU to abort.

If the TLB misses (it does not contain a translated entry for the virtual address), the translation table walk hardware is invoked to retrieve the translation information from a translation table in physical memory. Once retrieved, the translation information is placed into the TLB, possibly overwriting an existing value. The entry to be overwritten is chosen by cycling sequentially through the TLB locations.

When the MMU is turned off (as happens on reset), the virtual address is output directly onto the physical address bus.

7.2 MMU Program Accessible Registers

The ARM Processor provides several 32-bit registers which determine the operation of the MMU. The format for these registers is shown in *Figure 7-1: MMU register summary* on page 7-3. A brief description of the registers is provided below. Each register will be discussed in more detail within the section that describes its use.

Data is written to and read from the MMU's registers using the ARM CPU's MRC and MCR coprocessor instructions.

The **Translation Table Base Register** holds the physical address of the base of the translation table maintained in main memory. Note that this base must reside on a 16kB boundary.

The **Domain Access Control Register** consists of sixteen 2-bit fields, each of which defines the access permissions for one of the sixteen Domains (D15-D0).

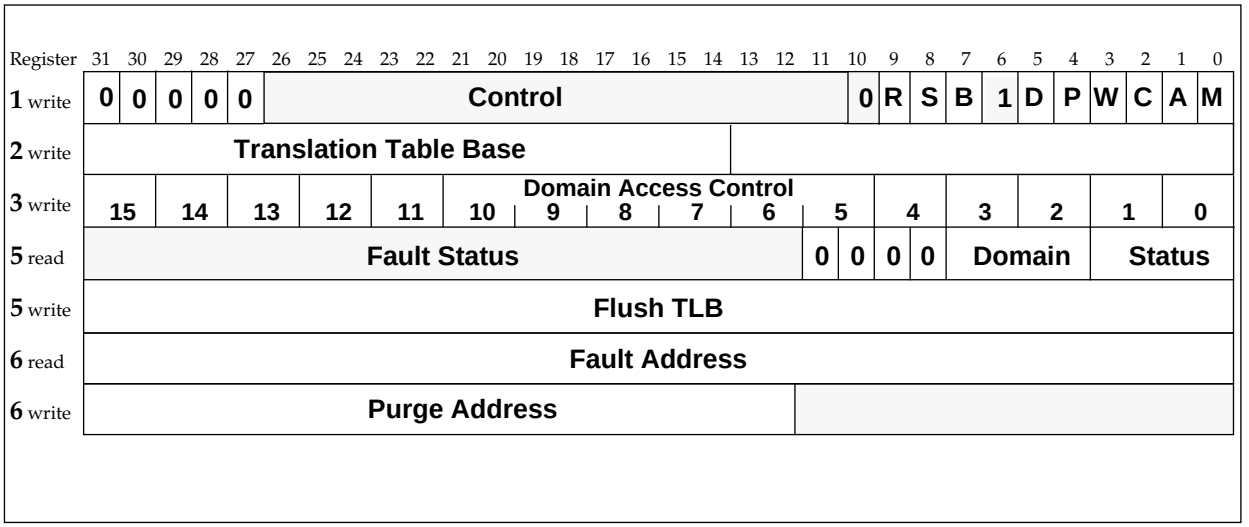


Figure 7-1: MMU register summary

Note *The registers not shown are reserved and should not be used.*

The **Fault Status Register** indicates the domain and type of access being attempted when an abort occurred. Bits 7:4 specify which of the sixteen domains (D15-D0) was being accessed when a fault occurred. Bits 3:1 indicate the type of access being attempted. The encoding of these bits is different for internal and external faults (as indicated by bit 0 in the register) and is shown in *Table 7-4: Priority encoding of fault status* on page 7-13. A write to this register flushes the TLB.

The **Fault Address Register** holds the virtual address of the access which was attempted when a fault occurred. A write to this register causes the data written to be treated as an address and, if it is found in the TLB, the entry is marked as invalid. (This operation is known as a TLB purge). The Fault Status Register and Fault Address Register are only updated for data faults, not for prefetch faults.

7.3 Address Translation

The MMU translates virtual addresses generated by the CPU into physical addresses to access external memory, and also derives and checks the access permission. Translation information, which consists of both the address translation data and the access permission data, resides in a translation table located in physical memory. The MMU provides the logic needed to traverse this translation table, obtain the translated address, and check the access permission.

There are three routes by which the address translation (and hence permission check) takes place. The route taken depends on whether the address in question has been marked as a section-mapped access or a page-mapped access; and there are two sizes of page-mapped access (large pages and small pages). However, the translation process always starts out in the same way, as described below, with a Level One fetch. A section-mapped access only requires a Level One fetch, but a page-mapped access also requires a Level Two fetch.

7.4 Translation Process

7.4.1 Translation table base

The translation process is initiated when the on-chip TLB does not contain an entry for the requested virtual address. The Translation Table Base (TTB) Register points to the base of a table in physical memory which contains Section and/or Page descriptors. The 14 low-order bits of the TTB Register are set to zero as illustrated in ➢*Figure 7-2: Translation table base register*, the table must reside on a 16kB boundary.

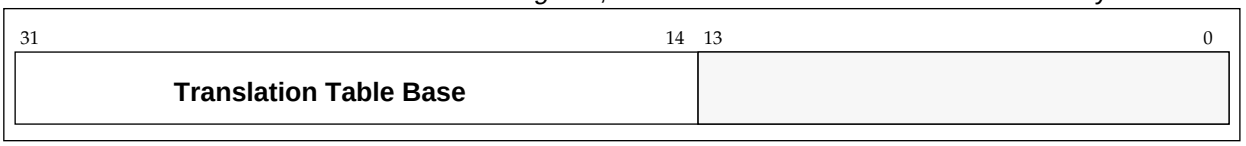


Figure 7-2: Translation table base register

7.4.2 Level one fetch

Bits 31:14 of the Translation Table Base register are concatenated with bits 31:20 of the virtual address to produce a 30-bit address as illustrated in ➢*Figure 7-3: Accessing the translation table first level descriptors*. This address selects a four-byte translation table entry which is a First Level Descriptor for either a Section or a Page (bit1 of the descriptor returned specifies whether it is for a Section or Page)

.

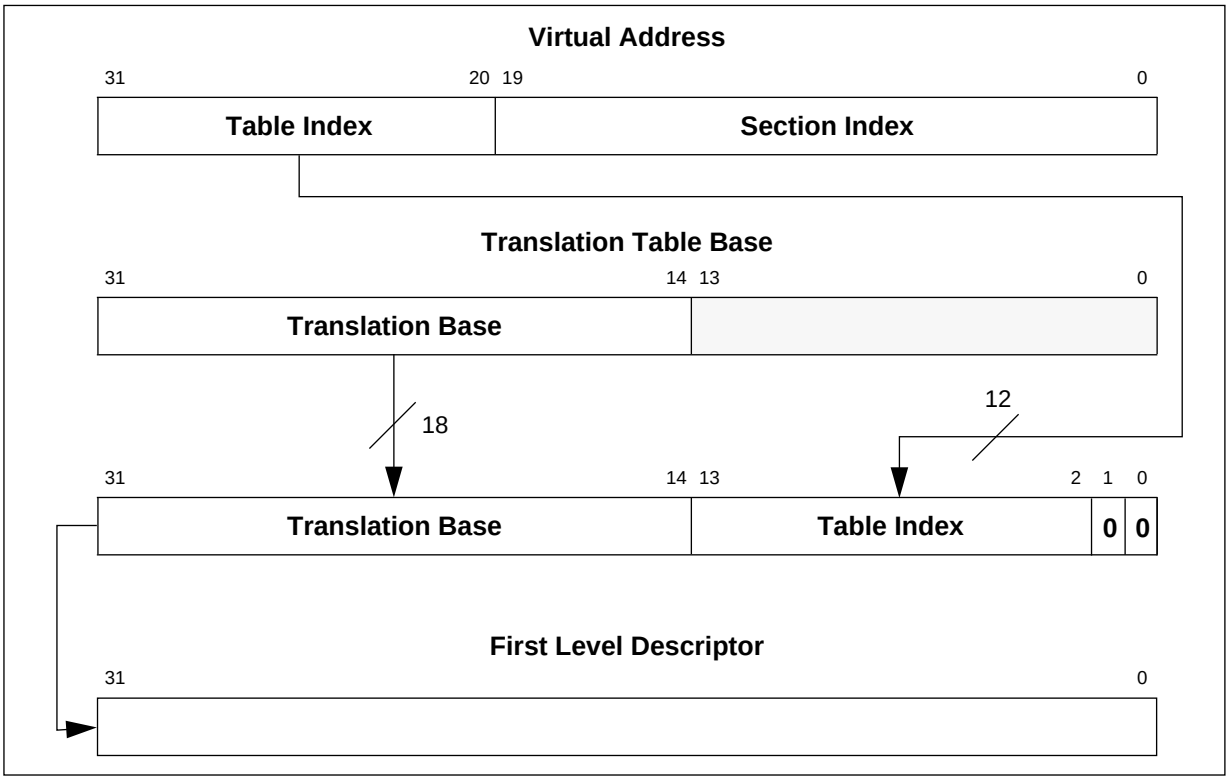


Figure 7-3: Accessing the translation table first level descriptors

7.4.3 Level one descriptor

The Level One Descriptor returned is either a Page Table Descriptor or a Section Descriptor, and its format varies accordingly. The following figure illustrates the format of Level One Descriptors.

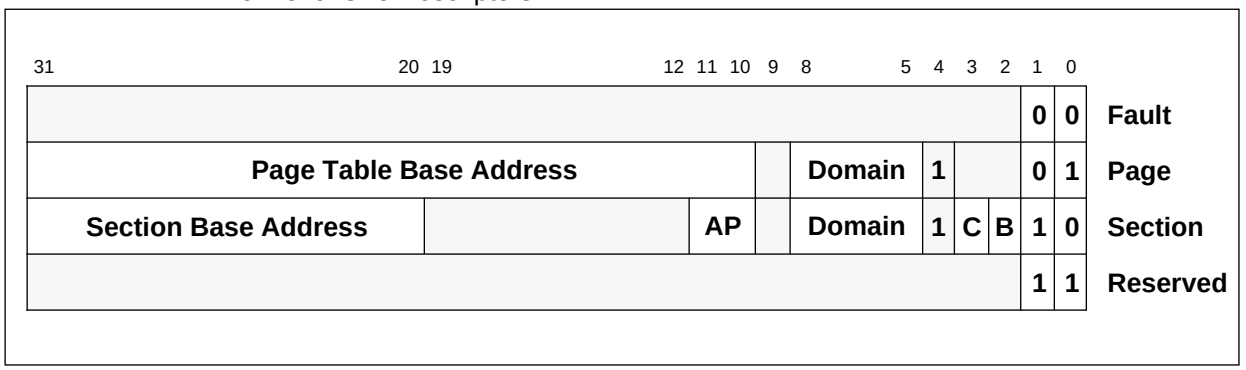


Figure 7-4: Level one descriptors

The two least significant bits indicate the descriptor type and validity, and are interpreted as shown below..

Value	Meaning	Notes
0 0	Invalid	Generates a Section Translation Fault
0 1	Page	Indicates that this is a Page Descriptor
1 0	Section	Indicates that this is a Section Descriptor
1 1	Reserved	Reserved for future use

Table 7-1: Interpreting level one descriptor bits [1:0]

7.4.4 Page table descriptor

Bits 3:2 are always written as 0.

Bit 4 should be written to 1 for backward compatibility.

Bits 8:5 specify one of the sixteen possible domains (held in the Domain Access Control Register) that contain the primary access controls.

Bits 31:10 form the base for referencing the Page Table Entry. (The page table index for the entry is derived from the virtual address as illustrated in *Figure 7-7: Small page translation* on page 7-10).

If a Page Table Descriptor is returned from the Level One fetch, a Level Two fetch is initiated as described below.

7.4.5 Section descriptor

Bits 3:2 (C, and B) control the cache- and write-buffer-related functions as follows:

C - Cacheable: indicates that data at this address will be placed in the cache (if the cache is enabled).

B - Bufferable: indicates that data at this address will be written through the write buffer (if the write buffer is enabled).

Bit 4 should be written to 1 for backward compatibility.

Bits 8:5 specify one of the sixteen possible domains (held in the Domain Access Control Register) that contain the primary access controls.

Bits 11:10 (AP) specify the access permissions for this section and are interpreted as shown in ❍*Table 7-2: Interpreting access permission (AP) bits* on page 7-7. Their interpretation is dependent upon the setting of the S and R bits (control register bits 8 and 9). Note that the Domain Access Control specifies the primary access control; the AP bits only have an effect in client mode. Refer to section on access permissions

AP	S	R	Permissions Supervisor	User	Notes
00	0	0	No Access	No Access	Any access generates a permission fault
00	1	0	Read Only	No Access	Supervisor read only permitted
00	0	1	Read Only	Read Only	Any write generates a permission fault
00	1	1	Reserved		
01	x	x	Read/Write	No Access	Access allowed only in Supervisor mode
10	x	x	Read/Write	Read Only	Writes in User mode cause permission fault
11	x	x	Read/Write	Read/Write	All access types permitted in both modes.
xx	1	1	Reserved		

Table 7-2: Interpreting access permission (AP) bits

Bits 19:12 are always written as 0.

Bits 31:20 form the corresponding bits of the physical address for the 1MByte section.

7.5 Translating Section References

Figure 7-5: Section translation illustrates the complete Section translation sequence. Note that the access permissions contained in the Level One Descriptor must be checked before the physical address is generated. The sequence for checking access permissions is described below.

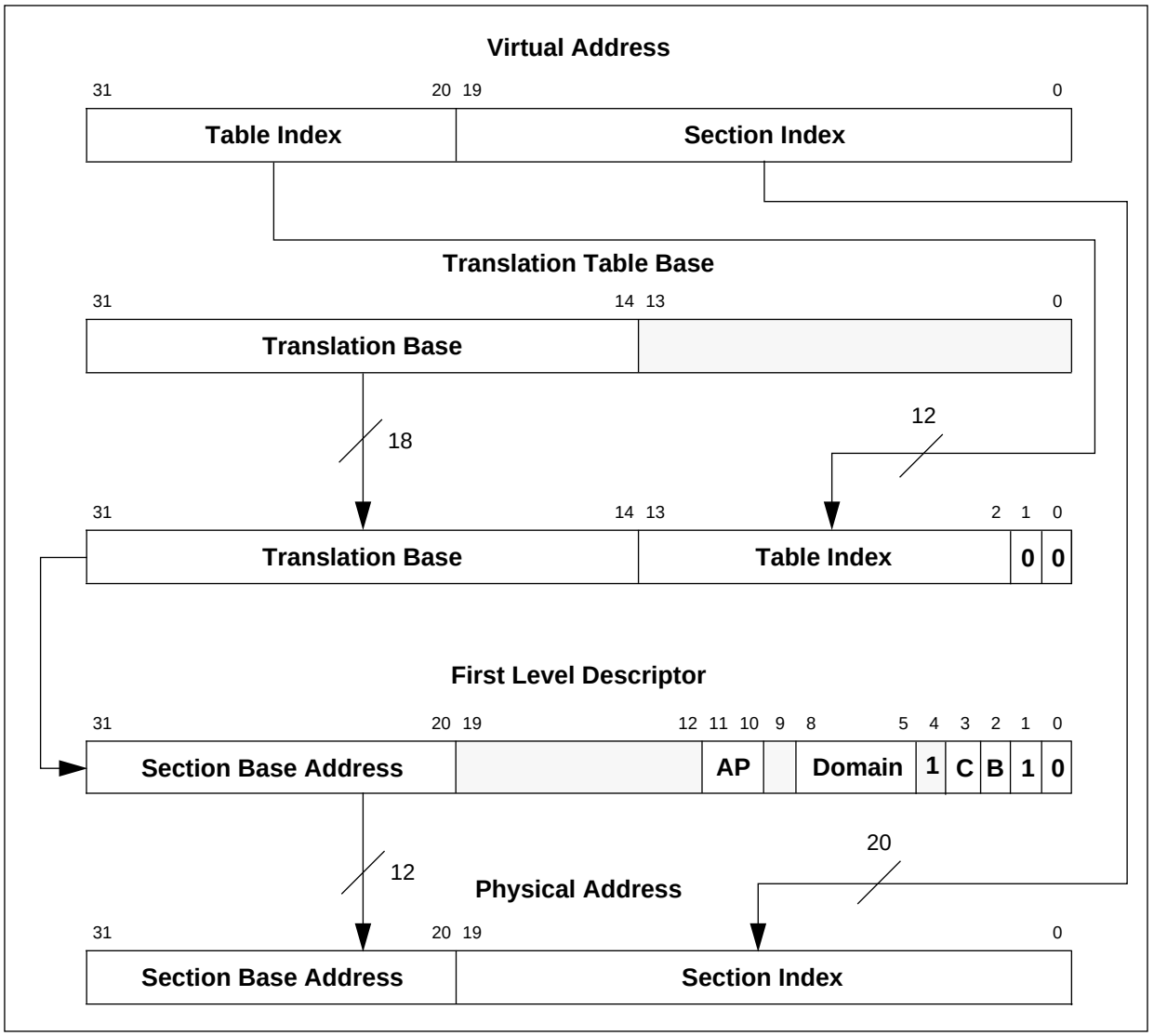


Figure 7-5: Section translation

7.5.1 Level two descriptor

If the Level One fetch returns a Page Table Descriptor, this provides the base address of the page table to be used. The page table is then accessed as described in *Figure 7-7: Small page translation* on page 7-10, and a Page Table Entry, or Level Two Descriptor, is returned. This in turn may define either a Small Page or a Large Page access. The figure below shows the format of Level Two Descriptors

.

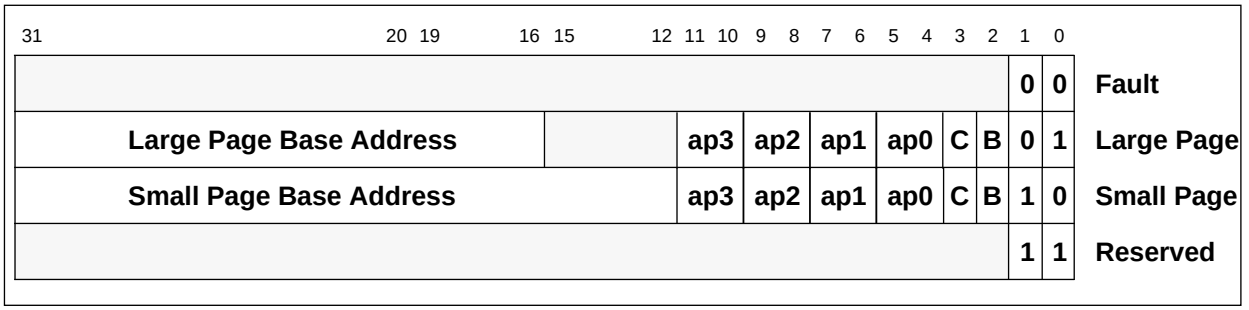


Figure 7-6: Page table entry (Level Two descriptor)

The two least significant bits indicate the page size and validity, and are interpreted as follows.

Value	Meaning	Notes
0 0	Invalid	Generates a Page Translation Fault
0 1	Large Page	Indicates that this is a 64 kB Page
1 0	Small Page	Indicates that this is a 4 kB Page
1 1	Reserved	Reserved for future use

Table 7-3: Interpreting page table entry bits 1:0

Bit 2 B - Bufferable: indicates that data at this address will be written through the write buffer (if the write buffer is enabled).

Bit 3 C - Cacheable: indicates that data at this address will be placed in the IDC (if the cache is enabled).

Bits 11:4 specify the access permissions (ap3 - ap0) for the four sub-pages and interpretation of these bits is described earlier in *Table 7-1: Interpreting level one descriptor bits [1:0]* on page 7-6.

For large pages, **bits 15:12** are programmed as 0.

Bits 31:12 (small pages) or bits **31:16** (large pages) are used to form the corresponding bits of the physical address - the physical page number. (The page index is derived from the virtual address as illustrated in *Figure 7-7: Small page translation* on page 7-10 and *Figure 7-8: Large page translation* on page 7-11).

7.6 Translating Small Page References

Figure 7-7: Small page translation illustrates the complete translation sequence for a 4kB Small Page. Page translation involves one additional step beyond that of a section translation: the Level One descriptor is the Page Table descriptor, and this is used to point to the Level Two descriptor, or Page Table Entry. (Note that the access permissions are now contained in the Level Two descriptor and must be checked before the physical address is generated. The sequence for checking access permissions is described later).

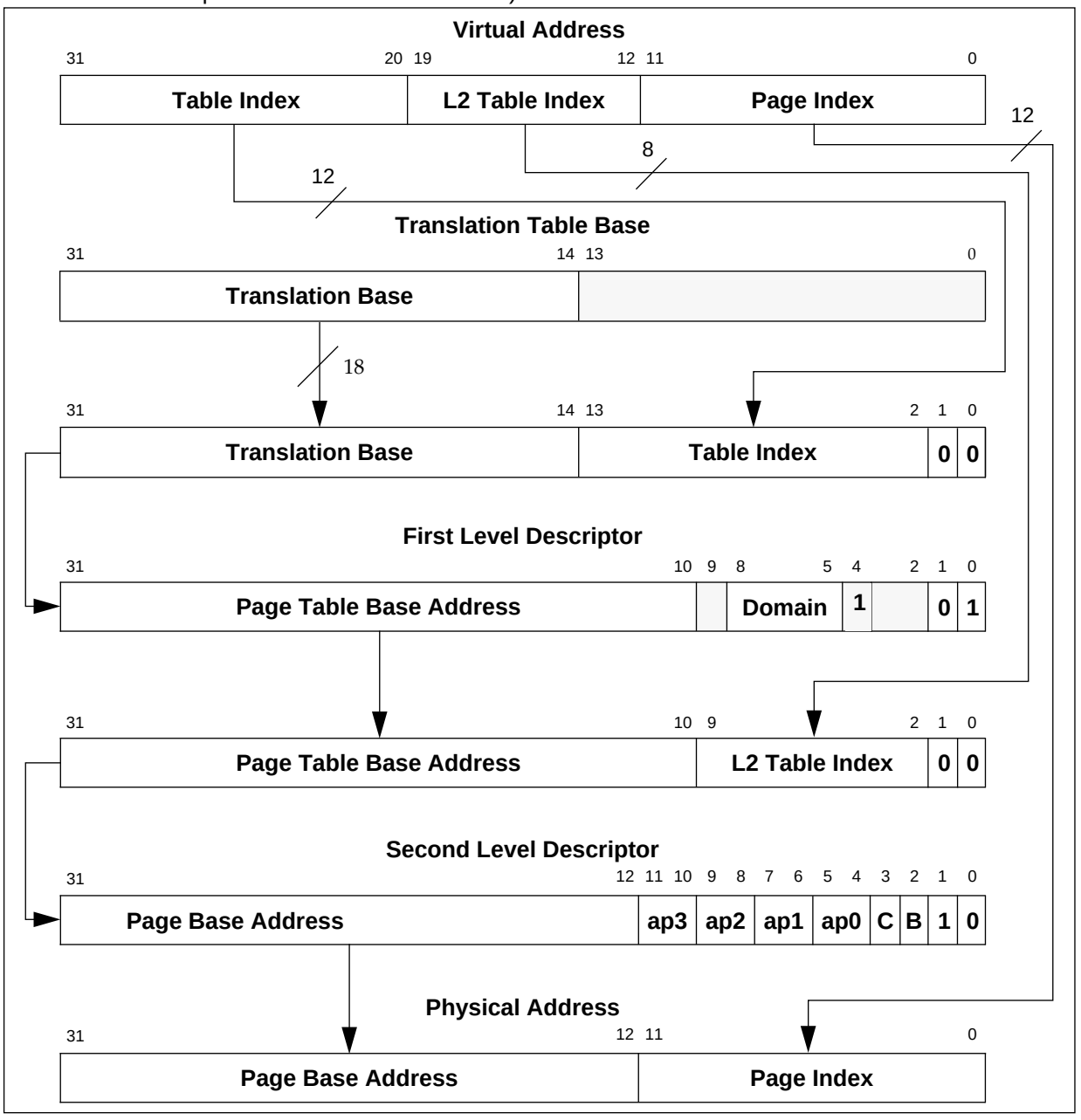


Figure 7-7: Small page translation

7.7 Translating Large Page References

Figure 7-8: Large page translation illustrates the complete translation sequence for a 64 kB Large Page. Note that since the upper four bits of the Page Index and low-order four bits of the Page Table index overlap, each Page Table Entry for a Large Page must be duplicated 16 times (in consecutive memory locations) in the Page Table.

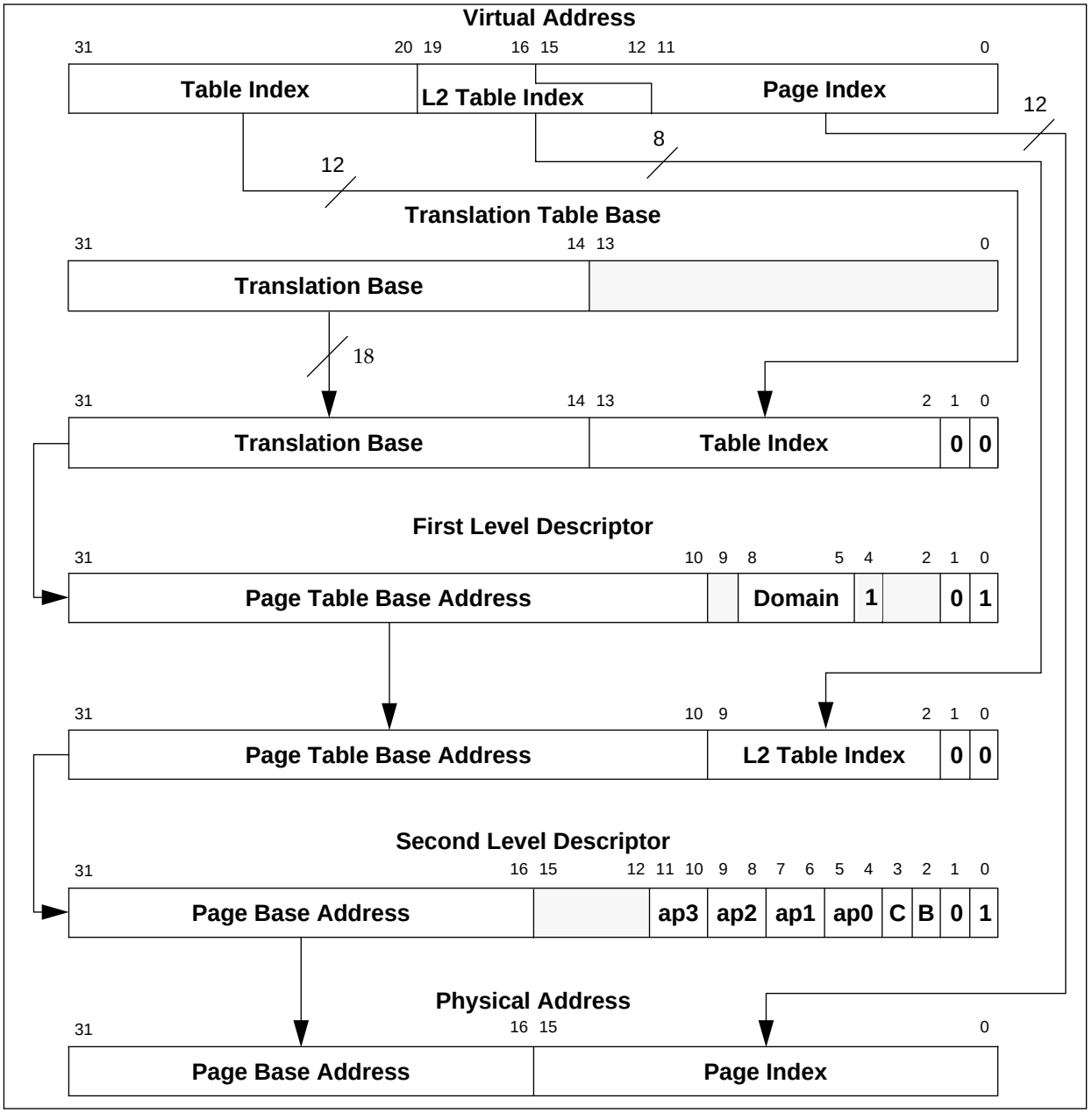


Figure 7-8: Large page translation

7.8 MMU Faults and CPU Aborts

The MMU generates four types of faults:

Alignment Fault

Translation Fault

Domain Fault

Permission Fault

The access control mechanisms of the MMU detect the conditions that produce these faults. If a fault is detected as the result of a memory access, the MMU will abort the access and signal the fault condition to the CPU. The MMU is also capable of retaining status and address information about the abort. The CPU recognises two types of abort: data aborts and prefetch aborts, and these are treated differently by the MMU.

If the MMU detects an access violation, it will do so before the external memory access takes place, and it will therefore inhibit the access.

If the ARM Processor is operating in fastbus mode an internally aborting access may cause the address on the external address bus to change, even though the external bus cycle has been cancelled. The address that is placed on the bus will be the translation of the address that caused the abort, though in the case of the a Translation Fault the value of this address will be undefined. No memory access will be performed to this address.

7.9 Fault Address and Fault Status Registers (FAR and FSR)

Aborts resulting from data accesses (data aborts) are acted upon by the CPU immediately, and the MMU places an encoded 4-bit value FS[3:0], along with the 4-bit encoded Domain number, in the Fault Status Register (FSR). In addition, the virtual processor address which caused the data abort is latched into the Fault Address Register (FAR). If an access violation simultaneously generates more than one source of abort, they are encoded in the priority given in *Table 7-4: Priority encoding of fault status* on page 7-13.

CPU instructions on the other hand are prefetched, so a prefetch abort simply flags the instruction as it enters the instruction pipeline. Only when (and if) the instruction is executed does it cause an abort; an abort is not acted upon if the instruction is not used (ie. it is branched around). Because instruction prefetch aborts may or may not be acted upon, the MMU status information is not preserved for the resulting CPU abort; for a prefetch abort, the MMU does not update the FSR or FAR.

The sections that follow describe the various access permissions and controls supported by the MMU and detail how these are interpreted to generate faults.

	Source		FS[3210]	Domain[3:0]	FAR
Highest	Alignment		00x1	x	valid
	Bus Error (translation)	level1 level2	1100 1110	x valid	valid valid
	Translation	Section Page	0101 0111	Note 2 valid	valid valid
	Domain	Section Page	1001 1011	valid valid	valid valid
	Permission	Section Page	1101 1111	valid valid	valid valid
	Bus Error (linefetch)	Section Page	0100 0110	valid valid	valid valid
Lowest	Bus Error (other)	Section Page	1000 1010	valid valid	valid valid

Table 7-4: Priority encoding of fault status

x is undefined, and may read as 0 or 1

Notes

1 Any abort masked by the priority encoding may be regenerated by fixing the primary abort and restarting the instruction.

2 In fact this register will contain bits[8:5] of the Level 1 entry which are undefined, but would encode the domain in a valid entry.

7.10 Domain Access Control

MMU accesses are primarily controlled via domains. There are 16 domains, and each has a 2-bit field to define it. Two basic kinds of users are supported: Clients and Managers. Clients use a domain; Managers control the behaviour of the domain. The domains are defined in the Domain Access Control Register. ◆*Figure 7-9: Domain access control register format* on page 7-14 illustrates how the 32 bits of the register are allocated to define the sixteen 2-bit domains.

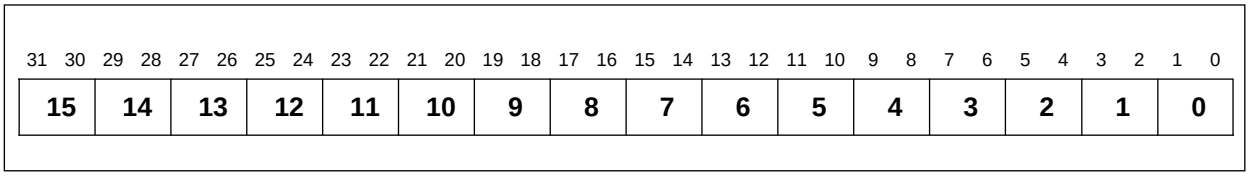


Figure 7-9: Domain access control register format

◆*Table 7-5: Interpreting access bits in domain access control register* defines how the bits within each domain are interpreted to specify the access permissions.

Value	Meaning	Notes
00	No Access	Any access will generate a Domain Fault.
01	Client	Accesses are checked against the access permission bits in the Section or Page descriptor.
10	Reserved	Reserved. Currently behaves like the no access mode.
11	Manager	Accesses are NOT checked against the access Permission bits so a Permission fault cannot be generated.

Table 7-5: Interpreting access bits in domain access control register

7.11 Fault Checking Sequence

The sequence by which the MMU checks for access faults is slightly different for Sections and Pages. The figure below illustrates the sequence for both types of accesses. The sections and figures that follow describe the conditions that generate each of the faults.

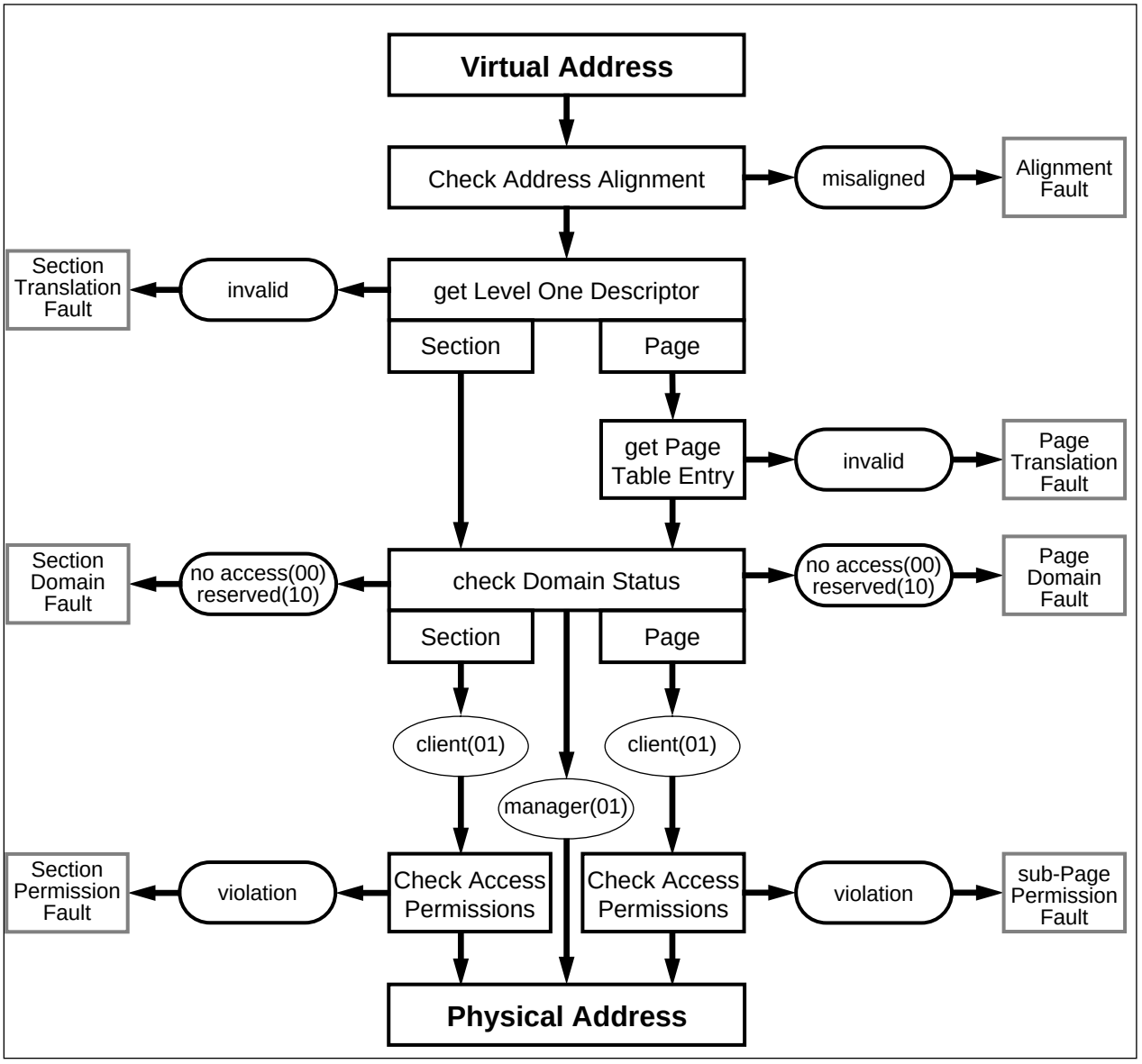


Figure 7-10: Sequence for checking faults

7.11.1 Alignment fault

If Alignment Fault is enabled (bit 1 in Control Register set), the MMU will generate an alignment fault on any data word access the address of which is not word-aligned irrespective of whether the MMU is enabled or not; in other words, if either of virtual address bits [1:0] are not 0. Alignment fault will not be generated on any instruction fetch, nor on any byte access. Note that if the access generates an alignment fault, the access sequence will abort without reference to further permission checks.

7.11.2 Translation fault

There are two types of translation fault: section and page.

1. A Section Translation Fault is generated if the Level One descriptor is marked as invalid. This happens if bits[1:0] of the descriptor are both 0 or both 1.
2. A Page Translation Fault is generated if the Page Table Entry is marked as invalid. This happens if bits[1:0] of the entry are both 0 or both 1.

7.11.3 Domain fault

There are two types of domain fault: section and page. In both cases the Level One descriptor holds the 4-bit Domain field which selects one of the sixteen 2-bit domains in the Domain Access Control Register. The two bits of the specified domain are then checked for access permissions as detailed in *Table 7-2: Interpreting access permission (AP) bits* on page 7-7. In the case of a section, the domain is checked once the Level One descriptor is returned, and in the case of a page, the domain is checked once the Page Table Entry is returned.

If the specified access is either No Access (00) or Reserved (10) then either a Section Domain Fault or Page Domain Fault occurs.

7.11.4 Permission fault

There are two types of permission fault: section and sub-page. Permission fault is checked at the same time as Domain fault. If the 2-bit domain field returns client (01), then the permission access check is invoked as follows:

section:

If the Level One descriptor defines a section-mapped access, then the AP bits of the descriptor define whether or not the access is allowed according to *Table 7-2: Interpreting access permission (AP) bits* on page 7-7. Their interpretation is dependent upon the setting of the S bit (Control Register bit 8). If the access is not allowed, then a Section Permission fault is generated.

sub-page:

If the Level One descriptor defines a page-mapped access, then the Level Two descriptor specifies four access permission fields (ap3..ap0) each corresponding to one quarter of the page. Hence for small pages, ap3 is selected by the top 1kB of the page, and ap0 is selected by the bottom 1kB of the page; for large pages, ap3 is selected by the top 16kB of the page, and ap0 is selected by the bottom 16kB of the page. The selected AP bits are then interpreted in exactly the same way as for a section (see *Table 7-2: Interpreting access permission (AP) bits* on page 7-7), the only difference being that the fault generated is a sub-page permission fault.

7.12 Interaction of the MMU, IDC and Write Buffer

The MMU, IDC and WB may be enabled/disabled independently. However, in order for the write buffer or the cache to be enabled the MMU must also be enabled. There are no hardware interlocks on these restrictions, so invalid combinations will cause undefined results.

MMU	IDC	WB
off	off	off
on	off	off
on	on	off
on	off	on
on	on	on

Table 7-6: Valid MMU, IDC and write buffer combinations

The following procedures must be observed.

To enable the MMU:

1 Program the Translation Table Base and Domain Access Control Registers

2 Program Level 1 and Level 2 page tables as required

3 Enable the MMU by setting bit 0 in the Control Register.

Note *Care must be taken if the translated address differs from the untranslated address as the two instructions following the enabling of the MMU will have been fetched using "flat translation" and enabling the MMU may be considered as a branch with delayed execution. A similar situation occurs when the MMU is disabled. Consider the following code sequence:*

```
MOV           R1, #0x1
MCR           15,0,R1,0,0     ; Enable MMU
Fetch Flat
Fetch Flat
Fetch Translated
```

To disable the MMU:

1 Disable the WB by clearing bit 3 in the Control Register.

2 Disable the IDC by clearing bit 2 in the Control Register.

3 Disable the MMU by clearing bit 0 in the Control Register.

Note *If the MMU is enabled, then disabled and subsequently re-enabled the contents of the TLB will have been preserved. If these are now invalid, the TLB should be flushed before re-enabling the MMU.*

Disabling of all three functions may be done simultaneously.

7.13 Effect of Reset

See *4.7 Reset* on page 4-16.

8 ARM7100 Programmer's Model

This chapter details the programmable registers for ARM7100.

8.1	Introduction	8-2
8.2	Summary of Registers	8-3
8.3	Register Descriptions	8-5

Preliminary

8.1 Introduction

ARM7100 contains internal programmable registers in addition to those in the ARM processor.

The registers internal to ARM7100 are all programmed by writing to memory locations 8000.0000 to 8000.FFFF. Accessing memory in this range will not cause any external bus activity unless broadcast mode is enabled. Any access to the undefined range from 8000.1000 to C000.0000 will have no effect.

Writes to bits that are not explicitly defined in the internal area are legal and will have no effect. Reads from bits not explicitly defined in the internal area are legal but will read undefined values.

It is only possible to access internal addresses as 32-bit words and they are always on a word boundary, except for the PIO port registers which can be accessed as bytes. Each internal register is valid for 256 bytes, since address bits in the range A[0:5] are not decoded, for example, the SYSFLG register appears at locations 8000.0140 to 8000.017C. The PIO port registers are byte wide but can be accessed as a word. These registers additionally decode A0 and A1.

8.2 Summary of Registers

Key

✓ can write/read

✗ do not write/read

Name	Address	Size	Read	Write	Function
PADR	8000.0000	8	✓	✓	Port A data register.
PBDR	8000.0001	8	✓	✓	Port B data register.
PCDR	8000.0002	8	✓	✓	Port C data register.
PDDR	8000.0003	8	✓	✓	Port D data register.
PADDR	8000.0040	8	✓	✓	Port A data direction register.
PBDDR	8000.0041	8	✓	✓	Port B data direction register.
PCDDR	8000.0042	8	✓	✓	Port C data direction register.
PDDDR	8000.0043	8	✓	✓	Port D data direction register.
PEDR	8000.0080	4	✓	✓	Port E data register.
PEDDR	8000.00C0	4	✓	✓	Port E data direction register.
SYSCON	8000.0100	32	✓	✓	System control register
SYSFLG	8000.0140	32	✓	✗	System status flags.
MEMCFG1	8000.0180	32	✓	✓	Expansion and ROM memory configuration register 1.
MEMCFG2	8000.01C0	32	✓	✓	Expansion and ROM memory configuration register 2.
DRFPR	8000.0200	8	✓	✓	DRAM refresh period register.
INTSR	8000.0240	16	✓	✗	Interrupt status register.
INTMR	8000.0280	16	✓	✓	Interrupt mask register.
LCDCON	8000.02C0	32	✓	✓	LCD control register.
TC1D	8000.0300	16	✓	✓	Read-write data to TC1.
TC2D	8000.0340	16	✓	✓	Read-write data to TC2.
RTCDR	8000.0380	32	✓	✓	Real time clock data register.
RTCMR	8000.03C0	32	✓	✓	Real time clock match register.
PMPCON	8000.0400	12	✓	✓	DC to DC pump control register.

Table 8-1: ARM7100 registers

Name	Address	Size	Read	Write	Function
CODR	8000.0440	8	✓	✓	CODEC data I/O register.
UARTDR	8000.0480	8	✓	✓	UART FIFO data register.
UBLCR	8000.04C0	32	✓	✓	UART bit rate and line control register.
SYNCIO	8000.0500	16	✓	✓	Synchronous serial I/O data register
PALLSW	8000.0540	32	✓	✓	Least significant 32-bit word of LCD palette register
PALMSW	8000.0580	32	✓	✓	Most significant 32-bit word of LCD palette register
STFCLR	8000.05C0	-	✗	✓	Write to clear all start up reason flags.
BLEOI	8000.0600	-	✗	✓	Write to clear battery low interrupt.
MCEOI	8000.0640	-	✗	✓	Write to clear **MEDCHG** interrupt.
TEOI	8000.0680	-	✗	✓	Write to clear tick and watchdog interrupt.
TC1EOI	8000.06C0	-	✗	✓	Write to clear TC1 interrupt.
TC2EOI	8000.0700	-	✗	✓	Write to clear TC2 interrupt.
RTCEOI	8000.0740	-	✗	✓	Write to clear RTC match interrupt.
UMSEOI	8000.0780	-	✗	✓	Write to clear UART modem status changed interrupt.
COEOI	8000.07C0	-	✗	✓	Write to clear CODEC sound interrupt
HALT	8000.0800	-	✗	✓	Write to enter idle state
STDBY	8000.0840	-	✗	✓	Write to enter standby state
Reserved	8000.0880 - 8000.0FFF	-			Write will have no effect, read is undefined

Table 8-1: ARM7100 registers (Continued)

8.3 Register Descriptions

All internal registers in ARM7100 are reset (cleared to zero) by a system reset (**nPOR**, **nRESET** or **nPWRFL** signals becoming active), except for the DRAM refresh period register (DRFPR) which is only reset by **nPOR** becoming active. This ensures that the contents of DRAM are preserved though a user reset or power fail condition. Additionally, the real time clock registers are only cleared by **nPOR**.

8.3.1 Port A data register (PADR)

Values written to this 8-bit read-write register are output on port A pins if the corresponding data direction bits are set HIGH (port output). Values read from this register reflect the external state of port A, not necessarily the value written to it. All bits are cleared by a system reset.

8.3.2 Port B data register (PBDR)

Values written to this 8-bit read-write register are output on port B pins if the corresponding data direction bits are set HIGH (port output). Values read from this register reflect the external state of port B, not necessarily the value written to it. All bits are cleared by a system reset.

8.3.3 Port C data register (PCDR)

Values written to this 8-bit read-write register are output on port C pins if the corresponding data direction bits are set LOW (port output). Values read from this register reflect the external state of port C, not necessarily the value written to it. All bits are cleared by a system reset.

8.3.4 Port D data register (PDDR)

Values written to this 8-bit read-write register are output on port D pins if the corresponding data direction bits are set LOW (port output). Values read from this register reflect the external state of port C, not necessarily the value written to it. All bits are cleared by a system reset.

8.3.5 Port A data direction register (PADDR)

Bits set in this 8-bit read-write register select the corresponding pin in port A to become an output. Clearing a bit sets the pin to input. All bits are cleared by a system reset.

8.3.6 Port B data direction register (PBDDR)

Bits set in this 8-bit read-write register select the corresponding pin in port B to become an output. Clearing a bit sets the pin to input. All bits are cleared by a system reset.

8.3.7 Port C data direction register (PCDDR)

Bits cleared in this 8-bit read-write register select the corresponding pin in port C to become an output. Setting a bit sets the pin to input. All bits are cleared by a system reset so that port C is output by default.

8.3.8 Port D data direction register (PDDDR)

Bits cleared in this 8-bit read-write register select the corresponding pin in port D to become an output, setting a bit sets the pin to input. All bits are cleared by a system reset so that port D is output by default.

8.3.9 Port E data register (PEDR)

Values written to this 4-bit read-write register will be output on port E pins if the corresponding data direction bits are set HIGH (port output). Values read from this register reflect the external state of port E, not necessarily the value written to it. All bits are cleared by a system reset.

8.3.10 Port E data direction register (PDDDR)

Bits set in this 4-bit read-write register will select the corresponding pin in port E to become an output, clearing bit sets the pin to input. All bits are cleared by a system reset so that port E is input by default.

8.3.11 The system control register (SYSCON)

The system control register is a 21-bit read /write register controlling all the general configuration of ARM7100 as well as operating modes for peripheral devices. All bits in this register are cleared by a system reset.

The bits in the system control register SYSCON are shown in *Figure 8-1: The system control register*.

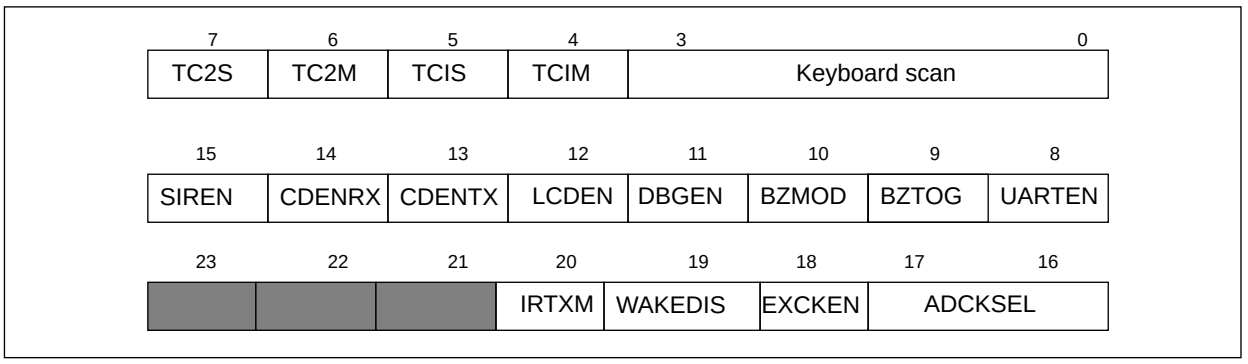


Figure 8-1: The system control register

Keyboard scan — This 4-bit field defines the state of the keyboard column drives. *Table 8-2: Keyboard scan field* gives definitions of these states.

Keyboard Scan	Column
0	All driven HIGH
1	All driven LOW
2 - 7	All Tristate
8	Column 0 only driven HIGH
9	Column 1 only driven HIGH
10	Column 2 only driven HIGH
11	Column 3 only driven HIGH
12	Column 4 only driven HIGH
13	Column 5 only driven HIGH
14	Column 6 only driven HIGH
15	Column 7 only driven HIGH

Table 8-2: Keyboard scan field

TC1M — Timer counter 1 mode. Setting this bit sets TC1 to prescale mode. Clearing it sets free running mode.

TC1S — Timer counter 1 clock source. Setting this bit sets the TC1 clock source to 512 KHz. Clearing it sets the clock source to 2KHz.

TC2M — Timer counter 2 mode. Setting this bit sets TC2 to prescale mode. Clearing it sets free running mode.

TC2S — Timer counter 2 clock source. Setting this bit sets the TC2 clock source to 512 KHz. Clearing it sets the clock source to 2KHz.

UARTEN — Internal UART enable bit. Setting this bit enables the internal UART.

BZTOG — This bit is used to drive the **BUZ** output directly.

BZMOD — This bit sets the **BUZ** output mode:

0 **BUZ** is connected directly to the BZTOG bit

1 **BUZ** is connected to the TC1 under flow bit

ADCKSEL — Microwire / SPI peripheral clock speed select. This 2-bit field selects the frequency of the ADC sample clock. This is twice the frequency of the synchronous serial ADC interface clock. ➲*Table 8-3: ADCCLK frequencies* shows the available frequencies.

Preliminary

ADCKSEL	ADC Sample Frequency (kHz)	ADC Interface Frequency (kHz)
00	8	4
01	32	16
10	128	64
11	256	128

Table 8-3: ADCCLK frequencies

DBGEN — Setting this bit enables debug/broadcast mode. In this mode, all internal accesses are output as if they were reads or writes to expansion memory addressed by CS6. CS6 will still be active in its standard address range. In addition the internal interrupt request and the fast interrupt request signals to the processor are output on port E bits 1 and 2 ie. in debug mode:

CS6 = CS6 / internal strobe
PE1 = nIRQ
PE2 = nFIQ

LCDEN — Setting this bit enables the LCD controller.

CDENTX — CODEC interface enable Tx bit. Setting this bit enables the CODEC interface for data transmission to an external CODEC device.

CDENRX — CODEC interface enable Rx bit. Setting this bit enables the CODEC interface for data reception from an external CODEC device.

SIREN — SIR protocol encoding enable bit. This has no effect if the UART is not enabled.

EXCLKEN — External expansion clock enable. If this bit is set the **EXPCLK** is enabled continuously with the same speed and phase as the CPU clock and will free run all the time the main oscillator is running. This bit should not be left set all the time for power consumption reasons. If the system enters the standby state the **EXPCLK** will become undefined. If this bit is clear **EXPCLK** will be active during memory cycles to expansion slots that have external wait state generation enabled only.

WAKEDIS — Switch on via the wakeup input is disabled if this bit is set.

IRTXM — IrDA Tx mode bit. This bit controls the IrDA encoding strategy. Clearing it means each zero bit transmitted is represented as a pulse of width 3/16th of the bit rate period. Setting this bit means each zero bit is represented as a pulse of width 3/16th of the period of 115,000 bit rate clock ie. 1.6µSec regardless of the selected bit rate. Setting this bit will use less power but probably reduce transmission distances.

BITS 21-31 — Reserved. Write will have no effect, will always read zero.

8.3.12 The System Status Flags Register (SYSFLG)

The system status flags register (SYSFLG) is a 32-bit read only register which indicates various system information. The bits in SYSFLG are defined in ➲*Figure 8-2: The System Status Flag Register* and are described below.

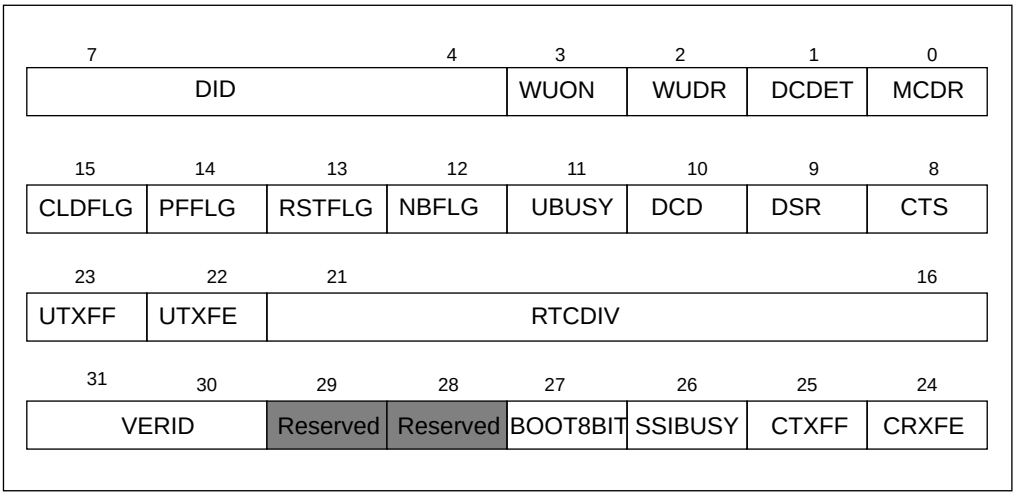


Figure 8-2: The System Status Flag Register

MCDR — This bit reflects the non-latched status of the media changed input.

DCDET — This bit reflects the inverted state of the **nEXTPWR** input pin.

WUDR — Wake up direct read. This bit reflects the non-latched state of the **WAKEUP** signal.

WUON — This bit is set if the system has been brought out of standby by a rising edge on the **WAKEUP** signal. It is only cleared by a system reset or by writing to the HALT or STDBY locations.

DID — Display ID nibble. This 4-bit nibble reflects the latched state of the 4 LCD data lines **DO[3:0]**. The state of the 4 LCD data lines is latched by the LCDEN bit and so always reflects the last state of these lines before the LCD controller was enabled. These bits identify the LCD display panel fitted.

CTS — This bit reflects the current status of the clear to send (CTS) modem control input to the built in UART.

DSR — This bit reflects the current status of the data set ready (DSR) modem control input to the built in UART.

DCD — This bit reflects the current status of the data carrier detect (DCD) modem control input to the built in UART.

UBUSY	UART transmitter busy. Set while the internal UART is busy transmitting data, it is guaranteed to remain set until the complete byte has been sent, including all stop bits.
NBFLG	The new battery flag bit is set if a LOW to HIGH transition has occurred on the **nBATCHG** input. It is cleared by writing to the STFCLR location.
RSTFLG	The reset flag is set if the **nURESET** input has been forced LOW. It is cleared by writing to the STFCLR location.
PFFLG	The Power Fail Flag is set if the system has been reset by the **PWRFL** input pin. It is cleared by writing to the STFCLR location.
CLDFLG	The cold start flag is set if ARM7100 has been reset with a power on reset. It is cleared by writing to the STFCLR location.
RTCDIV	This 6-bit field reflects the number of 64Hz ticks that have passed since the last increment of the RTC. It is the output of the divide by 64 chain that divides the 64Hz tick clock down to 1Hz for the RTC. The MSB is the 32Hz output, the LSB is the 1Hz output.
URXFE	UART receiver FIFO empty. The meaning of this bit depends on the state of the UFIFOEN bit in the UART bit rate and line control register. If the FIFO is disabled, this bit is set when the Rx holding register is empty. If the FIFO is enabled the URXFE bit will be set when the Rx FIFO is empty.
UTXFF	UART transmit FIFO full. The meaning of this bit depends on the state of the UFIFOEN bit in the UART bit rate and line control register. If the FIFO is disabled, this bit is set when the Tx holding register is full. If the FIFO is enabled the UTXFF bit will be set when the Tx FIFO is full.
CRXFE	The CODEC Rx FIFO empty bit is set if the 16 byte CODEC Rx FIFO is empty.
CTXFF	The CODEC Tx FIFO full bit is set if the 16 byte CODEC Tx FIFO is full.
SSIBUSY	The synchronous serial interface busy bit is set while data is being shifted in or out of the synchronous serial interface. When clear, data is valid to read.
Reserved	This will always read zero.
VERID	Version ID bits. These 2 bits determine the revision id for ARM7100. It will read 0 for the first revision.
BOOT8BIT	This bit indicates the default (power on reset) bus width of the ROM interface. If set, the initial bus width will be 8 bits. If clear, it will be 32 bits. See *8.3.13 Memory configuration register 1 (MEMCFG1)* on page 8-11 and *8.3.14 Memory configuration register 2 (MEMCFG2)* on page 8-11 for more

details on the ROM interface bus width. The state of this bit is determined by the state of Port E bit 0 during power on reset. LOW during power on reset will clear the BOOT8BIT bit and the system will boot from a 32-bit ROM. HIGH during power on reset will set the BOOT8BIT bit and the system will boot from an 8-bit ROM.

8.3.13 Memory configuration register 1 (MEMCFG1)

The memory configuration register 1 is a 32-bit read-write register which sets the configuration of the four expansion and ROM selects **nCS[0:3]**. Each select is configured with a one byte field, starting with expansion select 0.

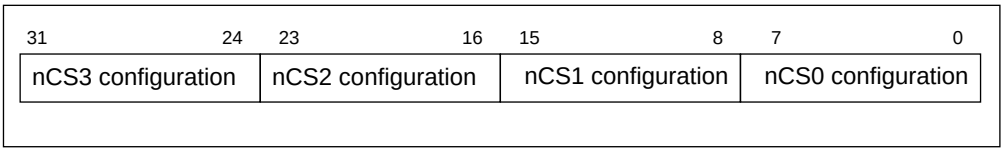


Figure 8-3: Memory configuration register 1

8.3.14 Memory configuration register 2 (MEMCFG2)

The memory configuration register 2 is a 32-bit read-write register which sets the configuration of the four expansion and ROM selects **CS[4:7**]. Each select is configured with a one byte field, starting with expansion select 4.

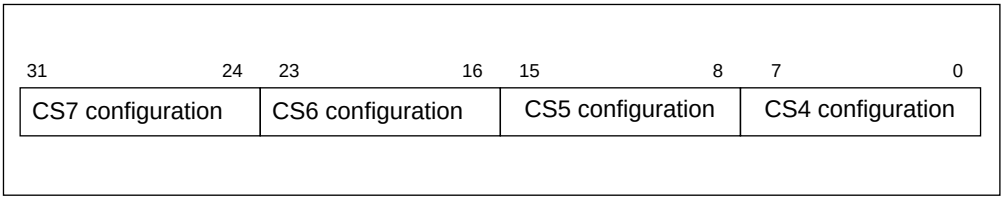


Figure 8-4: Memory configuration register 2

Each of the 8-bit fields in the memory configuration registers are identical and define the number of wait states, define the bus width, enable the **EXPCLK** output during accesses and enable sequential mode access. This is shown in *Figure 8-5: Byte fields in the memory configuration register* below.

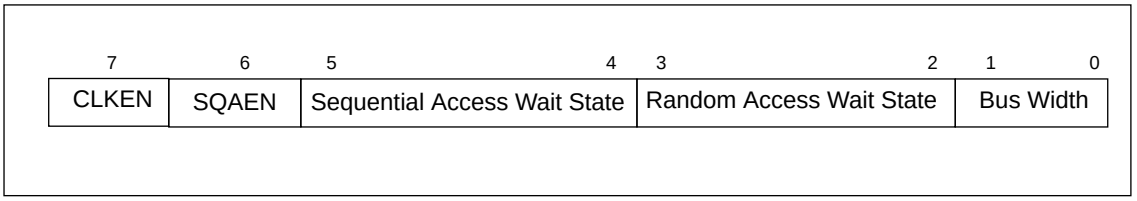


Figure 8-5: Byte fields in the memory configuration register

Table 8-4: Values of the bus width field defines the bus width field. The effect of this field is dependent on the BOOT8BIT bit which can be read in the SYSFLG register. All bits in the memory configuration register are cleared by a system reset and the state of the BOOT8BIT bit is determined by Port E bit 0 pin on ARM7100 during power on reset. In this way, pulling Port E bit 0 either LOW or HIGH during power on reset allows ARM7100 to boot from either 32-bit wide or 8-bit wide ROMs.

Bus Width Field	BOOT8BIT	Expansion Transfer Mode	Port E bit 0 during power on reset
00	0	32-bit wide bus access	LOW
01	0	16-bit wide bus access	LOW
10	0	8-bit wide bus access	LOW
11	0	PCMCIA mode	LOW
00	1	8-bit wide bus access	HIGH
01	1	PCMCIA mode	HIGH
10	1	32-bit wide bus access	HIGH
11	1	16-bit wide bus access	HIGH

Table 8-4: Values of the bus width field

When the bus width field is programmed to PCMCIA mode the bus width and bus conversion is defined by the state of **A[27**] and **A[26]**. *Table 8-5: PCMCIA mode bus widths* on page 8-12 defines the bus width and bus conversion for values of **A[27]** and **A[26]**. Word bus conversion converts an ARM 32-bit word access into a series of byte or 16-bit accesses. A special case is 16-bit I/O accesses (**A[26]** and **A[27]** HIGH). In this case, 32-bit ARM word accesses are not converted into two 16-bit accesses. This is to allow individual 16-bit register access. In this mode **D[16:31]** will be invalid and the output expansion address bit 1 is selected by the value of **A[25]**. ARM7100 will always output 0 on expansion address bit 25, ie. in 16-bit I/O mode processor address bit 25 becomes **PCMIA** address bit 1, and **PCMIA** address bit 25 is 0 limiting the 16-bit I/O address space to 32 Mb.

Note *16-bit I/O accesses are not converted to 32-bit ARM word accesses. This means that D[16:31] will be invalid during ARM word accesses to this memory area.*

A26	A27	Bus Width	Word Bus Conversion	PCMCIA Memory Area
0	0	8 Bits	Yes	8-bit attribute memory access
1	0	16 Bits	Yes	16-bit common memory access
0	1	8 Bits	Yes	8-bit I/O access
1	1	16 Bits	No	16-bit I/O access

Table 8-5: PCMCIA mode bus widths

Table 8-7: Values of the page mode access wait state field defines the values of the Random access wait state field.

Value	No. Wait States	Required Random Access Speed (nSEC)
00	4	250
01	3	200
10	2	150
11	1	100

Table 8-6: Values of the random access wait state field

Table 8-7: Values of the page mode access wait state field defines the values of the page mode access wait state field.

Value	No. Wait States	Required Random Access Speed (nSEC)
00	3	150
01	2	120
10	1	80
11	0	40

Table 8-7: Values of the page mode access wait state field

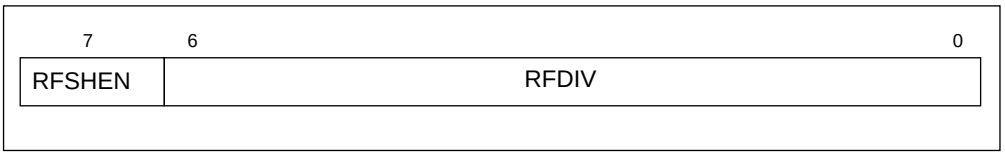
SQAEN — Sequential access enable. Setting this bit enables sequential accesses that are on a quad word boundary to take advantage of faster access times from devices that support page mode. The sequential access will be faulted after four words, (to allow video refresh cycles to occur) even if the access is part of a longer sequential access.

CLKEN — Expansion clock enable. Setting this bit enables the **EXPCLK** to be active during accesses to the selected expansion device. This provides a timing reference for devices that need to extend bus cycles using the **EXPRDY** input. Back to back (but not necessarily page mode) accesses result in a continuous clock.

For more details on bus timing, refer to *Chapter 20, DC and AC Parameters*.

8.3.15 DRAM refresh period register (DRFPR)

The DRAM refresh period register is an 8-bit read-write register which enables refresh and selects the refresh period used by the DRAM controller for its periodic CAS before RAS refresh. The value in the DRAM refresh period register is only cleared by a *power on reset*. Its state is maintained during a power fail or user reset.



RFSHEN — DRAM refresh enable. Setting this bit enables periodic refresh cycles to be generated by ARM7100 at a rate set by the RFDIV field. Setting this bit also enables self refresh mode when ARM7100 is in the standby state.

RFDIV — This 7-bit field sets the DRAM refresh rate. The refresh period is derived from a 128 KHz clock and is given by the following formula:

Frequency (KHz) = 128/(RFDIV + 1)

or

RFDIV = (128/Refresh frequency (KHz)) - 1

The maximum refresh frequency is 64 KHz, the minimum is 1KHz. The RFDIV field should not be programmed with zero as this results in no refresh cycles being initiated.

8.3.16 Interrupt status register (INTSR)

The interrupt status register is a 16-bit read only register. It reflects the current state of the 16 interrupt sources within ARM7100. Each bit is set if the appropriate interrupt is active. The interrupt assignment is given in ❍*Figure 8-6: Interrupt Assignment*.

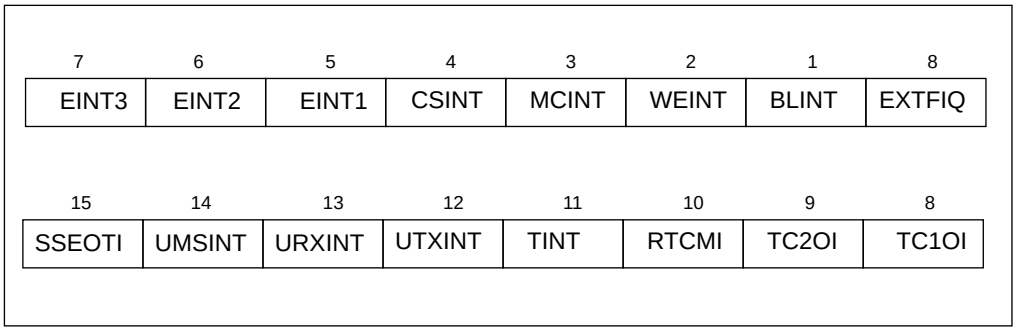


Figure 8-6: Interrupt Assignment

EXTFIQ — The external fast interrupt is active if the **nEXTFIQ** input pin is forced LOW and is mapped to the **nFIQ** input on the ARM7 processor.

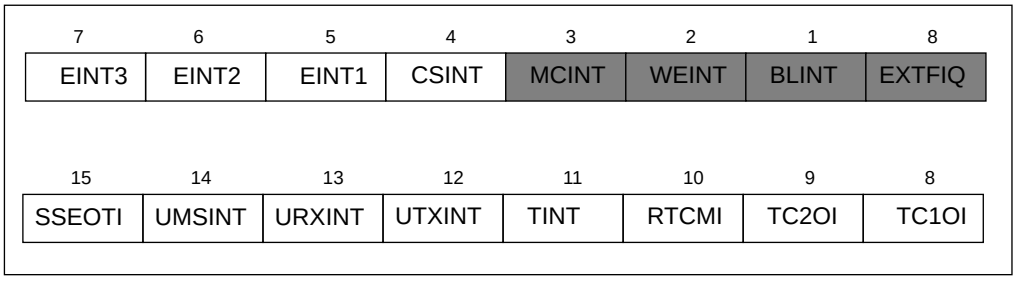
BLINT — Battery low interrupt is active if no external supply is present (**nEXTPWR** is HIGH) and the battery OK input pin **BATOK** is forced LOW. This interrupt is de-glitched with a 16 KHz clock so only generates an interrupt if it is active for longer than 62.5 mSec. It is mapped to the **nFIQ** input on the ARM7 processor and is cleared by writing to the BLEOI location.

WEINT	Watch dog expired interrupt becomes active on a rising edge of the periodic 64 Hz tick interrupt clock if the tick interrupt is still active, ie. if a tick interrupt has not been serviced for a complete tick period. It is cleared by writing to the TEOI location.
MCINT	This interrupt is active after a rising edge on the **MEDCHG** input pin has been detected, This input is de-glitched with a 16 KHz clock so only generates an interrupt if it is active for longer than 62.5 mSec. It is mapped to the **FIQ** input on the ARM7 processor and is cleared by writing to the MCEOI location.
CSINT	The CODEC sound interrupt is active if the CODEC interface is enabled and the CODEC data FIFO has reached half full or empty (depending on the interface direction). It is cleared by writing to the COEOI location.
EINT1	The external interrupt input 1 is active if the **nEINT1** input is active (LOW). It is cleared by returning **nEINT1** to the passive (HIGH) state.
EINT2	The external interrupt input 2 is active if the **NEINT2** input is active (LOW). It is cleared by returning **nEINT2** to the passive (HIGH) state.
EINT3	External interrupt input 3. This input will be active if the **EINT3** input is active (HIGH). It is cleared by returning **EINT3** to the passive (LOW) state.
TC1OI	The TC1 under flow interrupt becomes active on the next rising edge of the timer counter 1 clock after the timer counter has under flowed (reached zero). It is cleared by writing to the TC1EOI location.
TC2OI	The TC2 under flow interrupt becomes active on the next rising edge of the timer counter 2 clock after the timer counter has under flowed (reached zero). It is cleared by writing to the TC2EOI location.
RTCMI	The RTC compare match interrupt becomes active on the next rising edge of the 1Hz real time clock (one second later) after the 32-bit time written to the real time clock match register exactly matches the current time in the RTC. It is cleared by writing to the RTCEOI location.
TINT	64 Hz tick interrupt. This interrupt becomes active on every rising edge of the internal 64Hz clock signal. This 64 Hz clock is derived from the 15 stage ripple counter that divides the 32.768 KHz oscillator input down to 1Hz for the real time clock. This interrupt is cleared by writing to the TEOI location.

UTXINT	Internal UART transmit FIFO empty interrupt. The function of this interrupt source depends on whether the UART FIFO is enabled. If the FIFO is disabled (FIFOEN bit is clear in the UART bit rate and line control register), this interrupt is active when there is no data in the UART Tx data holding register. It is cleared by writing to the UART data register. If the FIFO is enabled, this interrupt is active when the UART Tx FIFO is half or more empty, and is cleared by filling the FIFO to at least half full.
URXINT	Internal UART receive FIFO full interrupt. The function of this interrupt source depends on whether the UART FIFO is enabled. If the FIFO is disabled this interrupt is active when there is valid Rx data in the UART Rx data holding register. It is cleared by reading this data. If the FIFO is enabled this interrupt is active when the UART Rx FIFO is half or more full or if the FIFO is non empty and no more characters have been received for a three character time out period. It is cleared by reading all the data from the Rx FIFO.
UMSINT	Internal UART modem status changed interrupt. This interrupt will be active if either of the two modem status lines (**CTS** or **DSR**) change state. It is cleared by writing to the UMSEOI location.
SSEOTI	Synchronous serial interface end of transfer interrupt. This interrupt is active after a complete data transfer to and from the external ADC has completed. It is cleared by reading the ADC data from the SYNCIO register.

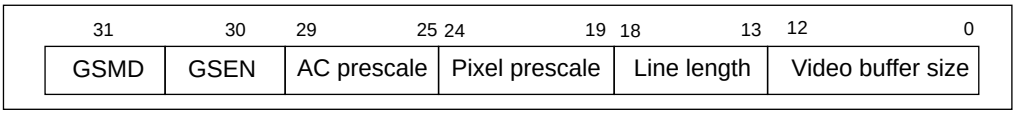
8.3.17 Interrupt mask register (INTMR)

The interrupt mask register is a 16-bit read-write register which is used to enable any of the 16 interrupt sources selectively within ARM7100. The four shaded interrupts all generate a fast interrupt request to the ARM7 processor. This causes a jump to processor virtual address 0000.0001C. All other interrupts generate a standard interrupt request causing a jump to processor virtual address 0000.00018. See *Table 9-1: Interrupt allocation* on page 9-3 for the interrupt allocation. Setting the appropriate bit in this register enables the corresponding interrupt. All bits are cleared by a *system reset.*



8.3.18 The LCD control register (LCDCON)

The LCD control register is a 32-bit read-write register which controls the size of the LCD screen and the mode in which the LCD controller operates. Refer to the system description of the LCD controller for more information on video buffer mapping.



Video buffer size — The video buffer size field is a 13-bit field that sets the total number of bytes (*128 quad words) in the video display buffer. This is calculated from the following formula:

```
Video buffer size = (Total bytes in video buffer/128) - 1
```

For example, for a 640 x 240 LCD and 4 bits per pixel the size of the video buffer = 640 x 240 x 4 = 614400 bits.
video buffer size field = (614400/128)-1
= 4799 or 0x12BF Hex

Line length — The line length field is a 6-bit field that sets the number of pixels in one complete line. This field is calculated from the formula:

```
Line length  = ( No. pixels in line/16 ) - 1
```

For example:
640 x 240 LCD line length = (640/16)-1
= 39 or 0x27 Hex

Pixel prescale — The pixel prescale field is a 6-bit number that sets the pixel rate prescale. The pixel rate is derived from a 36.864 MHz clock and is calculated from the following formula:

```
Pixel rate (MHz) = 36.864/ (Pixel prescale + 1)
```

The pixel rate should be chosen to give a complete screen refresh frequency of approximately 70 Hz to avoid flicker. Frequencies above 70 Hz should be avoided as this consumes additional power. The pixel prescale value can be expressed in terms of the LCD size by the following formula:

```
Pixel prescale = (526628/Total pixels in display)-1
```

The value should be rounded down to the nearest whole number, and zero is illegal and results in no pixel clock. For example:

640 x 240 LCD pixel prescale = 526628/(640x240)-1
= 2.428(2)

Actual pixel rate = 36.864E6/2+1
=12.288MHz

Actual refresh frequency = 12.288E6/(640x240)
= 80Hz

As the CL2 low pulse time is doubled after every CL1 high pulse, this refresh frequency is only an approximation, the accurate formula is:

12.288E6/((640x240)+120) =79.937Hz.

AC prescale — The AC prescale field is a 5-bit number that sets LCD AC bias frequency. This frequency is the required AC bias frequency for a given manufacturer's LCD plate. It is derived from the frequency of the line clock (CL1).

The **M** signal toggles after n + 1 counts of the line clock (CL1) where n is the number programmed into the AC prescale field. This number must be chosen to match the manufacturers recommendation. This is normally 13 but must not be exactly divisible by the number of lines in the display.

GSEN — Grey scale enable bit. Setting this bit enables grey scale output to the LCD. When it is cleared, each bit in the video map directly corresponds to a pixel in the display.

GSMD — Grey scale mode bit. Clearing this bit sets the controller to 2 bits per pixel (4 grey scales). Setting it sets it to 4 bits per pixel (15 grey scales).

8.3.19 Timer counter 1 data register (TC1D)

The timer counter 1 data register is a 16-bit read-write register which sets and reads data to TC1. Any value written will be decremented on the next rising edge of the clock.

8.3.20 Timer counter 2 data register (TC2D)

The timer counter 2 data register is a 16-bit read-write register which sets and reads data to TC2. Any value written will be decremented on the next rising edge of the clock.

8.3.21 Real time clock data register (RTCDR)

The real time clock data register is a 32-bit read-write register which sets and reads the binary time in the RTC. Any value written will be incremented on the next rising edge of the 1 Hz clock.

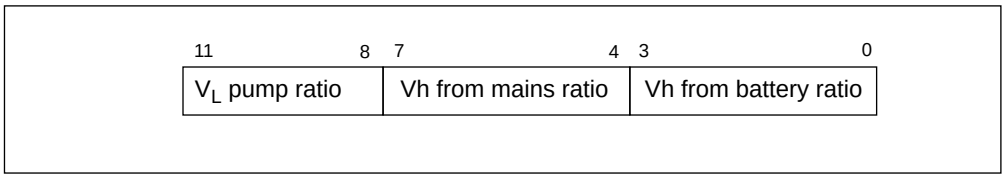
All bits in the real time clock data register are only cleared by an active **nPOR**.

8.3.22 Real time clock match register (RTCMR)

The real time clock match register is a 32-bit read-write register which sets and reads the binary match time to RTC. Any value written will be compared to the current binary time in the RTC, if they match it will assert the RTCMI interrupt source.

8.3.23 Pump control register (PMPCON)

The DC to DC converter pump control register is a 12-bit read-write only register which sets and controls the variable mark space ratio drives for two DC to DC converters. All bits in this register are cleared by a *system reset.*



Vh from battery — This 4-bit field controls the on time for the DC to DC pump for a Vh rail while the **nEXTPWR** input is HIGH. Setting these bits to 0 disables this pump. Setting them to 1 allows the pump to be driven in a 1:16 duty ratio, 2 in a 2:16 duty ratio etc. up to a 15:16 duty ratio. An 8:16 duty ratio results in a square wave of 96 KHz.

Vh from mains — This 4-bit field controls the on time for the DC to DC pump for a Vh rail while the **nEXTPWR** input is LOW. Setting these bits to 0 disables this pump. Setting them to 1 allows the pump to be driven in a 1:16 duty ratio, 2 in a 2:16 duty ratio etc. up to a 15:16 duty ratio. An 8:16 duty ratio results in a square wave of 96 KHz.

V_L pump ratio — This 4-bit field controls the on time for the DC to DC pump for the V_L voltage rail. Setting these bits to 0 disables this pump. Setting them to 1 allows the pump to be driven in a 1:16 duty ratio, 2 in a 2:16 duty ratio etc. up to a 15:16 duty ratio. An 8:16 duty ratio results in a square wave of 96 KHz. The state of the output drive pin (drive 1) is latched during power on reset, this latched value is used to determine the polarity of the bias voltage. The sense of the DC to DC converter control lines is summarised in *Table 8-8: Sense of DC to DC Converter Control Lines*.

Initial State of Drive n during POR	Sense of Drive n	Polarity of Bias Voltage
LOW	Active HIGH	+ve
HIGH	Active LOW	-ve

Table 8-8: Sense of DC to DC Converter Control Lines

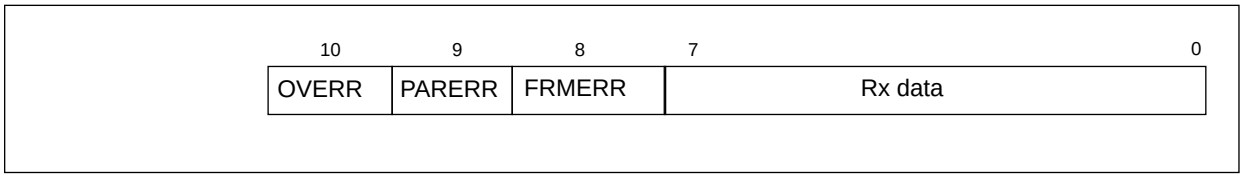
8.3.24 The CODEC interface data register (CODR)

The CODR register is an 8-bit read-write register. Data written to or read from this register is pushed or popped onto a 16 byte FIFO buffer. Data from this buffer is then serialised and sent to or received from the CODEC sound device. The CODEC interrupt CSINT is generated repetitively at 1/8th of the byte transfer rate and the state of the FIFOs can be read in the system flags register. The net data transfer rate to or from the CODEC device is 8 Kb per second giving an interrupt rate of 1 KHz.

8.3.25 UART data register (UARTDR)

The UARTDR register is an 11-bit read and an 8-bit write register for all data transfers to or from the internal UART. Data written to this register is pushed onto the 16-byte data Tx holding FIFO if the FIFO is enabled, or stored in a one byte holding register. This write initiates transmission from the UART. The UART data read register is made up of the 8-bit data byte received from the UART together with three bits of error status. Data read from this register is popped from the 16 byte data Rx FIFO if the FIFO is enabled, or read from a one byte buffer register containing the last byte received and error status if not enabled. Data received by the UART is automatically pushed onto the Rx FIFO if it is enabled.

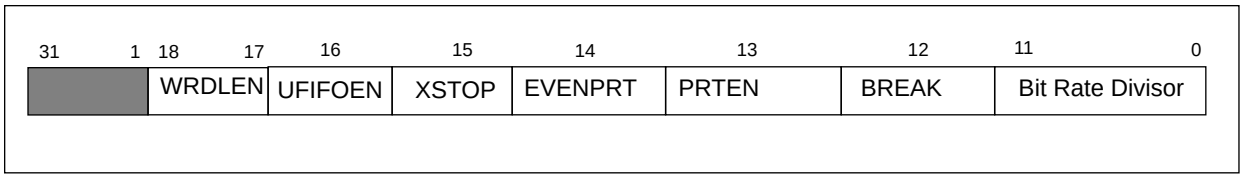
The Rx FIFO is 10 bits wide by 16 deep.



FRMERR	UART framing error. This bit is set if the UART detected an overrun or framing error while receiving the Rx data byte.
PARERR	UART parity error. This bit is set if the UART detected a parity error while receiving the Rx data byte.
OVERR	UART overrun error. This bit is set if more data is received by the UART and the FIFO is full. The overrun error bit is not associated with any single character and so is not stored in the FIFO. If this bit is set, the entire contents of the FIFO is invalid and should be cleared. This error bit is cleared by reading the UARTDR register.

8.3.26 UART bit rate and line control register (UBRLCR)

The bit rate divisor and line control register is a 19-bit read-write register. Writing to this register sets the bit rate and mode of operation for the internal UART.



Bit rate divisor	This 12-bit field sets the bit rate. The bit rate divider is fed by a clock frequency of 3.6864 MHz. It is then further divided internally by 16 to give the bit rate. The following formula gives the divisor value for any bit rate: `Divisor = (230400/bit rate)-1` A value of zero in this field is illegal.

Table 8-9: Internal UART Bit Rates shows some example bit rates with the corresponding divisor value.

Divisor Value	Bit Rate
1	115200
2	76800
3	57600
5	38400
11	19200
15	14400
23	9600
95	2400
191	1200
2094	110

Table 8-9: Internal UART Bit Rates

BREAK — Setting this bit drives the Tx output active (HIGH) to generate a break.

PRTEN — Parity enable bit. Setting this bit enables parity detection and generation.

EVENPRT — Even parity bit. Setting this bit sets parity generation and checking to even parity, clearing it sets odd parity. This bit has no effect if the PRTEN bit is clear.

XSTOP — Extra stop bit. Setting this bit will cause the UART to transmit two stop bits. Clearing it sets one stop bit after each data byte.

FIFOEN — Set to enable FIFO buffering of Rx and Tx data. Clear to disable the FIFO, ie. set its depth to one byte.

WRDLEN — This two bit field selects the word length according to *Table 8-10: UART word length* on page 8-21.

WRDLEN	Word length
00	5 bits
01	6 bits
10	7 bits
11	8 bits

Table 8-10: UART word length

8.3.27 Least significant word - LCD palette register (PALLSW)

The least and most significant word LCD palette registers make up a 64-bit read-write register which maps the logical pixel value to a physical grey scale level. The 64-bit register is made up of 16 4-bit nibbles, each nibble defining the grey scale level associated with the appropriate pixel value. If the LCD controller is operating in two bits per pixel, only the lower 4 nibbles are valid (D[15:0] in the least significant word). Similarly one bit per pixel means only the lower 2 nibbles are valid (D[7:0] in the least significant word). The pixel to grey scale level assignments are shown in ◗*Figure 8-7: Least significant word palette assignments* and ◗*Figure 8-8: Most significant word palette assignments*.

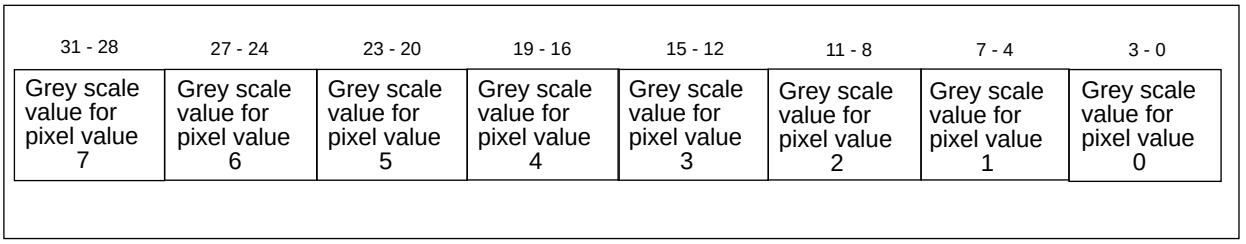


Figure 8-7: Least significant word palette assignments

8.3.28 Most significant word - LCD palette register (PALMSW)

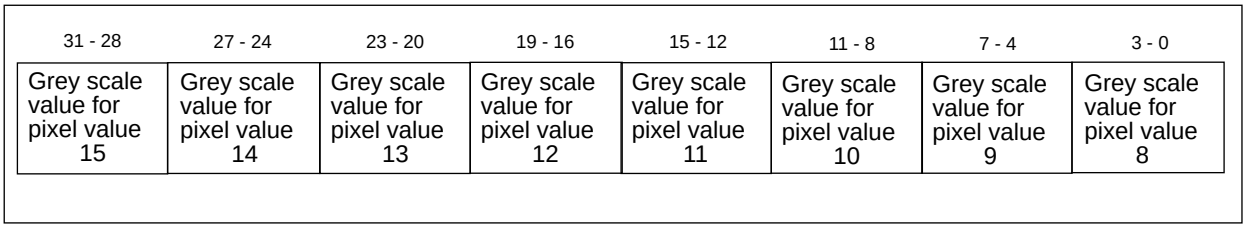


Figure 8-8: Most significant word palette assignments

The actual physical colour and pixel duty ratio for the grey scale values is shown in ◗*Table 8-11: Grey scale value to colour mapping*. Note that colours 8-15 are the inverse of colours 7-0 respectively. This means that colours 7 and 8 are identical. The steps in the grey scale are non linear but have been chosen to give a close approximation to perceived linear grey scales. This is due to the eye being more sensitive to changes in grey level close to 50% grey.

Grey scale value	Duty cycle	% pixels lit
0	0	0
1	1/9	11.1
2	1/5	20
3	4/15	26.7
4	3/9	33.3
5	2/5	40
6	4/9	44.4
7	1/2	50
8	1/2	50
9	5/9	55.6
10	3/5	60
11	6/9	66.7
12	11/15	73.3
13	4/5	80
14	8/9	88.9
15	1	100

Table 8-11: Grey scale value to colour mapping

8.3.29 Synchronous serial interface data register (SYNCIO)

SYNCIO is a 16-bit read-write register. The byte written to the SYNCIO register will be serialised and transmitted out of the synchronous serial interface. The clock will automatically be started at the programmed frequency and a synchronisation pulse will be issued. The **ADCIN** pin is sampled on every clock edge and the result is shifted in the SYNCIO read register.

During data transfer the SSIBUSY bit is set HIGH, at the end of a transfer the SSEOTI interrupt will be asserted. This interrupt is cleared by reading the SYNCIO register.

The data read from the SYNCIO register will be the last 16 bits shifted out of the ADC. The length of the data frame can be programmed by writing to the SYNCIO register allowing many different ADCs to be accommodated. ◔*Figure 8-9: Bits in SYNCIO write register* defines the bits in the SYNCIO register.

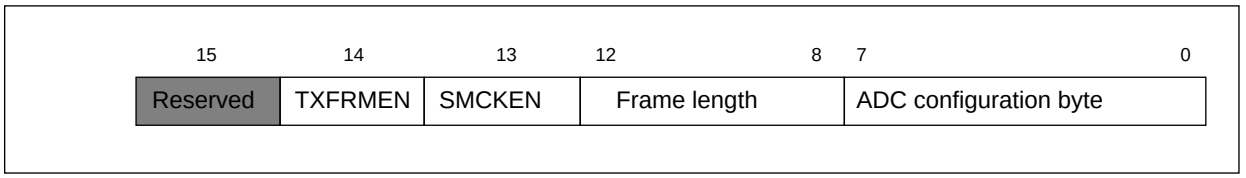


Figure 8-9: Bits in SYNCIO write register

ADC configuration	8-bit configuration data to be sent to the ADC.
Frame length	The 5-bit frame length field is the total number of shift clocks required to complete a data transfer. For many ADCs this is 25, 8 for configuration byte + 1 null bit + 16-bit result.
SMCKEN	Setting this bit will enable a free running sample clock at the programmed ADC clock frequency to be output on the **SMPLCK** pin.
TXFRMEN	Setting this bit causes an ADC data transfer to be initiated. The value in the ADC field will be shifted out to the ADC and depending on the frame length programmed, a number of bits will be captured from the ADC. If the SYNCIO register is written to with the TXFRMEN bit LOW, no ADC transfer will take place but the frame length and SMCKEN bits will be affected.

8.3.30 Clear all start up reason flags location (STFCLR)

A write to this location clears all the start up reason flags in the system flags status register SYSFLG.

8.3.31 Battery low end of interrupt (BLEOI)

A write to this location clears the interrupt generated by a low battery (falling **BATOK** with **nEXTPWR** HIGH).

8.3.32 Media changed end of interrupt (MCEOI)

A write to this location clears the interrupt generated by a rising edge of the **MEDCHG** input pin.

8.3.33 Tick end of interrupt location (TEOI)

A write to this location clears the current pending tick interrupt and watchdog interrupt.

8.3.34 TC1 end of interrupt location (TC1EOI)

A write to this location clears the under flow interrupt generated by TC1.

8.3.35 TC2 end of interrupt location (TC2EOI)

A write to this location clears the under flow interrupt generated by TC2.

8.3.36 RTC match end of interrupt (RTCEOI)

A write to this location clears the RTC match interrupt.

8.3.37 UART modem status changed end of interrupt (UMSEOI)

A write to this location clears the modem status changed interrupt.

8.3.38 CODEC end of interrupt location (COEOI)

A write to this location clears the sound interrupt (**CSINT**).

8.3.39 Enter idle state location (HALT)

A write to this location puts the system into the *idle* state by halting the clock to the processor until an interrupt is generated.

Note If the idle state is entered with no interrupts enabled, there is no mechanism for exiting the idle state except for a system reset.

8.3.40 Enter standby state location (STDBY)

A write to this location puts the system into the *standby* state by halting the main oscillator. It will automatically switch the DRAM's to self refresh if the RFSHEN bit is set in the DRAM refresh period register. All transitions to the *standby* state are synchronised with the DRAM cycles.

9 Interrupt Controller

This chapter describes the interrupt controller.

9.1 Interrupt Controller 9-2

Preliminary

9.1 Interrupt Controller

The ARM 710a has two interrupt types:

- interrupt request (IRQ)
- fast interrupt request (FIQ)

The interrupt controller in ARM7100 controls interrupts from 16 different sources. Twelve interrupt sources are mapped to the **IRQ** input and four sources to the **FIQ** input. FIQs have a higher priority than IRQs and if two interrupts at the same priority are active, the priority they are serviced in must be resolved in software.

All interrupts are *level sensitive*, ie. they must conform to the following sequence:

1. The device asserts the appropriate interrupt request line.
2. If the appropriate bit is set in the interrupt mask register, either FIQ or IRQ is asserted by the interrupt controller.
3. If interrupts are enabled, the processor jumps to the appropriate vector.
4. Interrupt despatch software reads the interrupt control and status register to establish the source(s) of the interrupt and calls the appropriate interrupt service routine(s).
5. Software in the interrupt service routine clears the interrupt source by an action specific to the device requesting the interrupt, eg. reading the UART Rx register.
6. The interrupt service routine may then re-enable interrupts and any other pending interrupts will be serviced in a similar way, or return to the interrupt dispatch code which can check for any more pending interrupts and dispatch them accordingly.

See ➲*Chapter 8, ARM7100 Programmer's Model* for details of interrupt registers.

Table 9-1: Interrupt allocation shows the names and allocation of interrupts in ARM7100.

Interrupt	Bit in Mask and ISR	Name	Comment
FIQ	0	EXTFIQ	External fast interrupt input
FIQ	1	BLINT	Battery low interrupt
FIQ	2	WEINT	Watch dog expired interrupt
FIQ	3	MCINT	MEDCHG interrupt
IRQ	4	CSINT	CODEC sound interrupt
IRQ	5	EINT1	External interrupt input 1
IRQ	6	EINT2	External interrupt input 2
IRQ	7	EINT3	External interrupt input 3
IRQ	8	TC1OI	TC1 under flow interrupt
IRQ	9	TC2OI	TC2 under flow interrupt
IRQ	10	RTCMI	RTC compare match interrupt
IRQ	11	TINT	64 Hz tick interrupt
IRQ	12	UTXINT	Internal UART transmit FIFO empty interrupt
IRQ	13	URXINT	Internal UART receive FIFO full interrupt
IRQ	14	UMSINT	Internal UART modem status changed interrupt
IRQ	15	SSEOTI	Synchronous serial interface end of transfer interrupt

Table 9-1: Interrupt allocation

10 The Expansion and ROM Interface

This chapter describes the ROM Interface.

10.1 The Expansion and ROM Interface 10-2

Preliminary

10.1 The Expansion and ROM Interface

Eight separate linear memory or expansion segments are decoded by the ARM7100. Each segment is 256 Mb and can be interfaced to a conventional SRAM-like interface.

Each segment can be programmed individually to:

- be 8, 16 or 32 bits wide
- support page mode access
- execute from 0 to 4 wait states.

In addition, bus cycles can be extended using the **EXPRDY** input signal.

Page mode access is accomplished by running up to four accesses together. This can significantly improve bus bandwidth to devices such as ROMs. Sequential burst mode access is always faulted (the bus returned to idle) after four *accesses* regardless of bus width to allow DMA and refresh cycles.

See ➲*Chapter 8, ARM7100 Programmer's Model* for details of the expansion and ROM interface registers.

11 DRAM controller

This chapter describes the DRAM controller.

11.1 DRAM Controller 11-2

Preliminary

11.1 DRAM Controller

The DRAM controller in ARM7100 provides connections allowing a direct interface to up to four banks of DRAM. Each bank is 32 bits wide and up to 256 Mb in size. Four RAS lines are provided (one per bank) and four CAS lines (one per byte line). The DRAM device size is not programmable if devices smaller than the largest size supported (1 Gbit) are used. This leads to a segmented memory map with each bank separated by 256 MBytes. Segments that are smaller than the bank size will repeat within the bank.

Table 11-1: Physical to DRAM address mapping shows the mapping of physical address to DRAM row and column address. This mapping has been organised to support any DRAM device size from 4 Mbit to 1 Gbit with a square row and column configuration, ie. the number of column addresses is equal to the number of row addresses. If a non-square DRAM is used, further fragmentation of the memory map will occur. However the smallest contiguous segment will always be 1 Mb.

Memory Address	DRAM Column	DRAM Row	Pin Name
0	A2	A10	A[27]/DRA[0]
1	A3	A11	A[26]/DRA[1]
2	A4	A12	A[25]/DRA[2]
3	A5	A13	A[24]/DRA[3]
4	A6	A14	A[23]/DRA[4]
5	A7	A15	A[22]/DRA[5]
6	A8	A16	A[21]/DRA[6]
7	A9	A17	A[20]/DRA[7]
8	A19	A18	A[19]/DRA[8]
9	A21	A20	A[18]/DRA[9]
10	A23	A22	A[17]/DRA[10]
11	A25	A24	A[16]/DRA[11]
12	A27	A26	A[15]/DRA[12]

Table 11-1: Physical to DRAM address mapping

Table 11-2: DRAM address mapping shows the address mapping for various DRAMs with square and non-square row and address inputs, assuming two x16 devices are connected to each RAS line. This mapping is repeated every 256 Mb for each DRAM bank. n is given by n = 0xC + bank number, eg. 0 for bank 0.

Device Size	Address Configuration	Total Size of Bank	Address Range of Segment(s)	Size of Segment(s)
4 Mbit	9 Row x 9 Column	1 Mbyte	n000.0000 - n00F.FFFF	1 Mbyte
16 Mbit	10 Row x 10 Column	4 MBytes	n000.0000 - n03F.FFFF	4 MBytes
16 Mbit	12 Row x 8 Column	4 MBytes	n000.0000 - n003.FFFF n010.0000 - n013.FFFF n040.0000 - n043.FFFF n050.0000 - n053.FFFF n100.0000 - n103.FFFF n110.0000 - n113.FFFF n140.0000 - n143.FFFF n150.0000 - n153.FFFF n400.0000 - n403.FFFF n410.0000 - n413.FFFF n440.0000 - n443.FFFF n450.0000 - n453.FFFF n500.0000 - n503.FFFF n510.0000 - n513.FFFF n540.0000 - n543.FFFF n550.0000 - n553.FFFF	256 KBytes 256 KBytes 256 KBytes 256 KBytes 256 KBytes 256 KBytes 256 KBytes 256 KBytes 256 KBytes 256 KBytes 256 KBytes 256 KBytes 256 KBytes 256 KBytes 256 KBytes 256 KBytes
64 Mbit	11 Row x 11 Column	16 MBytes	n000.0000 - n0FF.FFFF	16 MBytes
64 Mbit	13 Row x 9 Column	16 MBytes	n000.0000 - n00F.FFFF n020.0000 - n02F.FFFF n080.0000 - n08F.FFFF n0A0.0000 - n0AF.FFFF n200.0000 - n20F.FFFF n220.0000 - n22F.FFFF n280.0000 - n28F.FFFF n2A0.0000 - n2AF.FFFF n800.0000 - n80F.FFFF n820.0000 - n82F.FFFF n880.0000 - n88F.FFFF n8A0.0000 - n8AF.FFFF nA00.0000 - nA0F.FFFF nA20.0000 - nA2F.FFFF nA80.0000 - nA8F.FFFF nAA0.0000 - nAAF.FFFF	1 MByte 1 MByte 1 MByte 1 MByte 1 MByte 1 MByte 1 MByte 1 MByte 1 MByte 1 MByte 1 MByte 1 MByte 1 MByte 1 MByte 1 MByte 1 MByte
256 Mbit	12 Row x 12 Column	64 MBytes	n000.0000 - n3FF.FFFF	64 MBytes
1 Gbit	13 Row x 13 Column	256 MBytes	n000.0000 - nFFF.FFFF	256 MBytes

Table 11-2: DRAM address mapping

The DRAM controller contains a programmable refresh counter. The refresh rate is controlled using the DRAM refresh period register (DRFPR).

12 CODEC Interface

This chapter describes the ARM7100 CODEC interface.

12.1 CODEC Interface 12-2

Preliminary

12.1 CODEC Interface

The CODEC interface allows direct connection of a telephony type CODEC to ARM7100. It provides all the necessary clocks and timing pulses and performs serialisation or visa versa of the data stream to or from the CODEC. The interface is full duplex and contains two separate data FIFOs.

Data is transferred to or from the CODEC at 64 k bits per second. This is either written to or read from a 16-byte FIFO. The sound interrupt is generated every 8 bytes that are transferred (FIFO half full/empty) which means the interrupt rate is reduced from 8 KHz to 1 KHz with a latency of 1 mSec.

See ◆*Chapter 8, ARM7100 Programmer's Model* for details of the CODEC interface registers.

13 Synchronous Serial Interface

This chapter describes the synchronous serial interface.

13.1 Synchronous Serial Interface 13-2

Preliminary

13.1 Synchronous Serial Interface

The synchronous serial interface provides a four wire interface to serial peripheral devices such as ADCs that have a SPI™ or Microwire™ compatible interface. The clock output frequency is programmable and only active during data transmissions to save power. The output channel is fed by an 8-bit shift register. The input channel is captured by a 16-bit shift register. The clock and synchronisation pulses are activated by a write to the output shift register. During transfers, the SSIBUSY (synchronous serial interface busy) bit in the system status flags register is set when the transfer is complete. Valid data is in the 16-bit read shift register when the SSEOTI interrupt is asserted and the SSIBUSY bit is cleared.

See *Chapter 8, ARM7100 Programmer's Model* for details of the synchronous serial interface registers.

14 LCD Controller

This chapter describes the LCD controller.

14.1 LCD Controller 14-2

Preliminary

14.1 LCD Controller

The LCD controller provides all the necessary control signals to interface directly to a single panel multiplexed LCD. The panel size is programmable and can be any width (line length) from 16 to 1024 pixels in 16 pixel increments. The number of lines is achieved by programming the total number of pixels in the LCD. The total video frame size is prgrammable up to 128Kb equating to a theoretical maximum panel size of 640 x 409 or 1024 x 256 pixels.

The video RAM is mapped into the base of the main DRAM memory area which is fixed at the physical address 0xC000.0000. The number of bits per pixel is programmable from 1 to 4.

The screen is mapped to the video buffer as one contiguous block where each horizontal line of pixels is mapped to a set of consecutive bytes or words in the video RAM. The video buffer can be accessed word wide as pixel 0 is mapped to the LSB in the buffer ie. the pixels are arranged in a little-endian manner.

The pixel bit rate and hence the LCD refresh rate can be programmed from 18.432MHz to 576KHz. The LCD controller is programmed by writing to the LCD control register (LCDCON).

The LCD controller also contains two 32-bit palette registers. These allow any 4, 2 or 1 bit pixel value to be mapped to any of the 15 grey scale values available.

Figure 14-1: Video Buffer Mapping on page 14-3 shows the organisation of the video map for all combinations of bits per pixel.

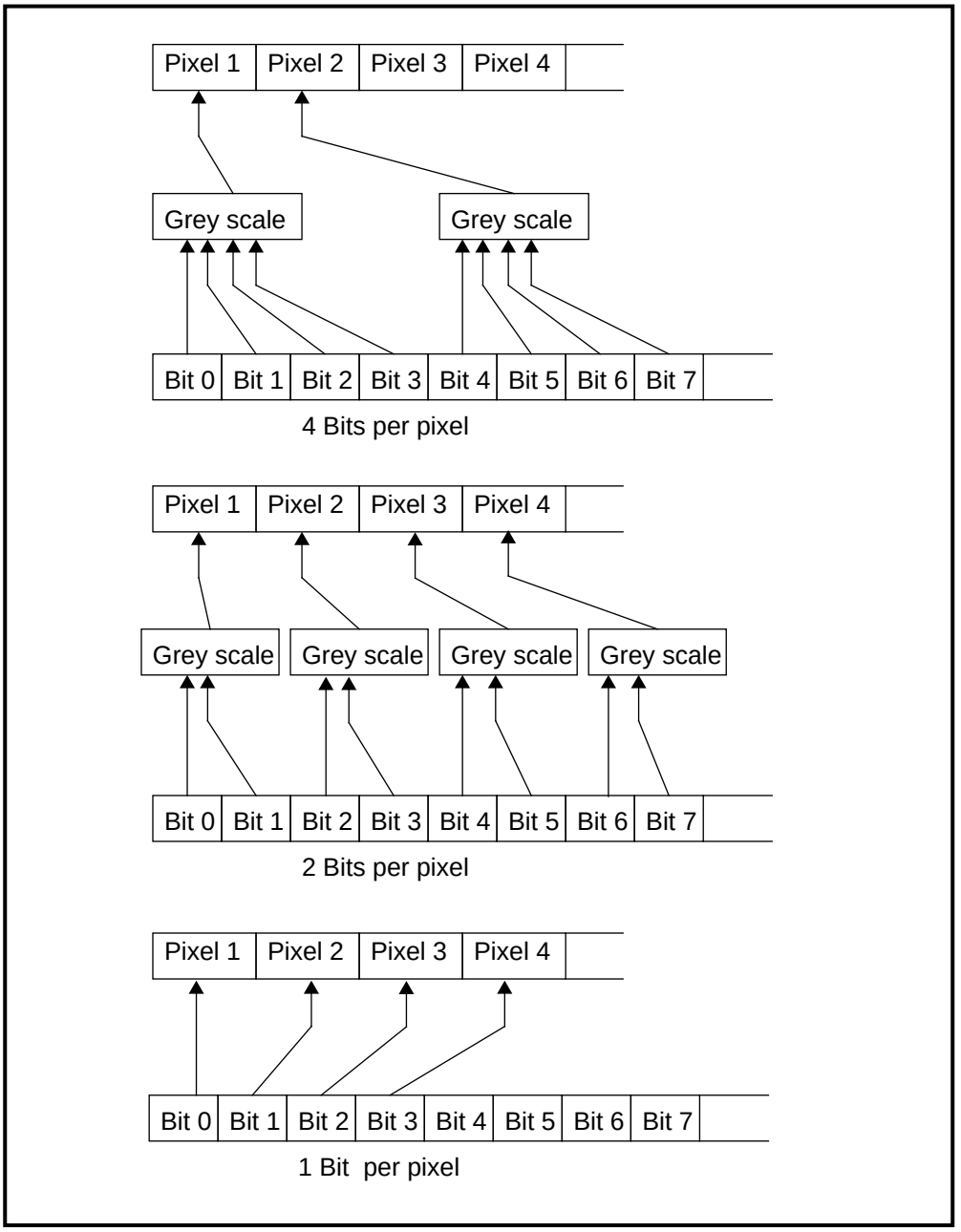


Figure 14-1: Video Buffer Mapping

The refresh rate is not affected by the number of bits per pixel . However, the LCD controller fetches twice the data per refresh for 4 bits per pixel compared to 2 bits per pixel. The main reason for reducing the number of bits per pixel is to reduce the power consumption of the DRAMs in bank 0 where the video buffer is mapped.

See ↻*Chapter 8, ARM7100 Programmer's Model* for details of the LCD controller registers.

15 UART and SiR Encoder

This chapter the UART and the SiR Encoder.

15.1 UART 15-2

15.2 SiR Encoder 15-2

Preliminary

15.1 UART

ARM7100 contains a built in UART which offers similar functionality to National Semiconductor's 16C550 device. It can support bit rates of up to 115.2 K bps and contains two 16 byte FIFOs for receive and transmit.

Only three MODEM control input signals, **CTS**, **DSR** and **DCD** are supported. The additional **RI** input MODEM control line is not supported. Output Modem control lines such as **RTS** and **DTR** are not explicitly supported but can be implemented using bits from the general purpose I/O ports in ARM7100.

UART operation and line speed are controlled by the UART bit rate and line control register (UBRLCR). Three interrupts can be generated by the UART:

Rx	is asserted when the FIFO becomes half full or if the FIFO is non empty for longer than 3 character length times with no more characters being received
Tx	is asserted if the FIFO buffer reaches half empty
Modem status	is generated if either of the modem status bits change state

Framing and parity errors are detected as each byte is received and pushed onto the Rx FIFO. An overrun error generates an Rx interrupt immediately. All error bits can be read from the 11-bit wide data register. The FIFO can also be programmed to be one byte depth only, like a conventional UART with double buffering.

15.2 SiR Encoder

ARM7100 also contains a IrDA (Infra-red data association) SiR protocol encoder. Optionally, this encoder can be switched in to the Tx and Rx signals so they can be used to drive an infra-red interface directly. For more details on the IrDA SiR protocol, see the appropriate document detailing this protocol standard. If the SiR protocol encoder is enabled, the UART Tx line is held in the passive state and transitions of the modem status or the Rx line will have no effect.

See ↻*Chapter 8, ARM7100 Programmer's Model* for details of the UART and SiR encoder registers.

16 Timer Counters

This chapter describes the timer counters and the real time clock.

16.1 Timer Counters 16-2

16.2 Real Time Clock 16-2

Preliminary

16.1 Timer Counters

Two timer counters are integrated in ARM7100. They are identical and are referred to as TC1 and TC2. TC1 and TC2 each have an associated 16-bit read/write data register and some control bits in the system control register.

The timer counters can be read at any time and each counter is loaded with the value written to the data register immediately. This value is *decremented* on the second clock edge to arrive after the write. When the timer counter under flows ie. reaches 0, it asserts its appropriate interrupt. The timers can be read at any time. The clock source and mode is selectable by writing to various bits in the system control register. Clock sources are 512KHz and 2KHz.

The timer counters can operate in two modes:

- Free running mode
- Pre-scale mode

Free running mode

In free running mode, the counter wraps round to 0xFFFF when it under flows and continues counting down. Any value written to TC1 or TC2 will be decremented on the second edge of the selected clock.

Prescale mode

In prescale mode the value written to TC1 or TC2 is automatically re-loaded when the counter under flows. Any value written to TC1 or TC2 will be decremented on the second edge of the selected clock. This mode can be used to produce a programmable frequency to drive the **BUZ** output or generate a periodic interrupt.

16.2 Real Time Clock

ARM7100 contains a 32-bit real time clock (RTC). This can be written to and read from in the same way as the timer counters, but it is 32 bits wide. The RTC is always clocked at 1 Hz. It also contains a 32-bit output match register which can be programmed to generate an interrupt when the time in the RTC matches a specific time written to this register.

See ↻*Chapter 8, ARM7100 Programmer's Model* for details of the timer counter registers.

17 DC to DC Converters

This chapter describes the two DC to DC Converter Interfaces.

17.1 DC to DC Converter Interfaces 17-2

Preliminary

17.1 DC to DC Converter Interfaces

ARM7100 has two programmable duty ratio 96 KHz clock outputs which are intended to be used as drives for DC to DC converters in the PSU subsystem. These clocks are enabled by external input pins which would normally be connected to the output from comparators monitoring the DC to DC converter output. The duty ratio (and hence the converter on time) can be programmed from 1 in 16 to 15 in 16. The sense of the DC to DC converter drive signal (active HIGH or LOW) is determined by latching the state of this drive signal during power on reset. ie. a pull up on the drive signal results in an active LOW drive output and vice versa. This allows either positive or negative voltages to be generated by the DC to DC converter.

18 Power Management and Reset

This chapter describes the power management states supported by ARM7100.

18.1 State Control 18-2

18.2 Reset 18-3

Preliminary

18.1 State Control

ARM7100 supports three basic power states:

Standby	This equates to a computer being switched off, that is, no display. The main oscillator is shut down.
Idle	The device is functioning and all oscillators are running but the processor clock is halted while it waits for an event such as a key press.
Operating	ARM7100 is fully operational.

In the standby state, all the system memory and state is maintained and the system time is kept up to date. The main oscillator is disabled and the system is static except for the low power watch crystal (32 KHz) oscillator and divider chain to the real time clock. The **RUN** signal is driven LOW when in the standby state.

When first powered or reset by the **nPOR** (not Power on reset) signal, the system is forced to the standby state. This is known as a cold reset and is the only completely asynchronous reset to ARM7100. The transition to the operating state is caused by one of the following:

- a rising edge on the wakeup input signal
- a selected interrupt being asserted

Once self refresh is enabled for the DRAMs, any transition to the standby state is synchronised to DRAM refresh cycles and forces all the DRAMs into self refresh mode.

Once in the operating state, the idle state is entered by writing to an internal memory location in ARM7100. Execution of the next instruction continues in the operating state if an interrupt becomes active. A write to another internal memory location causes the transition from the operating state to the standby state.

The system can also be forced into the standby state by hardware if the **nPWRFL** or **nURESET** inputs are forced LOW. In this case, the transition is synchronised with DRAM cycles to avoid any glitches or short cycles.

The system only transitions to the operating state from the standby state if the **nEXTPWR, BATOK** and **nPWRFL** inputs are HIGH. This prevents the system from attempting to start when the power supply is inadequate.

Figure 18-1: State diagram on page 18-3 shows a state diagram for ARM7100.

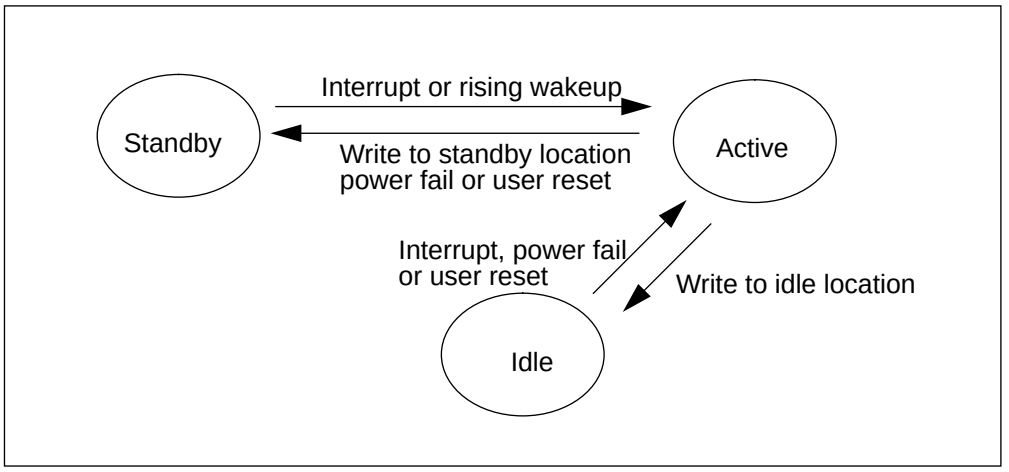


Figure 18-1: State diagram

18.2 Reset

There are three asynchronous resets to ARM7100; these are **nPOR**, **nPWRFL** and **nURESET**. If any of these are active a *system reset* is generated internally, this will clear all the internal registers in ARM7100 to zero except for the DRAM refresh period register and the real time clock data register which are only cleared by an active **nPOR**. It will also reset the ARM710a and cause it to start execution at the reset vector when ARM7100 returns to the operating state.

The **RUN** signal is HIGH when ARM7100 is in the operating or idle states and LOW when in the standby state.

19 Memory Map

This chapter describes the ARM7100 memory map.

19.1 Memory Map 19-2

Preliminary

19.1 Memory Map

The ARM7100 address space is allocated in the following way:

- The lower 2 Gb of the address space is allocated to ROM and expansion space.
- The upper 1 Gb of address space is allocated to DRAM.
- The remaining 1 Gb, less 4Kb for internal registers, is not accessible in ARM7100. The MMU should be programmed to cause an abort exception in this space.

Internal peripheral devices are communicated and configured through a set of internal memory locations from hex address 8000.000 to 8000.FFFF.

Figure 19-1: ARM7100 memory map on page 19-3 shows how the 4 Gb address range of the ARM710 processor is mapped in ARM7100.

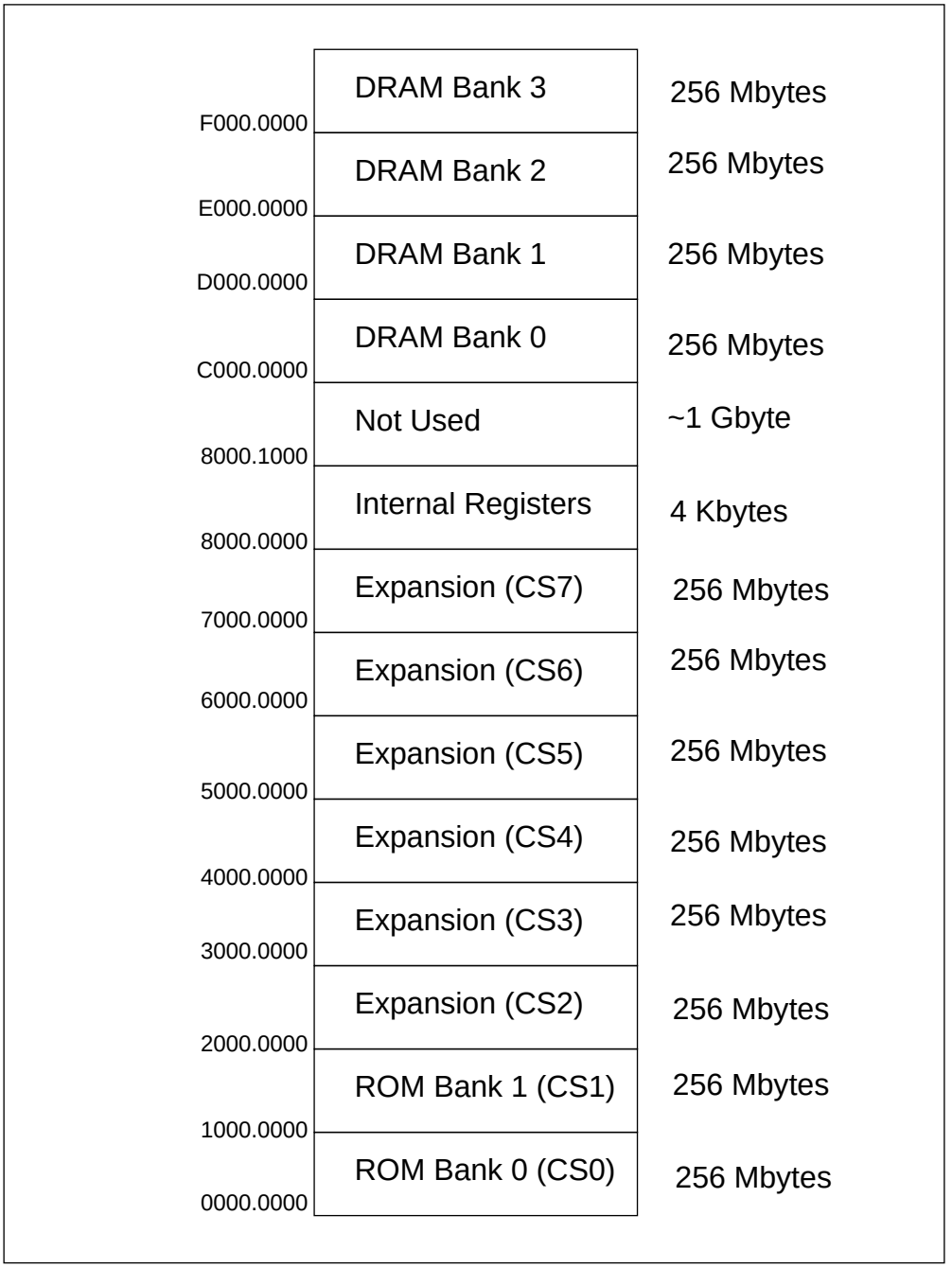


Figure 19-1: ARM7100 memory map

20 DC and AC Parameters

This chapter describes the DC and AC Parameters.

20.1 Absolute Maximum Ratings 20-2
20.2 DC Operating Conditions 20-2
20.3 DC Characteristics 20-3
20.4 AC Characteristics 20-5

Preliminary

20.1 Absolute Maximum Ratings

Parameters	Min	Max	Unit
DC Supply voltage	-0.5	+6	V
DC input / output voltage	-0.5	Vdd + 0.5	V
DC input current	-20	+20	mA
Storage temperature	-40	+125	°C
Lead temperature		+300	°C

Table 20-1: DC maximum ratings

20.2 DC Operating Conditions

Parameters	Min	Max	Unit
DC Supply voltage	+2.7	+5.5	V
DC input / output voltage	0	Vdd	V
DC input current	-15	+15	mA
Operating temperature	0	+70	°C

Table 20-2: DC operating conditions

20.3 DC Characteristics

All characteristics are specified at Vdd = 3.0 to 3.6 volts and Vss = 0 volts over an operating temperature of 0°C to +70°C.

Symbol	Parameter	Min	Max	Unit	Conditions
VIH	CMOS input high voltage	0.7 x Vdd	Vdd + 0.3	V	
VIL	CMOS input low voltage	-0.3	0.2 x Vdd	V	
VT+	Schmitt trigger positive going threshold	1.52	2.26	V	
VT-	Schmitt trigger negative going threshold	0.72	1.29	V	
Vhst	Schmitt trigger hysteresis	0.64	1.13	V	VIL to VIH
VOH	CMOS output high voltage	Vdd - 0.1		V	IOH = 0.8 mA
	Standard drive output	Vdd - 1.0		V	IOH = 4 mA
	Medium drive output	Vdd - 1.0		V	IOH = 6 mA
	High drive output	Vdd - 1.0		V	IOH = 12 mA
	Very high drive output	Vdd - 1.0		V	IOH = 24 mA
VOL	CMOS output low voltage		0.1	V	IOL = -0.8 mA
	Standard drive output		0.5	V	IOL = -4 mA
	Medium drive output		0.5	V	IOL = -6 mA
	High drive output		0.5	V	IOL = -12 mA
	Very high drive output		0.5	V	IOL = -24 mA
IIN	Input leakage current	-10	+10	µA	VIN = VDD or GND
IOZ	Output Tri-state leakage current	-10	+10	µA	VOUT = VDD or GND
CIN	Input capacitance		5	pF	
COUT	Output capacitance		5	pF	
CI/O	Transceiver capacitance		5	pF	

Table 20-3: DC characteristics

Symbol	Parameter	Min	Max	Unit	Conditions
$IDD_{startup}$	Startup current consumption		100	µA	Initial 100 mSec from power up, 32 KHz oscillator not stable, POR signal at VIL, all other I/O static, VIH = VDD ± 0.1V, VIL = GND ± 0.1V
$IDD_{standby}$	Standby current consumption		50	µA	Just 32 KHz oscillator running, all other I/O static, VIH = VDD ± 0.1V, VIL = GND ± 0.1V
IDD_{idle}	Idle current consumption		5	mA	Both oscillators running, CPU static, LCD refresh active, VIH = VDD ± 0.1V, VIL = GND ± 0.1V
$IDD_{operating}$	Operating current consumption		30	mA	All system active, running typical program
$VDD_{standby}$	Standby supply voltage	2.2		V	Minimum standby voltage for state retention and RTC operation only.

Table 20-3: DC characteristics

20.4 AC Characteristics

All characteristics are specified at Vdd = 3.0 to 3.6 volts and Vss = 0 volts over an operating temperature of 0°C to +70°C.

Symbol	Parameter	Min	Max	Unit
T1	Falling CS to data bus Hi-Z	0	25	nS
T2	Address change to valid write data	0	35	nS
T3	DATA in to falling EXPCLK setup time	18		nS
T4	DATA in to falling EXPCLK hold time	0		nS
T5	EXPRDY to falling EXPCLK setup time	18		nS
T6	Falling EXPCLK to EXPRDY hold time	0	50	nS
T7	Rising nMWE to data nvalid hold time	5		nS
T8	Data valid to falling nMWE setup time	15		nS
T9	Row address to falling nCAS setup time	18		nS
T10	Falling nRAS to row address hold time	25		nS
T11	Column address to falling nCAS setup time	2		nS
T12	Falling nCAS to column address hold time	25		nS
T13	Write data valid to falling nCAS setup time	2		nS
T14	Write data valid from falling nCAS hold time	50		nS
T15	LCD CL2 low time	80	3475	nS
T16	LCD CL2 high time	80	3475	nS
T17	LCD rising CL2 to rising CL1 delay	0	25	nS
18	LCD falling CL1 to rising CL21	80	3475	nS
T19	LCD CL1 high time	80	3475	nS
T20	LCD falling CL1 to falling CL2	200	6950	nS
T21	LCD falling CL1 to frm toggle	300	10425	nS
T22	LCD falling CL1 to m toggle	-10	20	nS
T23	LCD rising CL2 to display data change	-10	20	nS

Table 20-4: AC Characteristics

Symbol	Parameter	Min	Max	Unit
Textrd	zero wait state memory read access time		70	nS
Texwr	zero wait state memory write access time		70	nS
Trc	DRAM cycle time		150	nS
Trac	Access time from RAS		70	nS
Trp	RAS precharge time		70	nS
Tcas	CAS pulse width		20	nS
Tcp	CAS precharge in page mode		12	nS
Tpc	Page mode cycle time		45	nS
Tcsa	CAS setup time		15	nS
Tras	RAS pulse width		80	nS

Table 20-5: System memory device parameter requirements

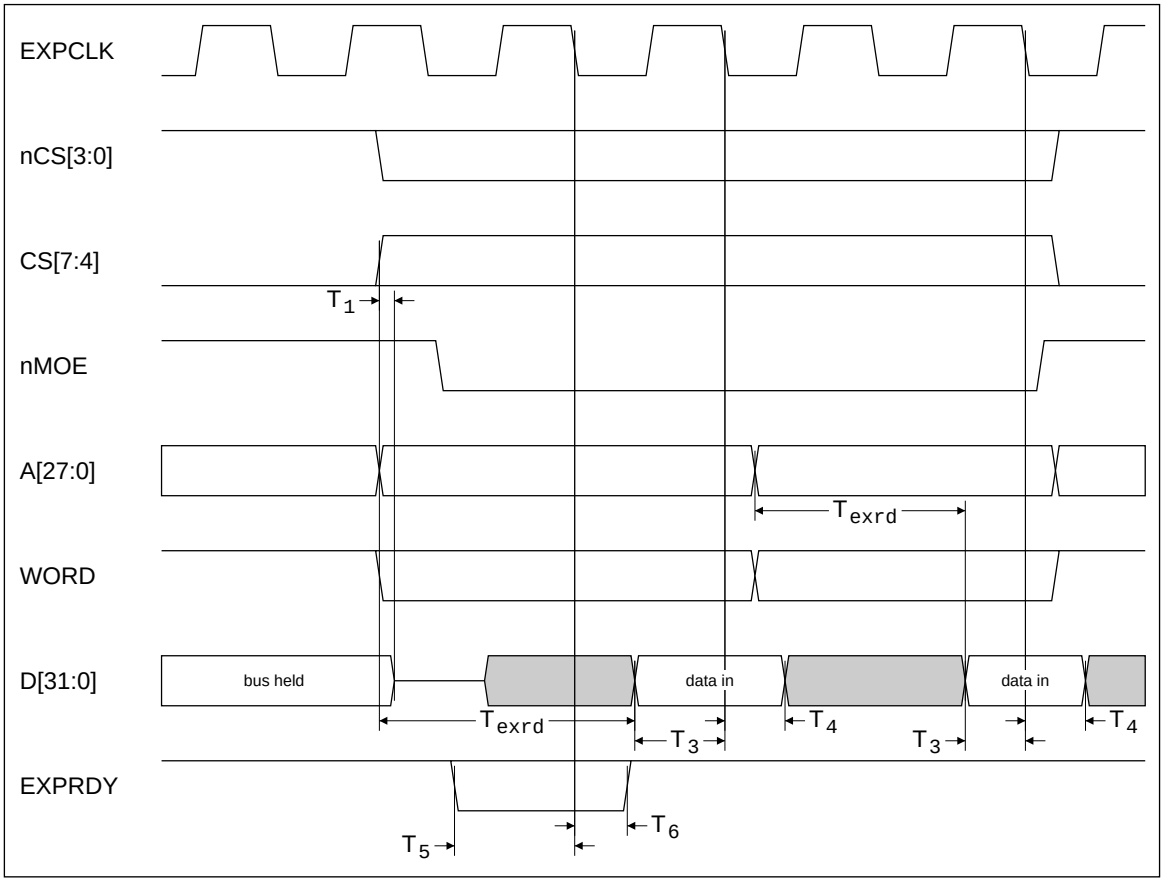


Figure 20-1: Expansion and ROM read timing

Notes

1 Texrd = 70nS for maximum wait states and a main oscillator frequency of 18.432 MHz. This time can be extended by integer multiples of the clock period (54nS), by either driving **EXPRDY** LOW or by by programming a number of wait states. **EXPRDY** is sampled on the falling edge of **EXPCLK** before the data transfer. If LOW at this point the transfer is delayed by one clock period where **EXPRDY** is sampled again. **EXPCLK** need not be referenced when driving **EXPRDY** but is shown for clarity.

2 Consecutive reads with sequential access enabled are identical except that the sequential access wait state field is used to determine the number of wait states.

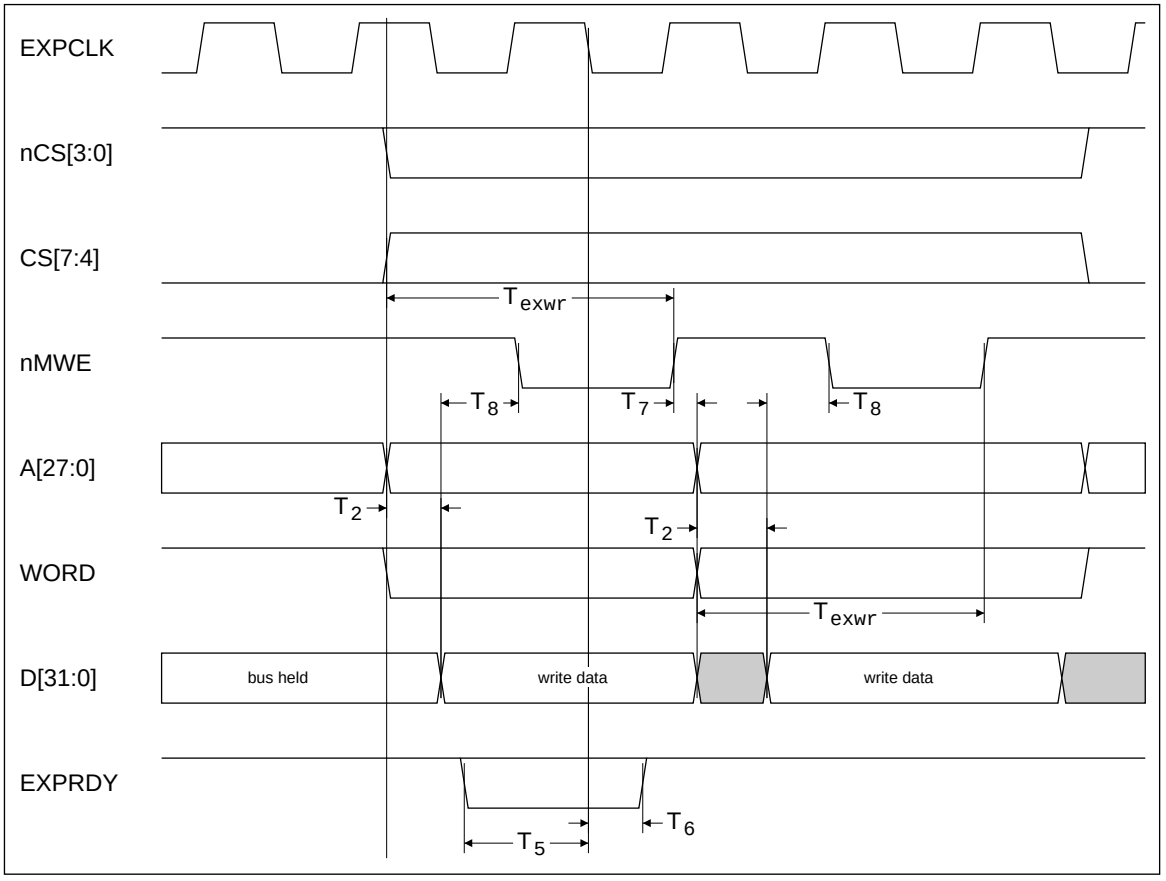


Figure 20-2: Expansion and ROM write timing

Notes

1 Texwr = 70nS max for zero wait states. This time can be extended by integer multiples of the clock period (54 nS), by either driving **EXPRDY** LOW or by programming a number of wait states. **EXPRDY** is sampled on the falling edge of **EXPCLK** before the data tansfer. If LOW at this point, the transfer is delayed by one clock period where **EXPRDY** is sampled again. **EXPCLK** need not be referenced when driving **EXPRDY** but is shown for clarity.

2 Consecutive writes with sequential access enabled are identical except that the sequential access wait state filed is used to determine the number of wait states.

3 Zero wait states for sequential writes is not supported, one state will automatically be added.

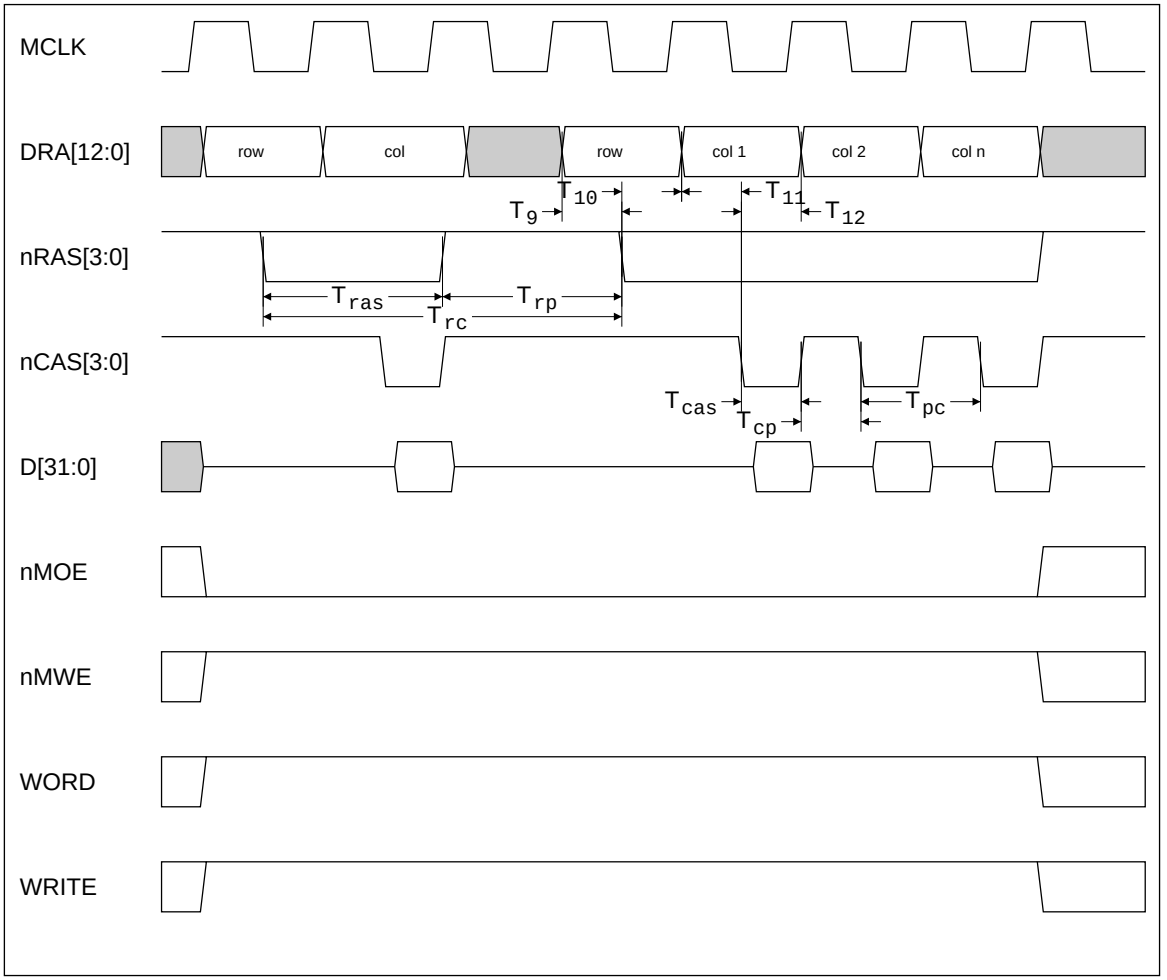


Figure 20-3: DRAM read cycles

Notes

1. Trc (Read cycle time) = 150nS max
2. Trac (Read access time from RAS) = 70nS max
3. Trp (RAS precharge time) = 70nS max
4. Tcas (CAS pulse width) = 20nS max
5. Tcp (CAS precharge in page mode) = 12nS max
6. Tpc (page mode cycle time) = 45nS min at max
7. Word reads shown, for byte reads only one of **CAS[3:0]** will be active, **CAS[0]** for byte 0 etc.

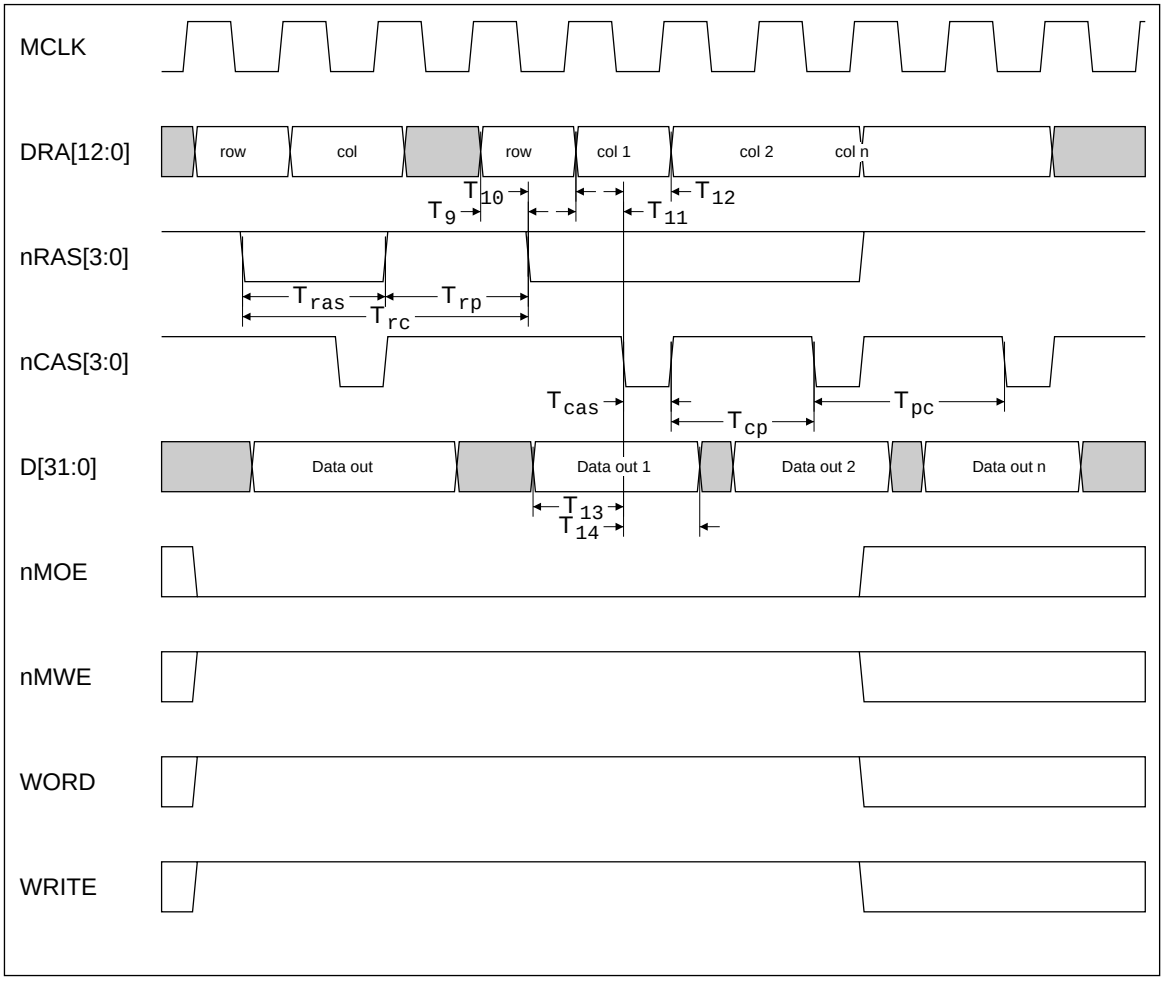


Figure 20-4: DRAM write cycles

Notes

1 Trc (Write cycle time) = 150nS max at **MCLK** = 18.432MHz

2 Trac (Write access time from RAS) = 70nS max at **MCLK** = 18.432MHz

3 Trp (RAS precharge time) = 70nS max at **MCLK** = 18.432MHz

4 Tcas (CAS pulse width) = 20nS max at **MCLK** = 18.432MHz

5 Tcp (CAS precharge in page mode) = 66nS max at **MCLK** = 18.432MHz

6 Tpc (page mode cycle time) = 45nS max at **MCLK** = 18.432MHz

7 Word writes shown, for byte writes only one off **CAS[3:0]** will be active, **CAS[0]** for byte 0 etc.

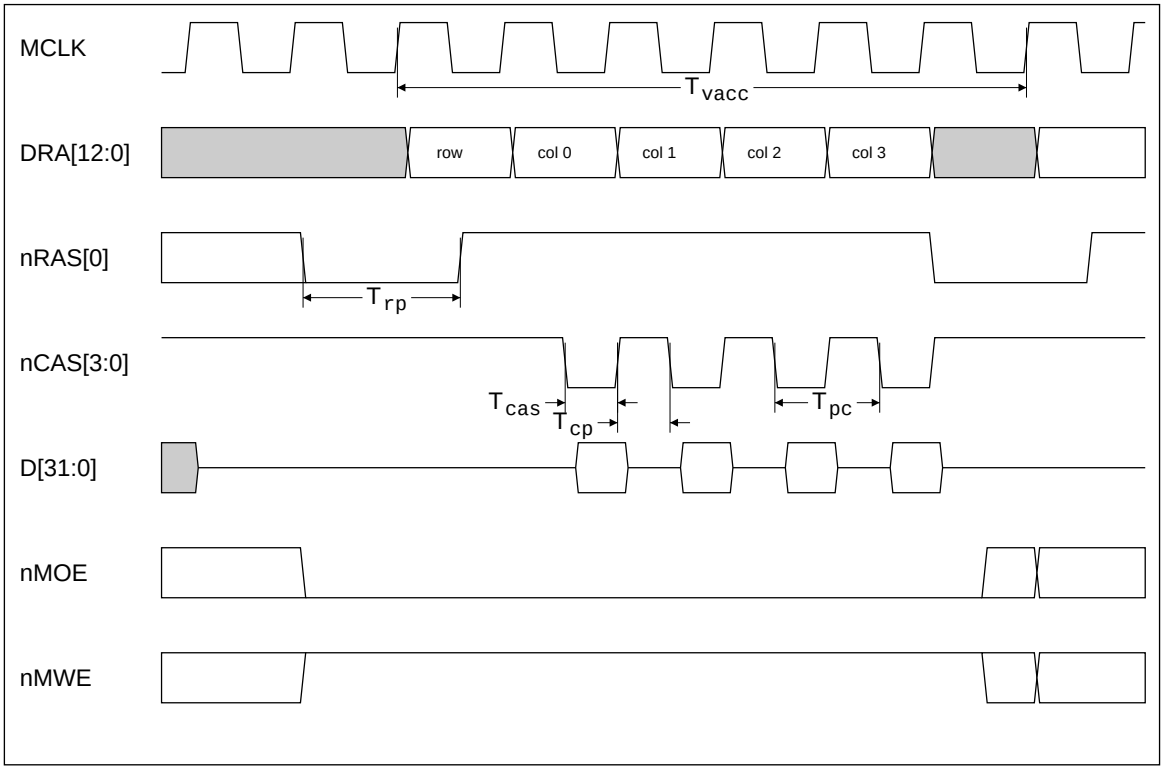


Figure 20-5: Video quad word read

Notes

1 Timings are the same as page mode word reads

2 Tvacc (video access cycle time) = 326nS at **MCLK** = 18.432 MHz

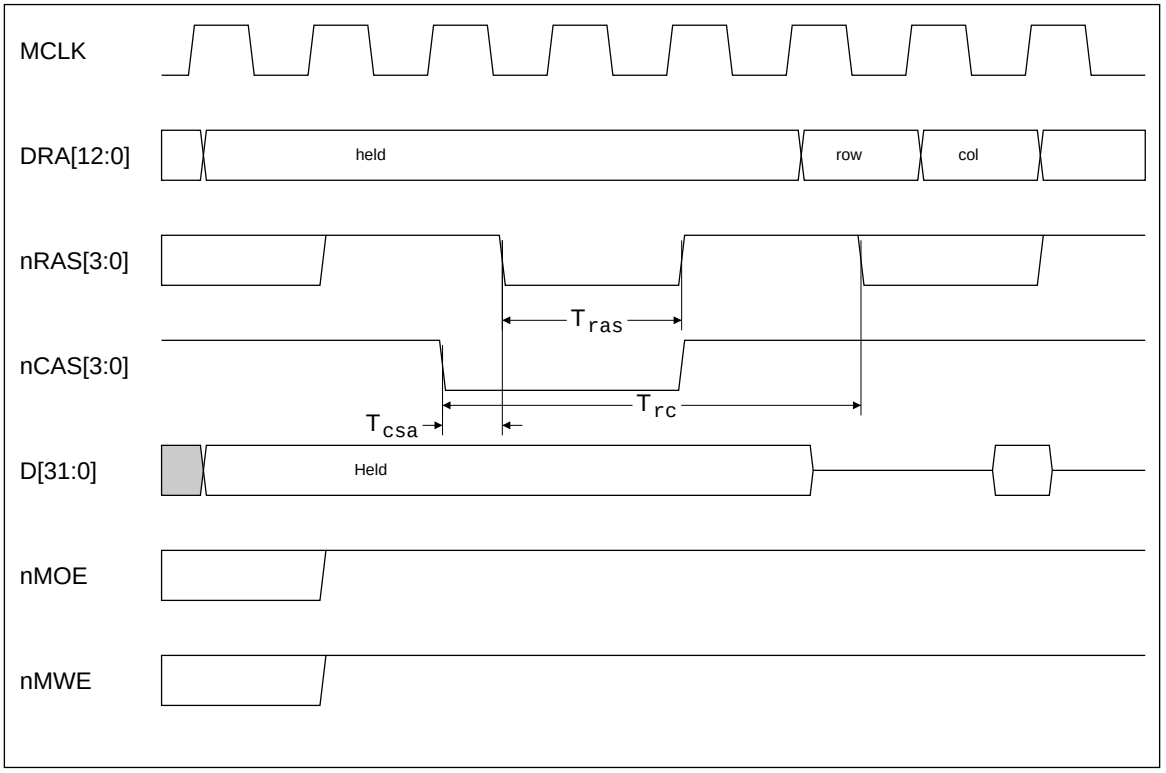


Figure 20-6: DRAM CAS before RAS refresh cycle

Notes

1 Tcsa (CAS set-up time) = 25nS min at **MCLK** = 18.432MHz

2 Tras (RAS pulse width) = 70nS min at **MCLK** = 18.432MHz

3 Trc (Cycle time) = 150nS min at **MCLK** = 18.432MHz

4 When DRAM's are placed in self refresh (entering standby), the same timings apply except that Tras is extended indefinitely.

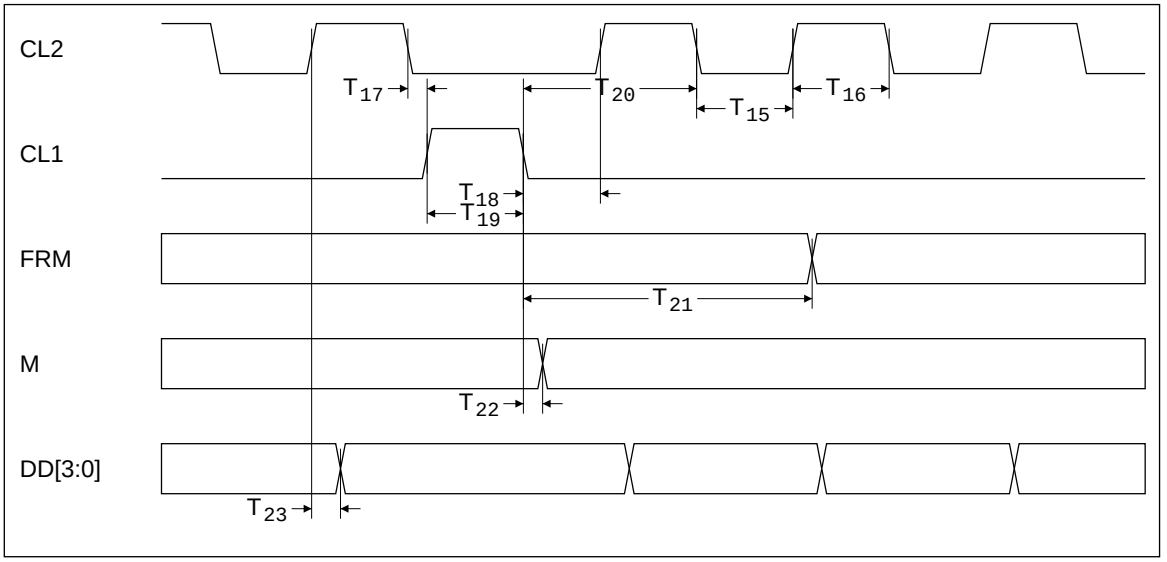


Figure 20-7: LCD controller timing

Notes

1 *Figure 20-7: LCD controller timing* shows the end of a line.

2 If **FRM** is HIGH during the **CL1** pulse, this marks the first line in the display.

3 **CL2** LOW time is doubled during the **CL1** high pulse.

21 Physical Details

This chapter describes the physical details of the ARM7100.

21.1 Pin diagrams for the ARM7100 21-2

Preliminary

21.1 Pin diagrams for the ARM7100

The following two diagrams illustrate the top and side views of the ARM7100. All dimensions are given in millimetres.

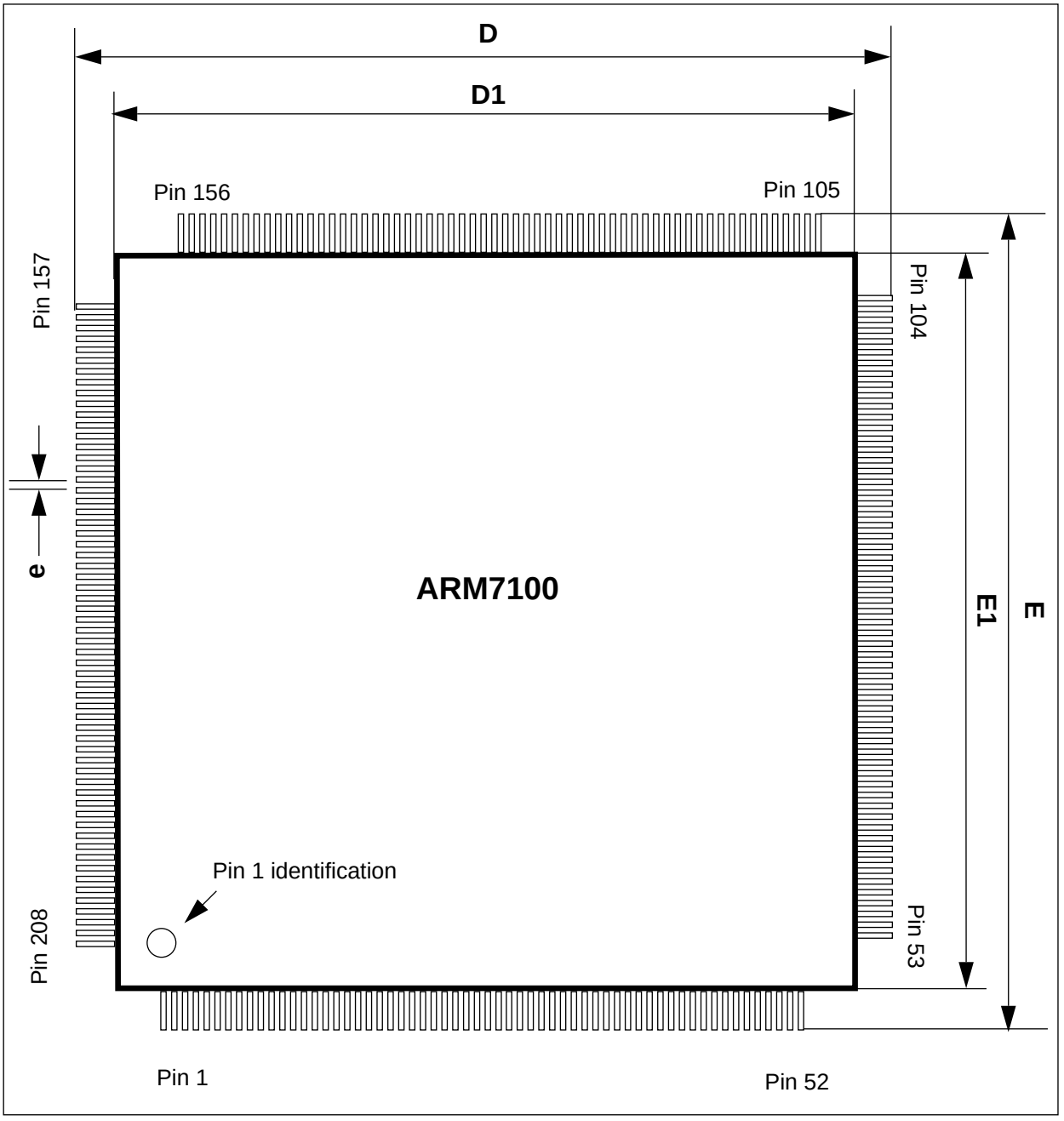


Figure 21-1: Pin diagram for the ARM7100

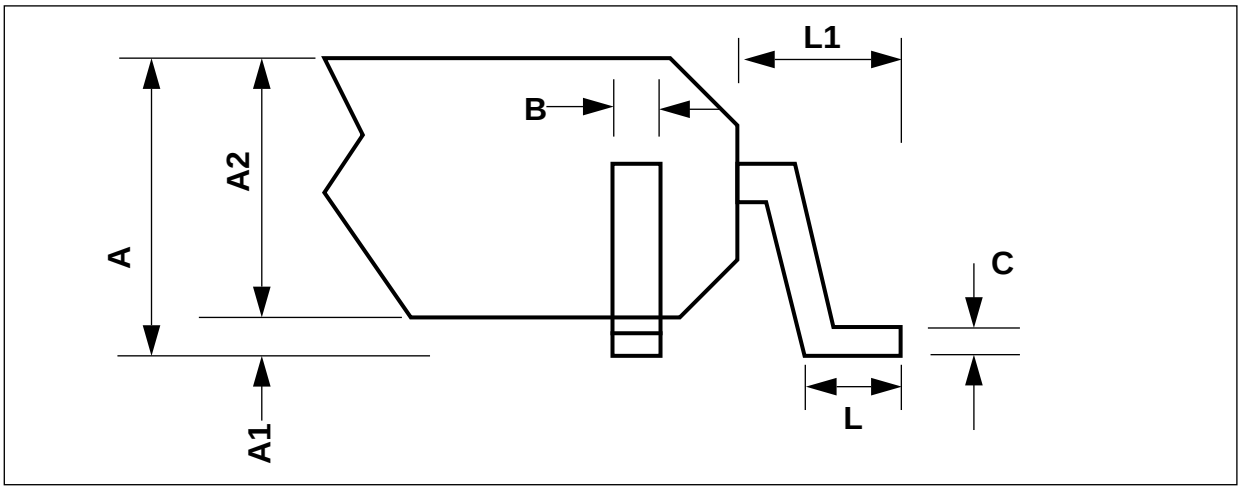


Figure 21-2: Side view of ARM7100 chip

Symbol	Description	Min	Max
A	Package thckness plus stand off	1.4	1.6
A1	Stand off	0.05	0.15
A2	Package thickness	1.35	1.45
D	Lead tip to Lead tip (X)	29.6	30.4
D1	Package body dimension (X)	27.8	28.2
E	Lead tip to Lead tip (Y)	29.6	30.4
E1	Package body dimension (Y)	27.8	28.2
L	Foot Length	0.45	0.75
L1	Total lead length	1.00 BSC	
e	Lead pitch	1.00 BSC	
B	Lead width with plating	0.17	0.27

Table 21-1: Chip dimensions

Notes

1 Controlling dimension is mm
2 Dimensions D1 and E1 do not include mold protrusion which is 0.25mm (0.010 inch)
3 Lead frame material is copper
4 Lead finish is solder plate
5 Pin 1 identification may be either ink dot or dimple
6 Package top dimensions can be smaller than bottom dimensions by 0.20mm (0.008 inch)

22 Pinout

This chapter describes the ARM7100 pinout.

22.1 Pin details 22-2

Preliminary

22.1 Pin details

The following table gives the signal name for each of the 208 pins of the ARM7100.

Pin number	Signal name	Pin number	Signal name
1	CS[5]	27	PB[2]
2	CS[6]	28	PB[1]
3	CS[7]	29	PB[0]
4	VDD	30	PE[3]
5	VSS	31	PE[2]
6	EXPCLK	32	VDD
7	WORD	33	VSS
8	WRITE	34	PA[7]
9	RUN	35	PA[6]
10	EXPRDY	36	PA[5]
11	PC[7]	37	PA[4]
12	PC[6]	38	PA[3]
13	PC[5]	39	PA[2]
14	PC[4]	40	PA[1]
15	PC[3]	41	PA[0]
16	PC[2]	42	LEDDRV
17	PC[1]	43	TXD
18	PC[0]	44	PHDIN
19	VDD	45	CTS
20	VSS	46	RXD
21	VSS	47	DCD
22	PB[7]	48	DSR
23	PB[6]	49	VSS
24	PB[5]	50	RTCOUT
25	PB[4]	51	RTCIN
26	PB[3]	52	VDD

Table 22-1: Pin numbers and signal names

Pin number	Signal name	Pin number	Signal name
53	nTEST1	82	DRIVE0
54	nTEST0	83	ADCCLK
55	EINT3	84	ADCOUT
56	nEINT2	85	SMPLCK
57	nEINT1	86	FB1
58	nEXTFIQ	87	FB0
59	PE[1]	88	COL7
60	PE[0]	89	COL6
61	PD[7]	90	COL5
62	PD[6]	91	COL4
63	PD[5]	92	COL3
64	PD[4]	93	COL2
65	VDD	94	VDD
66	VSS	95	VSS
67	PD[3]	96	COL1
68	PD[2]	97	COL0
69	PD[1]	98	BUZ
70	PD[0]	99	D[31]
71	PCMIN	100	D[30]
72	PCMCK	101	D[29]
73	PCMOUT	102	D[28]
74	PCMSYNC	103	A[27]/DRA[0]
75	ADCIN	104	D[27]
76	nADCCS	105	A[26]/DRA[1]
77	VSS	106	D[26]
78	VDD	107	A[25]/DRA[2]
79	VSS	108	D[25]
80	VDD	109	A[24]/DRA[3]
81	DRIVE1	110	VDD

Table 22-1: Pin numbers and signal names (Continued)

Pin number	Signal name	Pin number	Signal name
111	VSS	140	A[11]
112	D[24]	141	VDD
113	A[23]/DRA[4]	142	VSS
114	D[23]	143	D[11]
115	A[22]/DRA[5]	144	A[10]
116	D[22]	145	D[10]
117	A[21]/DRA[6]	146	A[9]
118	D[21]	147	D[9]
119	A[20]/DRA[7]	148	A[8]
120	D[20]	149	D[8]
121	A[19]/DRA[8]	150	A[7]
122	D[19]	151	D[7]
123	A[18]/DRA[9]	152	nBATCHG
124	D[18]	153	nEXTPWR
125	VDD	154	BATOK
126	VSS	155	nPOR
127	VSS	156	MEDCHG
128	A[17]/DRA[10]	157	VDD
129	D[17]	158	MOSCIN
130	A[16]/DRA[11]	159	MOSCOUT
131	D[16]	160	VSS
132	A[15]/DRA[12]	161	nURESET
133	D[15]	162	WAKEUP
134	A[14]	163	nPWRFL
135	D[14]	164	A[6]
136	A[13]	165	D[6]
137	D[13]	166	A[5]
138	A[12]	167	D[5]
139	D[12]	168	VDD

Table 22-1: Pin numbers and signal names (Continued)

Pin number	Signal name	Pin number	Signal name
169	VSS	189	DD[2]
170	A[4]	190	DD[1]
171	D[4]	191	DD[0]
172	A[3]	192	nRAS[3]
173	D[3]	193	nRAS[2]
174	A[2]	194	nRAS[1]
175	D[2]	195	nRAS[0]
176	A[1]	196	nCAS[3]
177	D[1]	197	nCAS[2]
178	A[0]	198	VDD
179	D[0]	199	VSS
180	VSS	200	nCAS[1]
181	VDD	201	nCAS[0]
182	VSS	202	nMWE
183	VDD	203	nMOE
184	CL2	204	nCS[0]
185	CL1	205	nCS[1]
186	FRM	206	nCS[2]
187	M	207	nCS[3]
188	DD[3]	208	CS[4]

Table 22-1: Pin numbers and signal names (Continued)

Index

A

Abort operating mode 4-4
Access faults
  checking 7-15
Address translation 7-4

B

Backward compatibility
  configuration bits 4-3
block diagram
  ARM704 3-3
Branch instructions 5-4

C

CDP instruction 5-39
compilers 3-2
Condition codes 5-3
Configuration bits
  for backward compatibility 4-3
Configuration settings
  register 4-2
Control register
  big endian format 4-2
  little endian format 4-2
Coprocessor data operations 5-39
Coprocessor instructions 5-38
CPSR flags 5-7

D

Data processing instructions 5-6
DC parameters 20-1
Domain access control 7-14
Domain access control register 7-3

E

Examples
  instruction set 5-48
Exceptions 4-7
  abort 4-8
  FIQ 4-7
  IRQ 4-8
  priorities 4-10

F

Fault address register 7-3, 7-13
Fault checking 7-15
Fault status register 7-3, 7-13
FIQ exception 4-7
FIQ operating mode 4-4

Preliminary

I

IDC
cacheable bit 6-2
disable 6-3
enable 6-3
interaction with MMU and write buffer 7-18
operation 6-2
read-lock-write 6-3
reset 6-3
validity 6-2
instruction set
ARM704 3-2
Instruction set examples
loading a halfword 5-51
loading a word from an unknown alignment 5-50
multiply by constant 5-49
pseudo random binary sequence gneerator 5-49
using conditional instructions 5-48
Instruction set summary 5-2
Instruction speed summary 5-52
Internal coprocessor instructions 4-11
IRQ exception 4-8
IRQ operating mode 4-4

L

LDC instruction 5-41
LDM instruction 5-27
LDR instruction 5-21

M

MCR instruction 5-44
MLA instruction 5-19
MMU
interaction with IDC and write buffer 7-18
MRC instruction 5-44
MRS instruction 5-15
MSR instruction 5-15
MUL instruction 5-19

O

Operating modes
selecting 4-4

P

Parameters
DC 20-1
physical details 21-2
pin details 22-2
pin diagrams 21-2

R

Register confidurations 4-2
Registers 4-4, 4-11
MMU 7-3

S

Shifts 5-9
Signal descriptions 2-2
Software interrupt instruction 4-9, 5-36
STC instruction 5-41
STM instruction 5-27
STR instruction 5-21
Supervisor operating mode 4-4
SWP instruction 5-34

T

Translating references 7-5
Translation table base register 7-3

U

Undefined instruction 5-47
Undefined instruction trap 4-9
Undefined operating mode 4-4
User operating mode 4-4

W

Write buffer
interaction with MMU and IDC 7-18