
ARM software

reference manual

ARM Evaluation System

Acorn OEM Products



ARM software

Part No 0448,007
Issue No 1.0
24 July 1986

© Copyright Acorn Computers Limited 1986

Neither the whole nor any part of the information contained in, or the product described in, this manual may be adapted or reproduced in any material form except with the prior written permission of the copyright holder. The only exceptions are as provided for by the Copyright (photocopying) Act, or for the purpose of review, or in order for the software herein to be entered into a computer for the sole use of the owner of this book.

Within this publication the term 'BBC' is used as an abbreviation for 'British Broadcasting Corporation'.

- The manual is provided on an 'as is' basis except for warranties described in the software licence agreement if provided.
- The software and this manual are protected by Trade secret and Copyright laws.

The product described in this manual is subject to continuous developments and improvements. All particulars of the product and its use (including the information in this manual) are given by Acorn Computers in good faith.

There are no warranties implied or expressed including but not limited to implied warranties or merchantability or fitness for purpose and all such warranties are expressly and specifically disclaimed.

In case of difficulty please contact your supplier. Every step is taken to ensure that the quality of software and documentation is as high as possible. However, it should be noted that software cannot be written to be completely free of errors. To help Acorn rectify future versions, suspected deficiencies in software and documentation, unless notified otherwise, should be notified in writing to the following address:

Customer Services Department,
Acorn Computers Limited,
645 Newmarket Road,
Cambridge
CB5 8PD

All maintenance and service on the product must be carried out by Acorn Computers. Acorn Computers can accept no liability whatsoever for any loss, indirect or consequential damages, even if Acorn has been advised of the possibility of such damage or even if caused by service or maintenance by unauthorised personnel. This manual is intended only to assist the reader in the use of the product, and therefore Acorn Computers shall not be liable for any loss or damage whatsoever arising from the use of any information or particulars in, or any error or omission in, this manual, or any incorrect use of the product.

Econet® and The Tube® are registered trademarks of Acorn Computers Limited.

ISBN 1 85250 001

Published by:

Acorn Computers Limited, Fulbourn Road, Cherry Hinton, Cambridge CB1 4JN, UK

Contents

1. Architectural description	1
1.1 Introduction	1
1.2 Programmer's model	1
1.2.1 Memory organisation	2
1.3 Registers	2
1.4 Modes	4
1.4.1 Mode 0	5
1.4.2 Mode 1	5
1.4.3 Mode 2	5
1.4.4 Mode 3	6
2. Instruction set	7
2.1 Branch and branch with link	7
2.1.1 Assembler syntax	9
2.2 Data processing	9
2.2.1 Data processing on registers	13
2.2.2 Data processing with register and immediate operand	16
2.2.3 Changing modes.	17
2.3 Single data transfer group	18
2.3.1 [Rn, off] is a pre-indexing addressing mode	20
2.3.2 [Rn,Rm] is a pre-indexed addressing mode	20
2.3.3 [Rn],off is a post-indexed addressing mode	20
2.3.4 [Rn],Rm is a post-indexed addressing mode	21
2.4 Block data transfer	22
2.4.1 Assembler syntax	24
2.5 Supervisor calls	25
3. Interrupts	27
3.1 Reset	27
3.2 Address exception trap	28
3.3 Abort	28
3.4 FIQ	29
3.5 IRQ	30
3.6 Undefined instruction trap	30
3.7 Software interrupt	30
4. Appendix A	32
4.1 Instruction speeds	32
5. Appendix B	34

5.1 Virtual memory concept	34
6. Appendix C	35
6.1 Instruction set summary	35
7. Appendix E	37
7.1 Notional stacking	37
7.1.1 Stacking	37



1. Architectural description

1.1 Introduction

The ARM (Acorn RISC Machine) is an 84-pin, 32-bit single-chip CMOS microprocessor designed for optimal support of high-level language and fast real-time operation. It features:

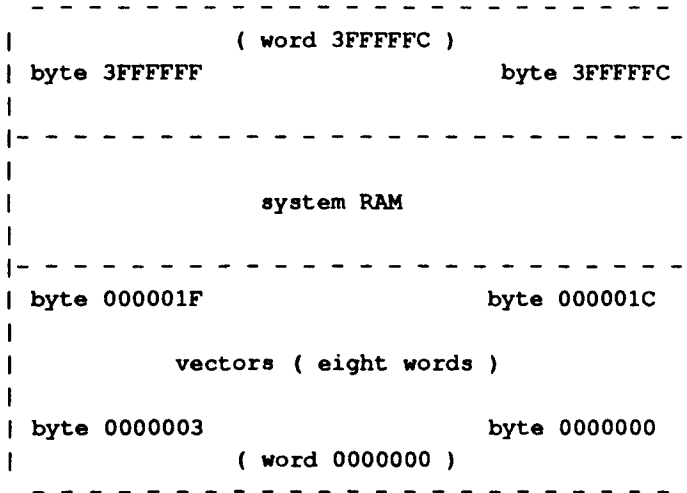
- reduced instruction set architecture
- 32-bit data bus
- all instructions are the same size (32 bits)
- 26-bit address bus
- performance in excess of 3 million instructions per second (MIPS)
- 18 Mbyte/second memory bandwidth using 150 nS DRAMs
- one instruction executed on every clock cycle
- supports demand-paged virtual memory
- user and supervisor modes
- fast interrupt capability.

This manual is intended to serve as a programmer's reference for both systems and applications programmers; the hardware system design aspects such as bus structure, control and timing waveforms are presented in the *ARM hardware reference manual*. The assembler for the ARM is described in the *ARM assembler reference manual*.

1.2 Programmer's model

The ARM utilises an instruction pipeline to hold consecutive instructions. While one instruction is manipulating data, a second is being decoded, and a third is being fetched from memory. The execution speed is always one instruction per clock cycle. During the execution of an instruction the program counter is eight bytes on, drawing a fresh instruction into the pipe.

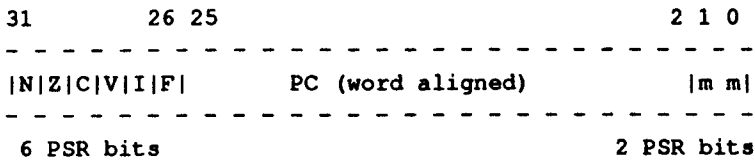
1.2.1 Memory organisation



1.3 Registers

The 24-bit PC shares a 32-bit register with various condition codes and status bits forming a compact processor status word. This register holds the ARM's current status and can be saved quickly and efficiently in one processor cycle. The condition codes and status bits are known collectively as the Processor Status Register or PSR. The programmer sees a bank of sixteen 32-bit registers, R0 to R15 when the ARM is in user mode. The only two special purpose registers are R14 and R15. R15 contains the 24-bit PC, and the PSR information making the full processor status word. R14 is the subroutine Link register, which always receives a copy of R15 (the PSW) on a branch with link instruction. Special bits in the processor's instructions allow the PC and PSR to be treated together or separately.

The format of register 15 is as follows:



The PSR bits consist of:

- (1) flags N, Z, C, and V, the Condition Codes Register (CCR).
These flags are set both by the ARM's Arithmetic Logic Unit or ALU and its barrel shifter. They can be altered by the programmer when the processor is in user mode.
- (2) flags I and F, which control the ARM's Interrupt mechanism.
The programmer cannot alter these bits while in user mode.
- (3) bits m m which determine the processor's mode of operation.
They are only alterable by the programmer when the processor is not in user mode.

The ARM executes instructions in one of four modes, one of which is user mode. The user mode is intended to provide the environment for the majority of application programs, while the other modes are intended for use by the operating system software.

The full 32 bits of register 15 consist of:

- N (bit 31) : has dual use:
negative flag
signed less than flag.
- Z (bit 30) : Zero flag : set by ALU arithmetic and compares.
- C (bit 29) : multiple use:
Carry flag: set by ALU or by the barrel shifter
Absence of borrow: set by ALU
Rotate extend flag: set by barrel shifter
- V (bit 28) : is the oVerflow flag
- I (bit 27) : is Interrupt ReQuest (IRQ) disable
- F (bit 26) : is Fast Interrupt reQuest (FIQ) disable

- bits B25-B2 are combined with a processor-produced B1 zero and B0 zero to form a full word address which is then output on the address bus
- m m (bits $m0$ and $m1$) determines which of four modes is selected.

The flags are read as logic high for the stated condition.

Access to bytes

Load and store operations can operate on either bytes or words. These instructions can put a 26-bit value, with bits B0 and B1 set as required, on to the address bus. B0 and B1 simply represent the offset from the word boundary and cause the data lines to access a particular byte.

The processor can access two types of data: bytes (8 bits) and words (32 bits). The data must lie on Byte and word boundaries respectively. The memory is organised in byte-wide fashion and the 24+2-bit program-counter PC is capable of counting to $\&3FFFFFFC$, giving the ARM a memory capacity of 64 Mbytes, or 16 Mwords. Since each ARM instruction occupies a word of memory, the PC moves in steps of four, and a valid PC value can be held in 24 bits (address lines B2-B25), with the two least-significant bits of zeroes (B0,B1) being added by hardware before the PC is placed on the 26 address lines.

1.4 Modes

The ARM has four modes of operation - user, supervisor, interrupt and fast interrupt. The mode in which the processor runs is determined by the state of bits 0 and 1 in the PSR. The processor has 25 physical registers, but the state of the mode bits determine which 16 registers, R0-R15, will be seen by the programmer. The four modes available are shown in the diagram on the next page.

Value of mode bits			
0	1	2	3
User/Normal	FIQ	IRQ	SVC/Abort/Undefined
R0	R0	R0	R0
R10	R10_FIQ	R10	R10
R11	R11_FIQ	R11	R11
R12	R12_FIQ	R12	R12
R13	R13_FIQ	R13_IRQ	R13_SVC
R14	R14_FIQ	R14_IRQ	R14_SVC
R15	R15	R15	R15

In each mode the conceptual registers R0-R9 and R15 correspond to the physical registers R0-R9 and R15.

1.4.1 Mode 0

User mode is the normal program execution state; registers R0-15 exist directly and in this mode only the N, Z, C and V bits of the PSR may be changed.

1.4.2 Mode 1

The FIQ processing state has five private registers mapped to R10-14 (R10_FIQ-R14_FIQ) and a fast interrupt will not destroy anything in R10-R14. Most FIQ programs, particularly those used for data transfer, will not need to use R0-R9, but if they do, then R0-R9 can be saved in memory using a single instruction.

1.4.3 Mode 2

The IRQ processing state has two private registers mapped to R13, R14 (R13_IRQ, R14_IRQ). If other registers are needed, their contents should be saved in memory using the single instruction available for this purpose.

1.4.4 Mode 3

Supervisor mode (entered on SVC calls and other traps) also has two private registers mapped to R13, R14 (R13_SVC, R14_SVC). If other registers are needed, they too must be saved in memory.

Non-user modes are privileged and allow trusted software to take control in a suitably protected memory style.

The code used to effect mode changes is shown in section 2.2.1.

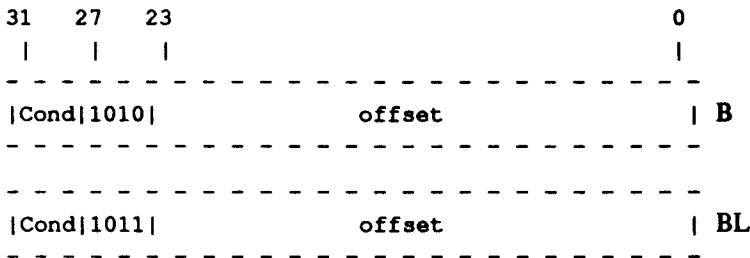
2. Instruction set

There are fourteen instructions determined by the bit pattern in B24-B27, divided into five classes. The full instruction set is given in chapter 3. The five classes are described in the following sections.

2.1 Branch and branch with link

Bits B24-B27 are set to 101x, giving two instruction types, the branch (B) and the branch with link (BL).

There are 16 branch instructions and 16 branch-with-link instructions, determined by the pattern in bits B28-B31.



The Branch instruction has a 24-bit word offset, to which two zero bits are appended. The 26-bit value formed allows forward jumps of up to +&2000004 and backward jumps of up to -&1FFFFFF8 to be made. This is sufficient to address the entire memory map, as the calculation wraps round between the top and bottom of memory.

The Branch-with-link instruction writes the address of the next instruction (the instruction following the BL) into R14. Due to pipelining the PC is already eight bytes in advance of the BL instruction, so the instruction following BL is at PC-4. The PSR is also copied into R14.

The encoding of bits B28-B31 is as follows:

Code	Mnemonic	Condition	Condition of flag(s)
0000	EQ	EQual	Z set
0001	NE	Not Equal	Z clear
0010	CS	Carry Set / unsigned higher or same	C set
0011	CC	Carry Clear / unsigned lower than	C clear
0100	MI	negative (MInus)	N set
0101	PL	positive (PLus)	N clear
0110	VS	oVerflow Set	V set
0111	VC	oVerflow Clear	V clear
1000	HI	Higher unsigned	C set and Z clear
1001	LS	Lower or Same unsigned	C clear or Z set
1010	GE	Greater or Equal	(N set and V set) or (N clear and V clear)
1011	LT	Less Than	(N set and V clear) or (N clear and V set)
1100	GT	Greater Than	((N set and V set) or (N clear and V clear)) and Z clear
1101	LE	Less or Equal	(N set and V clear) or (N clear and V set) or Z set
1110	AL	ALways	any
1111	NV	NeVer	none

Note: The assembler implements HS (Higher or Same) and LO (Lower than) as synonymous with CS and CC respectively, giving a total of 18 mnemonics.

- If a link is involved, the mnemonic is expanded to the form BLxx.
- BNV and BLNV are no operations: the branch is never taken.
- BAL and BLAL may be shortened to B and BL.

The branch offset must take account of the prefetch operation, which causes the PC to be two words ahead of the current instruction. An ARM assembler will handle this situation. For example, the calculated jump offset in the following piece of code is 000000 even though the jump is to a label, which is two PC locations ahead.

Code generated	Label	Mnemonic	Destination
EA000000	L1	BEQ	L2
xxxxxxxxxx		xxx	
xxxxxxxxxx	L2	xxx	

In spite of the prefetching of instructions the value written into the link register is the address of the instruction following the Branch-and-link instruction. Therefore after branching to a subroutine, the program flow can return to the memory address immediately following the branch instruction by writing back the R14 value into R15. Subroutines can be called by a BL instruction. The subroutine should end with `MOV PC,R14` if the link register has not been saved on a stack, or `LDMxx Rn,{PC}` if the link register has been saved on a stack addressed by Rn.

These methods of returning do not restore the original PSR. If the PSR does need to be restored, `MOV PC,R14` can be replaced by `MOVS PC,R14`, or `LDMxx Rn,{PC}` by `LDMxx Rn,{PC}^`. However, care should be taken when using these methods in modes other than user mode, as they will also restore the mode and the interrupt bits. The last in particular may interfere unintentionally with the interrupt system.

The significance of the different types of bracket and the ^ symbol are explained in the *ARM ASSEMBLER reference manual*.

2.1.1 Assembler syntax

`B {L} {cond} expression`

- `{L}` optional link
- `{cond}` is a two-character mnemonic as in the Table above (EQ, NE, VS etc.). If absent then ALWAYS will be used.
- `expression` is the destination. The assembler calculates the offset.

2.2 Data processing

Bits B25-B27 are set to 00x, giving two broad types of instruction, data processing on registers and data processing with register and immediate operand.

Bits B21-B24 extend this to 16 data-processing types. Any one of the 16 conditional tests may be applied to the instructions and the data-processing instructions will only be executed if the condition specified is true. The conditions allowed are the same as those used in the branch instructions.

The bit pattern in B21-B24 instructs the processor to perform various arithmetic operations:

Code	Mnemonic	Operation
0101	ADC	ADd with Carry
0100	ADD	ADD
0000	AND	AND
1110	BIC	Bit Clear
1011	CMN	CoMpare Negated
1010	CMP	CoMPare
0001	EOR	Exclusive OR
1101	MOV	MOVE
1111	MVN	MoVe Not
1100	ORR	logical OR
0011	RSB	Reverse SuBtract
0111	RSC	Reverse Subtract with Carry
0110	SBC	SuBtract with Carry
0010	SUB	SUBtract
1001	TEQ	Test EQivalence
1000	TST	TeST and mask

ADC causes an addition to be performed on *operand1* and *operand2* and the carry flag. The result is stored in the destination register. This instruction can be used to implement multi-word additions.

ADD causes an addition to be performed on *operand1* and *operand2*. The result is stored in the destination register, for example,

```
ADD R0,R1,R2 ;R0=R1+R2
```

```
ADDS R0,R1,#1 ;R0=R1+1 and set N,Z,C,V
```

AND performs a bitwise AND on *operand1* and *operand2*. The result is stored in the destination register.

BIC performs a bitwise inversion on *operand2*, then a bitwise AND is performed on *operand1* and the result of the inversion. The result is stored in the destination register.

CMN add *operand2* to *operand1*. This compare allows a negative data field to be created for a compare. Flags N, Z, C and V are altered.

CMP subtract *operand2* from *operand1*. Flags N, Z, C and V are altered.

EOR performs a bitwise Exclusive OR on *operand1* and *operand2*. The result is stored in the destination register.

MOV causes the operand to be placed unchanged in the destination register, for example,

```
MOV R0,R1,LSL#2
```

MVN causes the operand to be evaluated and its bitwise inverse to be placed in the destination register. The contents of register 1 are shifted left by 2 bits and transferred to register 0, for example,

```
MVN R2,R3
```

Register 2 is set to the bitwise inverse of the contents of register 3.

ORR performs a bitwise OR on *operand1* and *operand2*. The result is stored in the destination register.

RSB subtracts *operand1* from *operand2*. The result is stored in the destination register.

RSC subtracts *operand1* from *operand2* if the carry flag is set. If the carry flag is clear, $operand2 - operand1 - 1$ is calculated. The result is stored in the destination register.

SBC subtracts *operand2* from *operand1* if the carry flag is set. If the carry flag is clear, $operand1 - operand2 - 1$ is calculated. The result is stored in the destination register. This instruction can be used to implement multi-word subtractions.

SUB subtracts *operand2* from *operand1*. The result is stored in the destination register, for example:

```
SUBS R4,R2,R0 ;Do least significant word
           ;of subtraction
SBC R5,R3,R1 ;Do most significant word,
           ;taking account of the borrow
;This does the 64 bit subtraction
;(R5,R4) = (R3,R2) - (R1,R0)
```

The result is stored in the destination register.

TEQ performs a bitwise exclusive OR between *operand1* and *operand2*.

TST performs a bitwise AND operation between *operand1* and *operand2*.

In the case of TEQ and TST the N and V flags are altered according to the result, V is unchanged and C is set to the last bit shifted out by the barrel shifter, or is unchanged if no shifting took place.

In the case of `CMP`, `CMN`, `TEQ` and `TST`, the ARM assembler sets bit 20 automatically.

Mnemonic	Meaning	Operation	Flags Affected
<code>ADC</code>	Add with Carry	$Rd := Rn + \text{operand} + C$	N,Z,C,V
<code>ADD</code>	Add	$Rd := Rn + \text{operand}$	N,Z,C,V
<code>AND</code>	And	$Rd := Rn \text{ AND } \text{operand}$	N,Z,C
<code>BIC</code>	Bit Clear	$Rd := Rn \text{ AND } (\text{NOT}(\text{operand}))$	N,Z,C
<code>CMN</code>	Compare Negated	$Rn + \text{operand}$	N,Z,C,V
<code>CMP</code>	Compare	$Rn - \text{operand}$	N,Z,C,V
<code>EOR</code>	Exclusive Or	$Rd := Rn \text{ EOR } \text{operand}$	N,Z,C
<code>MOV</code>	Move	$Rd := \text{operand}$	N,Z,C
<code>MVN</code>	Move Not	$Rd := \text{NOT } \text{operand}$	N,Z,C
<code>ORR</code>	Logical Or	$Rd := Rn \text{ OR } \text{operand}$	N,Z,C
<code>RSB</code>	Reverse Subtract	$Rd := \text{operand} - Rn$	N,Z,C,V
<code>RSC</code>	Reverse Subtract with Carry	$Rd := \text{operand} - Rn - 1 + C$	N,Z,C,V
<code>SBC</code>	Subtract with Carry	$Rd := Rn - \text{operand} - 1 + C$	N,Z,C,V
<code>SUB</code>	Subtract	$Rd := Rn - \text{operand}$	N,Z,C,V
<code>TEQ</code>	Test Equivalence	$Rn \text{ EOR } \text{operand}$	N,Z,C
<code>TST</code>	Test and Mask	$Rn \text{ AND } \text{operand}$	N,Z,C

- The borrow operation on the `RSC` and `SBC` instructions is performed by *adding* the carry due to the hardware configuration of the CPU
- `Rd` is the destination register nextp `Rn` is a source register
- `S2` is a register, possibly shifted by a constant or by a register, or an 8-bit data value shifted by a constant
- `C` is the carry bit in the PSR
- the logical operations set flags `N` and `Z` from the ALU, `C` from the shifter; `V` is unaffected by the instruction
- the flags are not copied to the CCR in `R15` unless bit 20 of the instruction is set
- the arithmetic operations set all the flags from the ALU.

The ALU produces the `C`, `V`, `N` and `Z` signals which then become (if allowed by the instruction or the programmer) the CCR flags of the PSR. The barrel shifter accepts the PSR `C` flag on certain types of shift, and produces its own `C` signal. On logical operations, the barrel shifter's `C` signal rather than the ALU's signal will load the PSR.