ARM software

reference manual

ARM Evaluation System

Acorn OEM Products

ARM software

Part No 0448,007
Issue No 1.0
24 July 1986

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

ISBN 1 85250 001

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

Contents

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:

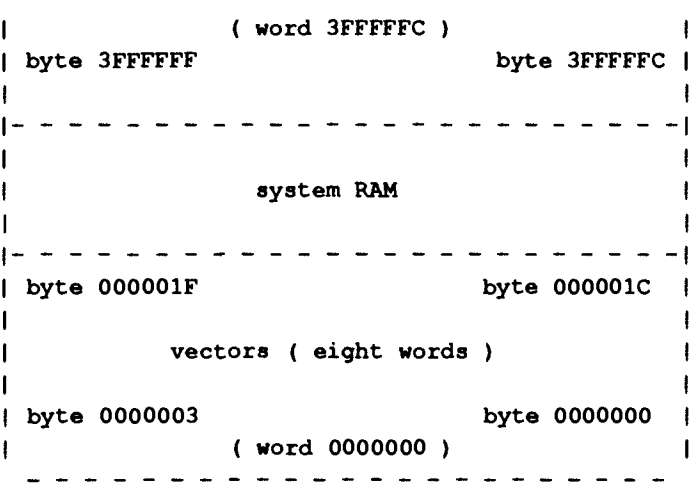
- reduced instruction set architecture
- 32-bit data bus
- all instructions are the same size (32 bits)
- 26-bit address bus
- performance in excess of 3 million instructions per second (MIPS)
- 18 Mbyte/second memory bandwidth using 150 nS DRAMs
- one instruction executed on every clock cycle
- supports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

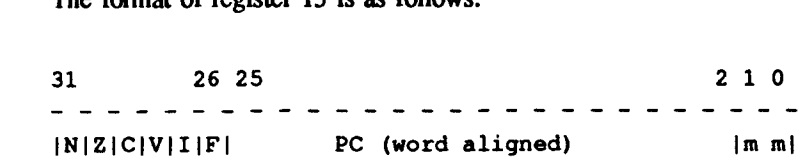
1.2.1 Memory organisation



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
  Absense 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 &3FFFFFC, 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.

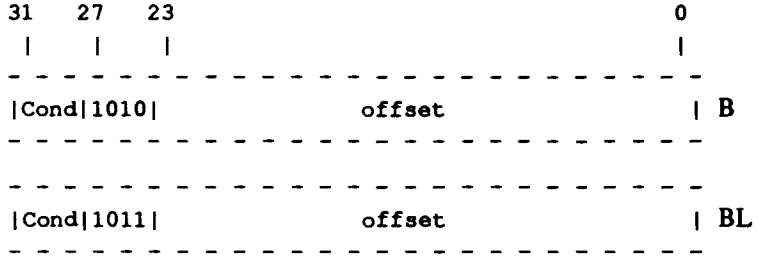
2. Instruction set

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 –&1FFFFF8 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
xxxxxxxx                 xxx
xxxxxxxx          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 EQuivalence
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
ADC	Add with Carry	Rd:= Rn+operand+C	N,Z,C,V
ADD	Add	Rd:= Rn+operand	N,Z,C,V
AND	And	Rd:= Rn AND operand	N,Z,C
BIC	Bit Clear	Rd:= Rn AND (NOT(operand))	N,Z,C
CMN	Compare Negated	Rn+operand	N,Z,C,V
CMP	Compare	Rn-operand	N,Z,C,V
EOR	Exclusive Or	Rd:= Rn EOR operand	N,Z,C
MOV	Move	Rd:= operand	N,Z,C
MVN	Move Not	Rd:= NOT operand	N,Z,C
ORR	Logical Or	Rd:= Rn OR operand	N,Z,C
RSB	Reverse Subtract	Rd:= operand-Rn	N,Z,C,V
RSC	Reverse Subtract with Carry	Rd:= operand-Rn-1+C	N,Z,C,V
SBC	Subtract with Carry	Rd:= Rn-operand-1+C	N,Z,C,V
SUB	Subtract	Rd:= Rn-operand	N,Z,C,V
TEQ	Test Equivalence	Rn EOR operand	N,Z,C
TST	Test and Mask	Rn AND operand	N,Z,C

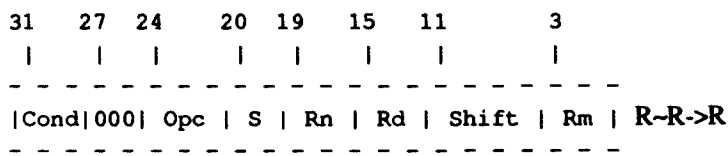
- The borrow operation on the RSC and SBC instructions is performed by *adding* the carry due to the hardware configuration of the CPU
- Rd is the destination register nextp Rn is a source register
- S2 is a register, possibly shifted by a constant or by a register, or an 8-bit data value shifted by 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.

ARM registers are linked to the ALU via two 32-bit read bus lines. On the first part of an instruction cycle (clock phase 1) the ALU receives two operands simultaneously; one passes from register to ALU directly, the other passes through the barrel shifter and may come from a register or from an immediate field in the instruction. The ALU performs its task immediately, and passes the result back to a register in the second half of the instruction cycle (clock phase 2).

2.2.1 Data processing on registers

(B25-B27 bit pattern 000)



- Rd is the destination register
- Rn and Rm are both source registers, and the contents of Rm can be shifted by a constant or by the contents of a register
- Opc is the opcode bit pattern as shown in the data processing opcodes table
- Shift is an 8 bit field specifying the operation of the barrel shifter.

The following four shift-types operate on the Rm register, shifting the contents by nnnnn bits:

nnnnn000	logical left	by 1 to 31 bits
nnnnn010	logical right	by 1 to 32 bits
nnnnn100	arithmetic right	by 1 to 32 bits
nnnnn110	rotate right	by 1 to 31 bits

Rotate right one bit does not require nnnnn:

00000110	rotate right	one bit with extend

Note: when the shift is by '1 to 32 bits' then zero is taken to mean 32 bits.

The following four shift-types operate on the Rm register, shifting the contents by N bits, where register Rsss holds the value of N:

Rsss0001	logical left	by Rs
Rsss0011	logical right	by Rs
Rsss0101	arithmetic right	by Rs
Rsss0111	rotate right	by Rs

Shifts specified by Rs require one additional execution cycle. Only the least significant byte of Rs is used, and signifies a shift of 0 to +255 bits. If shifting operations are not required, the bit-pattern of the shift field is set to 00000000. This is decoded as logical shift left 0 bits, and the carry bit is not changed by this action.

The shifting action is as follows, where *n* represents the number of bits shifted:

Logical shift left

```
C <- - bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb <- - 0
              < - - - - - - - - - - - -
```

The *n* least significant bits become zero, the carry becomes b(32–*n*). If the shift amount is zero, no shift is performed and the carry flag is not altered. If the shift amount lies in the range 1 to 32, the carry flag is set to b(32–n). If the shift amount is greater than 32, the carry flag is set to zero.

Logical shift right

```
0 - -> bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb - -> C
               - - - - - - - - - - - - >
```

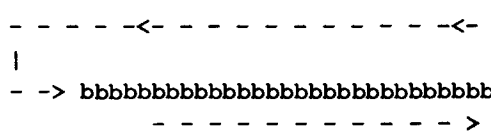
The *n* most significant bits become zero, the carry becomes b(*n*–1). If the shift amount is zero, no shift is performed and the carry flag is not altered. If the shift amount lies in the range 1 to 32, the carry flag is set to b(n–1). If the shift amount is greater than 32, the carry flag is set to zero.

Arithmetic shift right

```
- - -> xbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb - -> C
|       |      - - - - - - - - - - - - >
<- - -
```

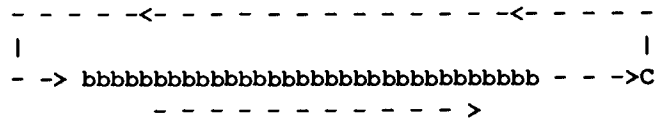
The *n* most significant bits become equal to b31 (that is on every single shift, bit 31 is duplicated).
If the shift amount is zero, no shift is performed and the carry flag is not altered. If the shift amount lies in the range 1 to 32, the carry flag is set to b(n–1). If the shift amount is greater than 32, the carry flag is set to b31.

Rotate right



On every bit of the shift, the content of b0 transfers to b31. The carry becomes b(*n*–1). That is the carry is set to the last bit rotated into b31. When no shift is performed the carry bit is not altered.

Rotate right with extend



n is always 1. The contents of the carry transfers to b31, the contents of b0 transfers to the carry.

The use of S (the set condition codes bit) will be described in the section below dealing with the second data processing instruction.

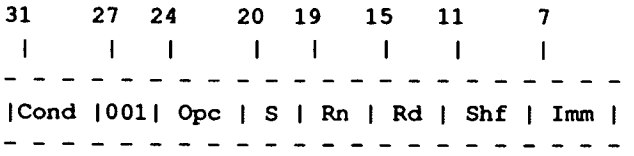
Syntax
Mnemonic {cond} {S} {P} Rd {, Rn}, Rm {, SHIFT}

- *{cond}* two character conditional code mnemonic
- *{S}* set condition codes if S present (the ARM assembler will automatically set condition codes for CMP, CMN, TEQ, TST)
- *{P}* make Rd = R15 in instructions where Rd is not actually written to. Also sets S bit. Used for changing PSR. (see section 2.2.5)

- Rd, Rn and Rm are expressions evaluating to a register number
- {Rn} is not required in instructions with only two operands
- {SHIFT} is {shiftname register} or {shiftname #expression} or {RRX}.

2.2.2 Data processing with register and immediate operand

(B25-B27 bit pattern 001)



- Imm is an 8-bit immediate field. The processor supplies bits B8-B31 as zero, and thereby zero-extends the field to 32 bits
- Shf is a 4-bit field defining a rotate right by Shf*2 of the 32-bit zero extended Imm field

Syntax

Mnemonic {cond} {S} {P} Rd {,Rn},#expression

- {cond} two character conditional code mnemonic
- {S} set condition codes if S present (the ARM assembler will automatically set condition codes for CMP, CMN, TEQ, TST)
- {P} make Rd = R15 in instructions where Rd is not actually written to. Also sets S bit. Used for changing PSR
- Rd and Rn are expressions evaluating to a register number
- {Rn} is not required in instructions with only two operands
- *#expression* the assembler will attempt to generate a shifted immediate 8-bit field to match the expression. Any number which cannot be expressed as a rotation by an even number of a value in the range 0-255 will be rejected. For example,

```
Eg: Mnemonic Rd,#&FF ; = 11111111  No error
    Mnemonic Rd,#&81 ; = 10000001  No error
    Mnemonic Rd,#&101; =100000001  Error reported by
                      ;             the assembler
```

The assembler fails when asked to process a number where the distance between the highest and lowest bit exceeds 8-bit positions.

The S bit, bit 20; the {S} mnemonic.

S should be reset to zero to preserve the current condition codes (the flags), and set to 1 to allow their update:

- when Rd is not R15, the condition codes are updated from the ALU flags
- when Rd is R15, the PSR is overwritten by the corresponding bits in the ALU result, though some bits can only be changed in particular modes.

So normal moves to R15 (that is Rd) are only 24 bits, moves with S set are 28 bits (full PC and user PSR: in non-user modes all 32 bits are used) while the result is used to set the PSR bits by the assembler only in CMP, CMN, TST, TEQ instructions when Rd is R15. For example:

```
ADDEQ R2,R4,R5
TEQS R4,#3
SUB R4,R5,R7,LSR R2
```

When Rm is R15, the value of the PC plus the PSR is presented to the ALU. When Rn is R15, the PC is presented without the PSR, that is those bits are 0.

2.2.3 Changing modes.

A TEQP instruction is used to change modes. For example:

```
TEQP R15,#2 changes to IRQ mode.
TEQP R15,#0 changes to user mode.
```

There is a one clock-cycle delay in mode change instructions due to the

pipelined nature of the processor. When a data-processing instruction is used to change processor mode, the next instruction has already been fetched and is waiting to be executed. If this instruction attempts to access any of registers R10-R14, it will use the wrong one because re-mapping has taken place. Registers R0-R9 are safe to use, because they will not be affected by the mode change.

A no-operation instruction such as BNV should be given to the processor to allow the re-mapping to take place before using R10-R14.

The action of TEQP R15,#n is subtle. Whenever R15 is presented to the processor as the Rn register, 24 bits are presented; the PSR bits are supplied as zero. The TEQ causes the value #n to be written into the register, and the P causes the PSR bits (now altered by #n) to be written back into R15. Since two of the PSR bits are the mode-control bits, the processor assumes its new mode.

The First Devices

The first CPUs have two minor faults in the shift logic:

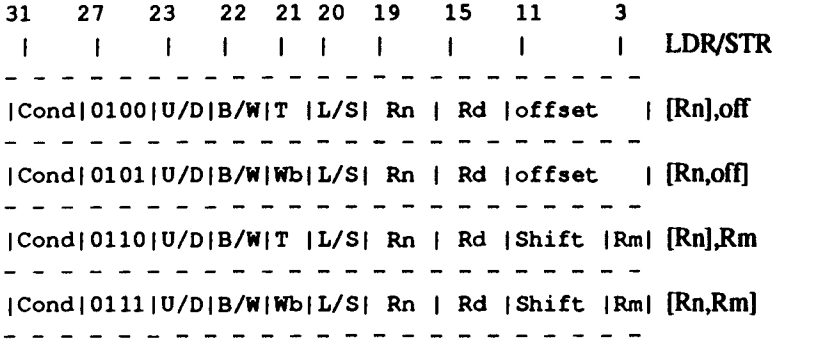
- when using RRX shifts with S set, the carry out is the result of ORing the top and bottom bits of the source together, instead of just the bottom bit.
- when using ROR with a register controlled shift and the register is greater than 31, the carry out will always be zero, instead of the bit MOD 32.

2.3 Single data transfer group

Bits B24-B27 are set to 11xx, giving four basic instruction types of the LDR (Load register) and STR (Store register) variety.

These instructions can load any register or save any register using any of the registers as a base-address pointer, which may then be modified by an offset. Loading or saving is specified by the L/S bit, and the direction of movement from the base by bit 23.

Any one of the 16 conditional tests may be applied to these instructions.



- Rn is the index register
- The Wb bit gives optional auto-increment and decrement addressing modes
- L/S 1: Rd becomes the operand that is LDR; data into register
- L/S 0: the operand becomes Rd that is STR; data out of register
- B/W 1: transfer byte between register and any byte address. On load the byte will be zero extended to a word
- B/W 0: transfer word between register and any word aligned address. If the address is not word aligned, the result of the transfer is not defined
- T 1: forces the translate output (the TRANS pin, pin 10) to be active during the data-transfer cycle, thereby allowing programs running in supervisor mode to load and save user memory areas
- The eight shift control bits are the same as those described in the data processing instructions
- the offset is either {offset} or {Rm{shifted}}.

2.3.1 [Rn,#off] is a pre-indexing addressing mode

Rn is the index register, modified by the 12-bit binary offset before it is used. The calculated address Rn+offset is only written back to Rn if the Wb bit is 1. For example:

```
Action:   LOAD/STORE to/from the indexed address
          Then only if Wb=1:
          Rn <-  Rn+offset  when U/D =1
          Rn <-  Rn-offset  when U/D =0
```

2.3.2 [Rn,Rm] is a pre-indexed addressing mode

Rn is the index register, modified by the offset, which is in register Rm, before it is used. The calculated address Rn+offset is only written back to Rn if the Wb bit is 1. For example:

```
Action:   LOAD/STORE to/from the indexed address
          Then only if Wb=1:
          Rn <-  Rn+Rm{shifted}  when U/D =1
          Rn <-  Rn-Rm{shifted}  when U/D =0
```

In both [Rn,#off] and [Rn,Rm] the value within the square brackets is evaluated and used as an address first: the processor evaluates Rn +/- offset and uses the result as the store or load address, and makes a temporary note of the address. Only if the Wb bit is set to 1 is the temporary value copied back into Rn.

2.3.3 [Rn],off is a post-indexed addressing mode

The index register is Rn; after it has been used, it is modified by the 12-bit binary offset. For example:

```
Action:   LOAD/STORE to/from the address in Rn
          Then always:
          Rn <-  Rn+offset  when U/D =1
          Rn <-  Rn-offset  when U/D =0
```

2.3.4 [Rn],Rm is a post-indexed addressing mode

The index register is Rn, but in this case the offset is in register Rm, and may be optionally modified before it is used. As described in section 2.3.3, Rn takes on its new value after it has been used. For example:

```
Action:   LOAD/STORE to/from the address in Rn
          then always:
          Rn <-  Rn+Rm{shifted}  when U/D =1
          Rn <-  Rn-Rm{shifted}  when U/D =0
```

The single data-transfer instructions will never affect the PSR, even when Rd or Rn is R15.

When using the contents of the PC register as the base address it must be remembered that it contains an address 8 byte addresses further on than that of the start of the current instruction.

Syntax

Mnemonic {cond} {B} {T} Rd, addressn

- *{cond}* two-character condition code mnemonic
- *{B}* if B is present then a byte transfer will take place, otherwise a word transfer will take place
- *{T}* if T is present the translate bit will be set
- *Rd* is an expression evaluating to a valid register number
- address can be:

  1. 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. If out of range, an error will be generated

  2. pre-indexed

[Rn]	offset of zero
[Rn,#expression]{!}	offset of {expression}
{!}	writeback the base register if present.
[Rn,{+or-}Rm{,shift}]{!}	offset of +or- contents of {index}

register,shifted by {shift}.

Rn and Rm are expressions evaluating to a valid register number

3. post-indexed

[Rn],#expression	offset of {expression}
[Rn],{+or–}Rm{,shift}	offset of +or– contents of {index} register shifted as in 2. above

Some examples are:

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

STR R1,[BASE,INDEX]! ;store R1 at BASE+INDEX (both
                     ; register contents) and write
                     ; back address to BASE

STR R1,[BASE],INDEX  ;store R1 at BASE and writeback
                     ;BASE+INDEX to BASE

LDR R1,[BASE,#17]    ;load R1 from contents of
                     ;BASE+17
                     ;Don't writeback

LDR R1,[BASE,INDEX,LSL #2] ;load R1 from contents of
                           ;BASE+INDEX*4
```

2.4 Block data transfer

Bits B24-B27 are set to 100x, giving two instruction types, LDM (Load multiple) and STM (STore Multiple). These instructions can load any registers or save any registers using any of the registers as the memory address (the base). The base is incremented or decremented a number of times to enable the register contents to be transferred to and from memory. Loading or saving is specified by the L/S bit, and the direction of movement from the base by the U/D bit.

The purpose of these instructions is to permit a defined register group to be transferred into contiguous memory words on consecutive CPU cycles.

Any one of the 16 conditional tests may be applied to these instructions.

```
31   27   23  22  21   20    19   15
 |    |    |   |   |    |     |    |        LDM/STM
- - - - - - - - - - - - - - - - - - - -
|Cond|1000|U/D| P | Wb | L/S | Rn |operand| post-
- - - - - - - - - - - - - - - - - - - -

- - - - - - - - - - - - - - - - - - - -
|Cond|1001|U/D| P | Wb | L/S | Rn |operand| pre-
- - - - - - - - - - - - - - - - - - - -
```

- Rn holds the base address
- L/S is the Load/Store bit:

  When L/S =0 data is stored in memory – STM
  When L/S =1 data is loaded into registers – LDM

- operand: this is a bit-image of the registers which need to be transferred.

  ```
  R15             R0
  |               |
  1111111111111111 : (with L/S=0) stack all registers.

  0000000000000010 : (with L/S=0) stack register R1.
  ```

  Note: the-bit image is interpreted directly by the ARM hardware and the speed of block data transfer operations is directly proportional to the number of registers involved

- U/D is the Up/Down bit:

  ```
  1000    1  for up post-increment
  1001    1  for up pre-increment
  1000    0  for down post-decrement
  1001    0  for down pre-decrement
  ```

  Register values are stacked in register order. This is always the case and cannot be altered. The lowest register always occupies the lowest memory location and registers contents are written to, or read from,

memory starting with the lowest numbered register.

- the P bit. The PC is represented by bit B15=1, and the condition codes are saved with it in the case of a STACK (STM) instruction. When B15=1 in an UNSTACK (LDM) instruction, the bits of the PSR are loaded if the P bit (B22) is 1 and not loaded if it is 0
- Wb is the writeback bit. The Wb bit indicates whether the base register is to be updated, as explained in the Single Data Transfer Group of instructions. When writeback is specified, the base is written back during the second cycle of the instruction. During a stacking operation (STM), the first register is written out during the first cycle. An STM which includes storing the base, with the base as the lowest-numbered register in the list, will therefore store the unchanged value, whereas with a base register which is not the lowest-numbered it will be the new value which is stored. An LDM will always overwrite the updated base, if the base is in the list.

  With the writeback bit set these instructions can be used to stack and unstack multiple registers. (See appendix E.)

Special notes:

1 When the base is the PC, the PSR bits will be used to form the address as well, so unless all interrupts are enabled and all flags are zero (making bits B26-B31 of R15 all zero) an address exception will occur. Writeback is never allowed when the base is the PC.

2 In the case of a stack/STM instruction, and when the processor is in user mode, the P bit is ignored. In other modes it may be used to force transfers from the user-mode register-bank. In this case, writeback will also be to the user-mode register-bank, though the base-register will be fetched from the current bank. In this situation, the writeback must be turned off (Wb=0).

3 The action of the instructions when {operand} is zero is undefined and may affect registers or store.

2.4.1 Assembler syntax

Mnemonic {cond} FD/ED/FA/EA/IA/IB/DA/DB R*n {!}*, R*list{^}*

- *{cond}* two-character condition code mnemonic
- FD/ED/FA/EA define pre/post indexing and up/down bit. It is assumed that these instructions will normally be used for stack operations. The

F and E refer to notion of a "full" or "empty" stack, i.e. whether the stack poiner register is pointing to a word holding valid data (full) or a word holding data which is not needed (empty). More information on the use of stack is given in appendix E

IA/IB/DA/DB are mnemonics used for non-stacking operations and mean Increment After, Increment Before, Decrement After and Decrement Before. For further information the LDM and STM mnemonics, see the ARM assembler manual

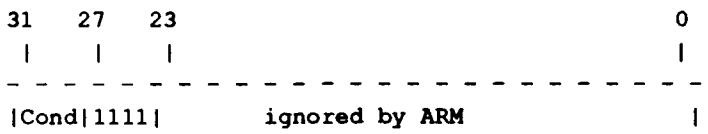
- R*n* is an expression evaluating to a valid register number
- R*list* can be either a list of registers enclosed in {} or an expression evaluating to the 16 bit operand
- *{!}* is the optional writeback
- *{^}* means set PSR if present (note different meanings for PSR bit in LDM and STM)

For examples

```
LDMFD SP!,{R0,R1,R2} ;unstack 3 registers
STMIA BASE,{R0-R15}  ;save all regs
```

2.5 Supervisor calls

bits B24-B27 are set to 1111. This is the software interrupt, SWI. Any one of the 16 conditional tests may be applied to this single instruction.



The PC word and PSR are saved in R14_SVC, with the PC adjusted to point to the word after the SWI instruction. The PC 24 bits are set to 8 (4*2) and mm of the PSR is set to SVC mode and the processor continues

(see the section on traps and vectors).

Syntax

`SWI`*{cond} expression*

- *{cond}* two-character condition code mnemonic
- the expression is ignored by ARM, but can be interpreted by other hardware. In the ARM evaluation system SWI with its expression is used to link ARM machine code to the Acorn 6502 MOS as used in the BBC Microcomputer. For further information on the SWI calls to the I/O MOS, consult the *ARM Assembler reference manual*

For examples:

```
SWI   ReadC
SWI   WriteI+"k"
```

3. Interrupts

Interrupts are channeled through vectors in RAM:

vector summary

```
Vector 0 : 0000000  (0)  reset
       1 : 0000004  (4)  undefined instruction
       2 : 0000008  (8)  software interrupt
       3 : 000000C  (12) abort (prefetch)
       4 : 0000010  (16) abort (data)
       5 : 0000014  (20) address exception
       6 : 0000018  (24) IRQ
       7 : 000001C  (28) FIQ
```

The interrupt priority is:

Reset	(highest)
Address exception	
Data Abort	
FIQ	
IRQ	
Prefetch Abort	
Undefined Instruction	
Software interrupt	(lowest)

3.1 Reset

When reset goes low ARM will stop the currently executing instruction and start executing no-ops.

When reset goes high again it will:

- save R15 in R14_SVC
- set M0, M1 to SVC mode and set the F and I bits in the PC word
- jump to RAM vector 0 by setting the PC to 0 (4*0).

Note: due to ARM pipelining the processor actually saves (R15)-4 in the SVC register R4.

3.2 Address exception trap

When an address exception that is a data transfer at an address above &3FFFFFF is seen, ARM will:

(1) complete the instruction if it is a block data transfer type, eg: LDM, or return to the state just before execution (single data transfer type eg: LDR) - see section 3.3.

(2) save R15 in R14_SVC

(3) set M0, M1 to SVC mode and set the I bit in the PC word

(4) jump to RAM vector 5 by setting the PC to 20 (4*5).

A return from this trap can be achieved by subtracting 4 from R14_SVC and placing the result in R15 and PSR. This will return the PC to the instruction after the one causing the trap.

Normally an address exception will prove fatal, at least for the currently running task which caused it. However, in a simple implementation the trap might be used to send an appropriate warning message to an I/O device.

A more sophisticated use would be to invoke a paged-memory system using a 32-bit address bus.

LDM and STM (see section 2.4) will only cause address exceptions on the first data transfer of the instruction. A transaction which starts in the legal area and moves into the illegal area will not cause an address exception, and will attempt to access store addressed by the bottom 26 bits.

3.3 Abort

The abort signal is externally generated by a memory management system to alert the processor that a current instruction is handling memory incorrectly or that a virtual-memory system is in use. ARM checks for existence of abort at the end of the first phase of each bus cycle. Note that on some ARM development PCBs the abort line may be tied to ground and therefore be inoperative.

When successfully aborted, the ARM will respond in one of two ways:

(1) if the abort occurs during an instruction prefetch, the prefetched instruction is marked as invalid; and when it comes to execution, it is re-interpreted as shown in the examples below

(2) if the abort occurs during a data access, the action depends upon the instruction type. Data transfer instructions (e.g. LDR) are aborted as though the instruction had not been executed. The LDM and STM instructions are completed, and if writeback is set, the base is *always* updated, even if the instruction would have overwritten it (i.e. LDM with base in list). Then:

For example:

- save R15 in R14_SVC
- set M0, M1 to SVC mode and set the I bit in the PC word
- jump to RAM vector 3 or 4 by setting the PC to 12 (4*3) for a prefetch abort, or 16 (4*4) for a data abort.

Continue after an instruction prefetch abort by subtracting 4 from R14_SVC and placing the result in R15 and PSR. A data-access abort requires any auto-indexing to be reversed before returning to re-execute the offending instruction, the return being achieved by subtracting 8 from R14_SVC and placing the result in R15 and PSR.

3.4 FIQ

The FIQ (Fast interrupt request) signal is designed to be used as a data transfer or channel process. Its effect may be masked out by setting the F flag in the PSR (but note that this is not possible from user mode). ARM checks for the existence of FIQ at the end of instructions. When the ARM responded to a fast interrupt request it will:

(1) save R15 in R14_FIQ

(2) set M0, M1 to FIQ mode and set the F and I bits in the PC word

(3) jump to RAM vector 7 by setting the PC to 28 (4*7).

Return from FIQ by programming the assembler line: SUBS PC,R14_FIQ,#4.

3.5 IRQ

The IRQ (Interrupt request) signal is a normal interrupt. 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 PC (but note that this is not possible from user mode). ARM checks for the existence of IRQ at the end of instructions. When successfully interrupted by an IRQ, the ARM will:

(1) save R15 in R14_IRQ

(2) set M0, M1 to IRQ mode and set the I bit in the PC word

(3) jump to RAM vector 6 by setting the PC to 24 (4*6).

Return from IRQ by programming the assembler line: SUBS PC,R14_IRQ,#4.

3.6 Undefined instruction trap

Undefined instructions are identical to supervisor calls except that the program flow diverts to a different vector. The undefined instructions are reserved for future expansion and the trap may be used for the emulation of future instruction set enhancements, such as floating-point arithmetic. When an undefined instruction is seen, ARM will:

(1) save R15 in R14_SVC

(2) set M0, M1 to SVC mode and set the I bit in the PC word

(3) jump to RAM vector 1 by setting the PC to 4 (4*1).

Return from trap by transferring R14_SVC to R15 and PSR.

3.7 Software interrupt

The software interrupt is used for entering supervisor mode (mode 3). From mode 3, it is possible to select any other mode. ARM will:

(1) save R15 in R14_SVC

(2) set M0, M1 to SVC mode and set the I bit in the PC word

(3) set PC to 8 (4*2).

Return from SWI by transferring R14_SVC to R15 and PSR. See the supervisor call instruction in section 2.

These are byte addresses, and each group of four bytes will normally contain a 32-bit branch instruction pointing to the relevant routine. The exception is FIQ, where the routine might reside at 000001C onwards.

4. Appendix A

4.1 Instruction speeds

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.

If the condition is met the instructions take various S and N cycles:

R~# ->Rd	1 S	+ *ShiftTime* + *R15time*
R~R ->Rd	1 S	+ *ShiftTime* + *R15time*
LDR	2 S + 1 N	+ *ShiftTime* + *R15time*
STR	2 N	+ *ShiftTime*
LDM	(n+1)S + *R15time*	
STM	(n–1)S + 2 N	
B,BL	2 S + 1 N	
SWI	2 S + 1 N	

ShiftTime is 1 S for SHIFT(Rs)
R15time is 1 S + 1 N if R15 written

n is the number of registers transferred in a LDM or STM.

If the condition is not true all instructions take one S cycle.

S is a sequential cycle or a cycle which does not require memory at all.
N is a non-sequential cycle.

With the initial second-processor product S cycles take 3/20μS. N cycles take 6/20μS. This corresponds to the 20MHz crystal frequency divided by 3 or 6.

5. Appendix B

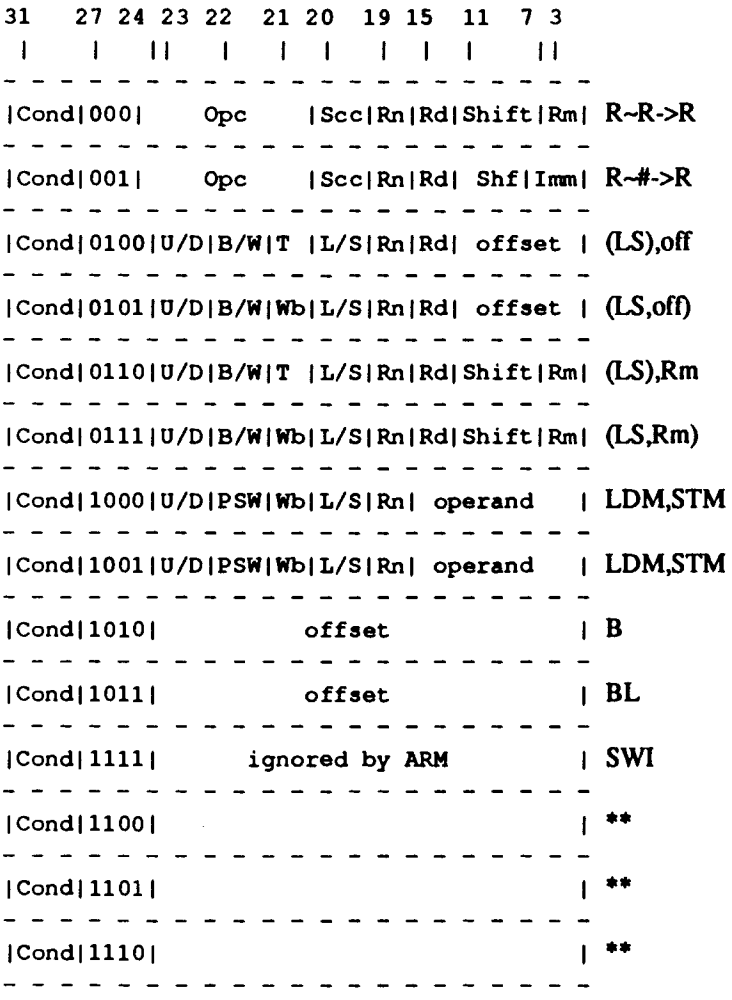
5.1 Virtual memory concept

On most systems using the ARM only part of the 64 Mbyte memory capacity will be represented by physical memory, but by using virtual memory techniques an ARM program can be written which makes use of the total memory.

The mechanism for supporting virtual memory is to provide a limited amount of high-speed memory that can be accessed by the CPU directly, while maintaining an image of a larger amount of memory on secondary storage devices such as hard disc. When the CPU tries to access an address which exists in the virtual memory but not in the physical memory, the attempt is postponed while the address map of the physical memory is adjusted so that it conforms with the virtual address requested. The ARM has the capability to stop an instruction which references memory that is not physically present, alter the memory, and re start the instruction. The code to do this is available from Acorn.

6. Appendix C

6.1 Instruction set summary



The instructions of the form Cond 110X and Cond 1110 and marked "**" will cause undefined instruction traps. These codes are reserved for future internal or coprocessor expansion.

7. Appendix E

7.1 Notional stacking

7.1.1 Stacking

Push to stack
Various instructions may be used to save the ARM registers on a stack.

There are four types of instruction which PUSH register values on to a stack. They are:

`STMFD`	Full stack, Descending (uses Pre-Decrement)
`STMED`	Empty stack, Descending (uses Post-Decrement)
`STMFA`	Full stack, Ascending (uses Pre-Increment)
`STMEA`	Empty stack, Ascending (uses Post-Increment)

For example: `STMEA Rn!,{R6,R3,R7,R8}`

which may also be written: `STMEA Rn!,{R6-R8,R3}`.

STMEA in action
Prior to the instruction, assume that a stack holding three values already exists, and that more values need to be pushed on to it:

```
      |     |
      |     |
      |     |
Rn->  |     |<-empty location
      | V3  |
      | V2  |
      | V1  | stack base
       - - -
```

The stack is ascending, and the location currently pointed to is deemed to be empty. Then, after `STMEA Rn!,{R6,R3,R7,R8}` the stack grows.

```
|      |<-Rn
| R8   |
| R7   |
| R6   |
| R3   |
| V3   |
| V2   |
| V1   | stack base
 - - -
```

Notice that register values are stacked in register order. This is always the case and cannot be altered. The lowest register always occupies the lowest memory location and registers are placed on, or removed from, the stack starting with the lowest numbered register. This can be seen in the next example, which shows the order of stacking following two 'full stack descending' instructions.

`STMFD` in action
For example: `STMFD R`*n*`!,{R6,R3,R7,R8}`
`STMFD R`*n*`!,{R0-R4}`

```
|    |<- Rn sets stack base
 - - -
| R8 |
| R7 |
| R6 |
| R3 |<- Rn after 1st instruction
| R4 |
| R3 |
| R2 |
| R1 |
| R0 |<- Rn after 2nd instruction
|    |
```

Pop from stack

There are four types of instruction which pop register values from a stack. They are:

`LDMEA`	Empty stack, Ascending. (uses Pre-Decrement)
`LDMFA`	Full stack, Ascending. (uses Post-Decrement)
`LDMED`	Empty stack, Descending. (uses Pre-Increment)
`LDMFD`	Full stack, Descending. (uses Post-Increment)

To recover one set of the saved registers from the stack set up by `STMFD R{\ em n!,{R0-R4}`, the instruction `LDMFD R n!,{R0-R4}` would be used. Afterwards, Rn will point to location Lx.

```
 - - -
| R8  |
| R7  |
| R6  |
| R3  |<- Lx
| R4  |
| R3  |
| R2  |
| R1  |
| R0  |<- Rn
|     |
```

Special points

- When Rn is in the stacking list.

  The base register Rn may be pushed onto the stack and if write-back is not in operation, no problem will occur. If write back is in operation, the STM is performed in the following order:

  Write lowest-numbered register to memory.
  Perform the write back.
  Write other registers to memory in ascending order.

  Thus, if Rn is the lowest-numbered register in the list, its original value is stored. Otherwise, its written back value is stored.

  If Rn is popped from the stack, the pop operation will continue successfully: the entire block-transfer runs on an internal copy of the Rn value, and will not be aware that register Rn has been loaded with a new value. (The writeback action occurs in the second cycle of the instruction.)

  Pushing Rn is not forbidden when writeback is in operation: but if Rn

is not the lowest-numbered register then at the end of the second cycle it will accept the written back value.

- When R15, the PC register, is in the stacking list.

  When R15 is pushed on to the stack, all the PSR condition codes are saved as well.

  When R15 is popped from the stack, the PSR condition codes are only included if the symbol ^ is coded following the register list. The condition codes included will in any case only be those which may be modified in the currently selected ARM mode. For example,

  ```
  LDMFD SP!,{FP,PC}^
  ```

- When Rn, the base register, is R15.

  When the PC is used as the base register, the PSR condition code bits form part of the 32-bit address. Unless all flags are zero and the interrupts enabled, an address greater than &3FFFFFF will be formed. This is an address exception and will be noticed by the ARM processor.

  Writeback is switched off when PC is the base register.

- The register list {R0,R1,etc} is used by the assembler to form a 16 bit operand where the setting of a bit indicates which register contents will move.

- In order to force the saving of the user mode registers when executing in a different mode, ^ should be coded following the register list. For example,

  ```
  STMFD R0,{R0-R15}^
  ```

- The operation of placing or removing registers to and from the stack starting with the lowest-numbered register is independent of stack type and exists to ensure that if a data abort occurs during a machine instruction, the PC is preserved.


Acorn
The choice of experience.