ARM Datasheet

Part No 1 85250 0360 0
Issue No 1.0
17 March 1987

ISBN 1 85250 026 3

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

Contents

1. Introduction

The ARM (Acorn RISC Machine) is a general purpose 32-bit single-chip microprocessor. The 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. This simplification results in a high instruction throughput and a good real-time interrupt response from a small and cost-effective chip.

The instruction set comprises nine basic instruction types. Two of these make use of the on-chip arithmetic logic unit (ALU), barrel shifter and multiplier to perform high-speed operations on the data in a bank of 27 registers, each 32 bits wide. Two instruction types control the transfer of data between main memory and the register bank, one optimised for flexibility of addressing and the other for rapid context switching. Two instructions control the flow and privilege level of execution, and the remaining three types are dedicated to the control of external Co-Processors which allow the functionality of the instruction set to be extended off-chip in an open and uniform way.

The ARM instruction set has proved to be 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.

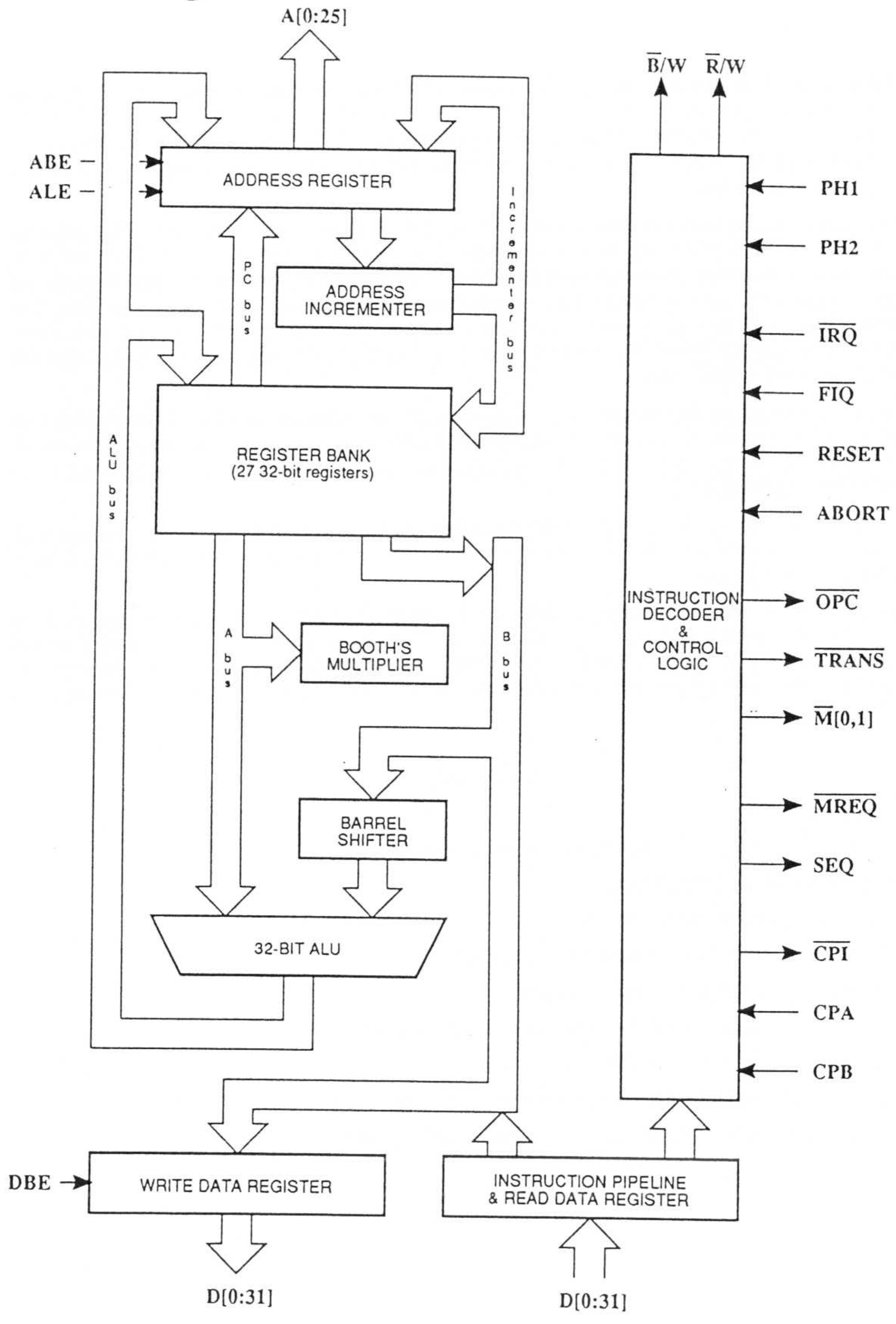
Pipelining is employed so that all parts of the processing and memory systems can operate continuously. Typically, while one instruction is being executed, its successor is being decoded, and a third instruction is being fetched from memory.

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

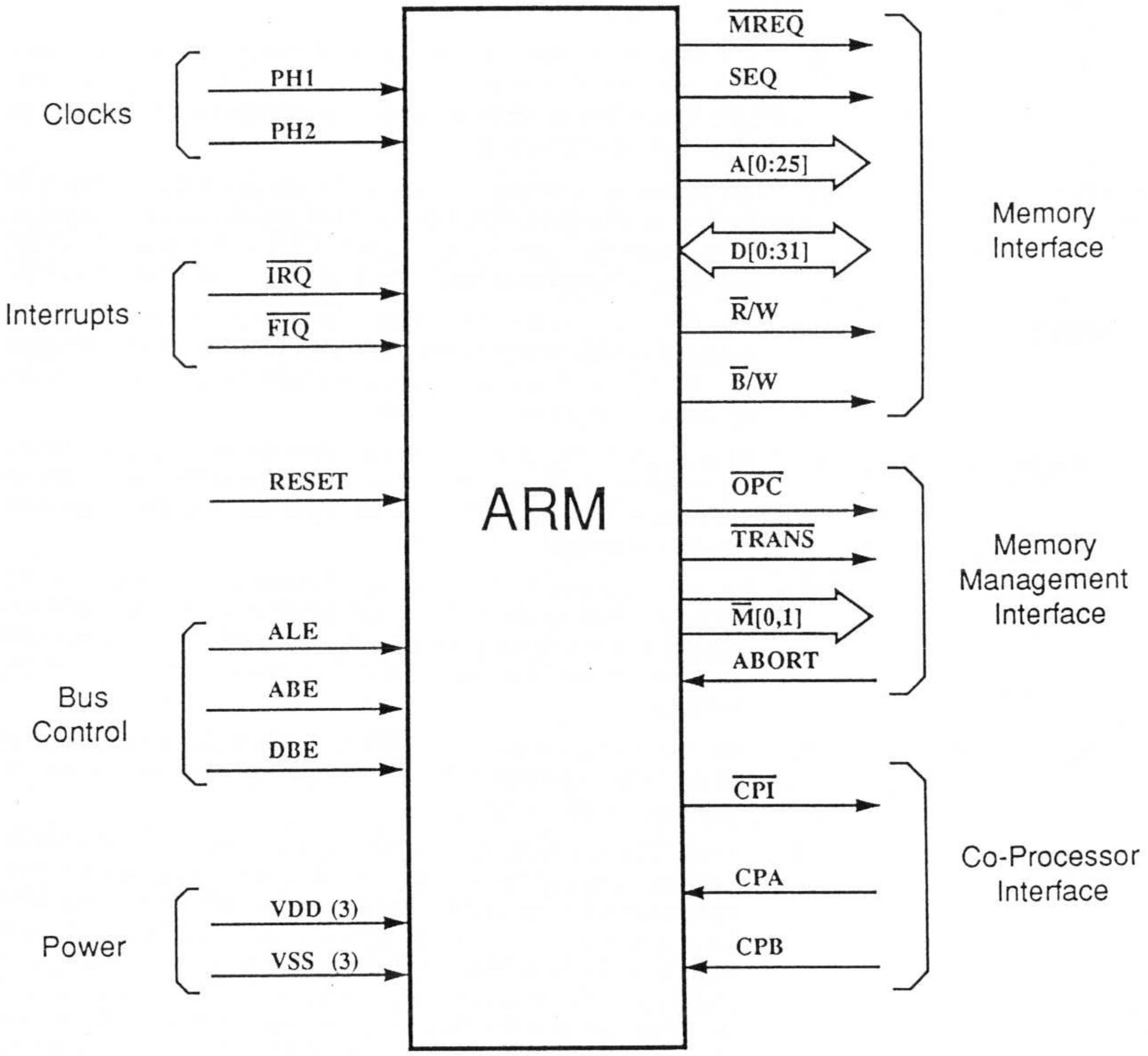
FEATURES

* 32-bit data bus
* 26-bit address bus giving a 64-MByte uniform address space
* Support for virtual memory systems
* Simple but powerful instruction set
* Co-Processor interface for instruction set extension
* Good high-level language compiler support
* Peak execution rate of 10 million instructions per second (MIPS)
* Fast interrupt response for real-time applications
* Low power consumption (0.1 W typical) with a single +5 V supply
* 84-pin JEDEC B leadless chip carrier or plastic leaded chip carrier

2. Block Diagram



3. Functional Diagram



4. Description of Signals

Name	Pin	Type	Description
PH2	1	ICk	Phase two clock.
PH1	2	ICk	Phase one clock.
$\overline{\text{R}}$/W	3	OC	Not read / write. When HIGH this signal indicates a processor write cycle; when LOW, a read cycle. It becomes valid during phase 2 of the cycle before that to which it refers, and remains valid to the end of phase 1 of the referenced cycle.
$\overline{\text{OPC}}$	4	OC	Not op-code fetch. When LOW this signal indicates that the processor is fetching an instruction from memory; when HIGH data (if anything) is being transferred. The signal becomes valid during phase 2 of the previous cycle, remaining valid through phase 1 of the referenced cycle.
$\overline{\text{MREQ}}$	5	OC	Not memory request. This signal, when LOW, indicates that the processor requires memory access during the following cycle. The signal becomes valid during phase 1, remaining valid through phase 2 of the cycle preceding that to which it refers.
ABORT	6	IT	Memory abort. This is an input which allows the memory system to tell the processor that a requested access is not allowed. The signal must be valid before the end of phase 1 of the cycle during which the memory transfer is attempted.
$\overline{\text{IRQ}}$	7	IT	Not interrupt request. This is an asynchronous interrupt request to the processor which causes it to be interrupted if taken LOW when the appropriate enable in the processor is active. The signal is level sensitive and must be held LOW until a suitable response is received from the processor.
$\overline{\text{FIQ}}$	8	IT	Not fast interrupt request. As $\overline{\text{IRQ}}$, but with higher priority. May be taken LOW asynchronously to interrupt the processor when the appropriate enable is active.
RESET	9	IT	Reset. This is a level sensitive input signal which is used to start the processor from a known address. A HIGH level will cause the instruction being executed to terminate abnormally. When **RESET** becomes LOW for at least one clock cycle, the processor will re-start from address 0. **RESET** must remain HIGH for at least two clock cycles, and during the HIGH period the processor will perform dummy instruction fetches with the address incrementing from the point where reset was activated. The address value will overflow to zero if **RESET** is held beyond the maximum address limit.
$\overline{\text{TRANS}}$	10	OC	Not memory translate. When this signal is LOW it indicates that the processor is in user mode, or that the supervisor is using a single transfer instruction with the force translate bit active. It may be used to tell memory management hardware when translation of the addresses should be turned on, or as an indicator of non-user mode activity.
VDD	11,32,55	PWR	Supply.

VSS	33,54,75	PWR	Supply.
$\overline{\text{M}}$[1,0]	13,14	OC	Not processor mode. These are output signals which are the inverses of the internal status bits indicating the processor operation mode.
SEQ	15	OC	Sequential address. This is an output signal. It will become HIGH when either: • the address for the next cycle is being generated in the address incrementer, so will be equal to the present address (in bytes) plus 4, *or* • during a cycle which did not use memory (**$\overline{\text{MREQ}}$** inactive), when the next cycle will use memory and the address will be the same as the current address. The signal becomes valid during phase 1 and remains so through phase 2 of the cycle before the cycle whose address it anticipates. It may be used, in combination with the low-order address lines, to indicate that the next cycle can use a fast memory mode (for example DRAM page mode) and/or to by-pass the address translation system.
ALE	16	IT	Address latch enable. This input to the processor is used to control transparent latches on the address outputs. Normally the addresses change during phase 2 to the value required during the next cycle, but for direct interfacing to ROMs they are required to be stable to the end of phase 2. Taking **ALE** LOW until the end of phase 2 will ensure that this happens. If the system does not require address lines to be held in this way, **ALE** may be held permanently HIGH. The **ALE** latch is dynamic, and **ALE** should not be held LOW indefinitely.
A[25:0]	17-31,34-44	OCZ	Addresses. This is the processor address bus. If **ALE** (address latch enable) is HIGH, the addresses become valid during phase 2 of the cycle before the one to which they refer and remain so during phase 1 of the referenced cycle. Their stable period may be controlled by **ALE** as described above.
ABE	45	IC	Address bus enable. This is an input signal which, when LOW, puts the address bus drivers into a high impedance state. **ABE** may be tied HIGH when there is no system requirement to turn off the address drivers.
D[0:31]	46-53,56-74, 77-81	IOTZ	Data Bus. These are bi-directional signal paths which are used for data transfers between the processor and external memory, as follows: • during read cycles (when $\overline{\text{R}}$/W = 0), the input data must be valid before the end of phase 2 of the transfer cycle • during write cycles (when $\overline{\text{R}}$/W = 1), the output data will become valid during phase 1 and remain so throughout phase 2 of the transfer cycle.
DBE	83	IT	Data bus enable. This is an input signal which, when LOW, forces data bus drivers into a high impedance state. (The drivers will always be high impedance except during write cycles, and **DBE** may be tied HIGH in systems which do not require the data bus for DMA or similar activities.)
$\overline{\text{B}}$/W	84	OC	Not byte / word. This is an output signal used by the processor to indicate to the external memory system when a data transfer of a byte

			length is required. The signal is HIGH for word transfers and LOW for byte transfers and is valid for both read and write cycles. The signal will become valid during phase 2 of the cycle before the one during which the transfer will take place. It will remain stable throughout phase 1 of the transfer cycle.
$\overline{\text{CPI}}$	82	OC	Co-Processor instruction. When ARM executes a Co-Processor instruction, it will take this output LOW and wait for a response from the Co-Processor. The action taken will depend on this response, which the Co-Processor signals on the **CPA** and **CPB** inputs.
CPB	12	IT	Co-Processor busy. A Co-Processor which is capable of performing the operation which ARM is requesting (by asserting $\overline{\text{CPI}}$), but cannot commit to starting it immediately, should indicate this by letting **CPB** float HIGH. When the Co-Processor is ready to start it should take **CPB** LOW. ARM samples **CPB** at the end of phase 1 of the cycle when $\overline{\text{CPI}}$ is LOW.
CPA	76	IT	Co-Processor absent. A Co-Processor which is capable of performing the operation which ARM is requesting (by asserting $\overline{\text{CPI}}$) should take **CPA** LOW immediately. If **CPA** is HIGH at the end of phase 1 of the cycle when $\overline{\text{CPI}}$ is LOW, ARM will abort the Co-Processor handshake and take the undefined instruction trap. If **CPA** is LOW and remains LOW, ARM will busy-wait until **CPB** is LOW and then complete the Co-Processor instruction.

Key to Signal Types

ICk	Unbuffered clock inputs
IT	Input with TTL compatible levels
OC	Output with CMOS compatible levels
OCZ	3-state output with CMOS compatible levels
IOTZ	Bi-directional 3-state input/output with TTL compatible levels
PWR	Power pins

5. Programmers' Model

5.1 Introduction

ARM has a 32 bit data bus and a 26 bit address bus. The data types the processor supports are Bytes (8 bits) and Words (32 bits), where words must be aligned to four byte boundaries. Instructions are exactly one word, and data operations (e.g. ADD) are only performed on word quantities. Load and store operations can transfer either bytes or words.

ARM supports four modes of operation, including protected supervisor and interrupt handling modes.

5.2 Registers

The processor has 27 32-bit registers, 16 of which are visible to the programmer at any time. The visible subset depends on the processor mode; special registers are switched in to support interrupt and supervisor processing. The register bank organisation is shown in figure 1.

User mode is the normal program execution state; registers R0-15 are directly accessible.

All registers are general purpose and may be used to hold data or address values, except that register R15 contains the Program Counter (PC) and the Processor Status Register (PSR). Special bits in some instructions allow the PC and PSR to be treated together or separately as required. Figure 2 shows the allocation of bits within R15.

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 and R14_fiq are used similarly to hold the return values of R15 when interrupts and exceptions arise, or when Branch and Link instructions are executed within supervisor or interrupt routines.

The FIQ processing state (described in the Exceptions section) has seven private registers mapped to R8-14 (R8_fiq-R14_fiq). Many FIQ programs will not need to save any registers.

The IRQ processing state has two private registers mapped to R13 and R14 (R13_irq and R14_irq).

Supervisor mode (entered on SWI instructions and other traps) has two private registers mapped to R13 and R14 (R13_svc and R14_svc).

The two private registers allow the IRQ and supervisor modes each to have a private stack pointer and link register. Supervisor and IRQ mode programs are expected to save the User state on their respective stacks and then use the User registers, remembering to restore the User state before returning.

In User mode only the N, Z, C and V bits of the PSR may be changed. The I, F and Mode flags will change only when an exception arises. In supervisor and interrupt modes all flags may be manipulated directly.

5.3 Exceptions

Exceptions arise whenever there is a need for the normal flow of program execution to be broken, so that (for instance) 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.

ARM handles exceptions by making use of the banked registers to save state. The old PC and PSR are copied into the appropriate R14, and the PC and processor mode bits are forced to a value which depends on the exception. Interrupt disable flags are set where required to prevent otherwise unmanageable nestings

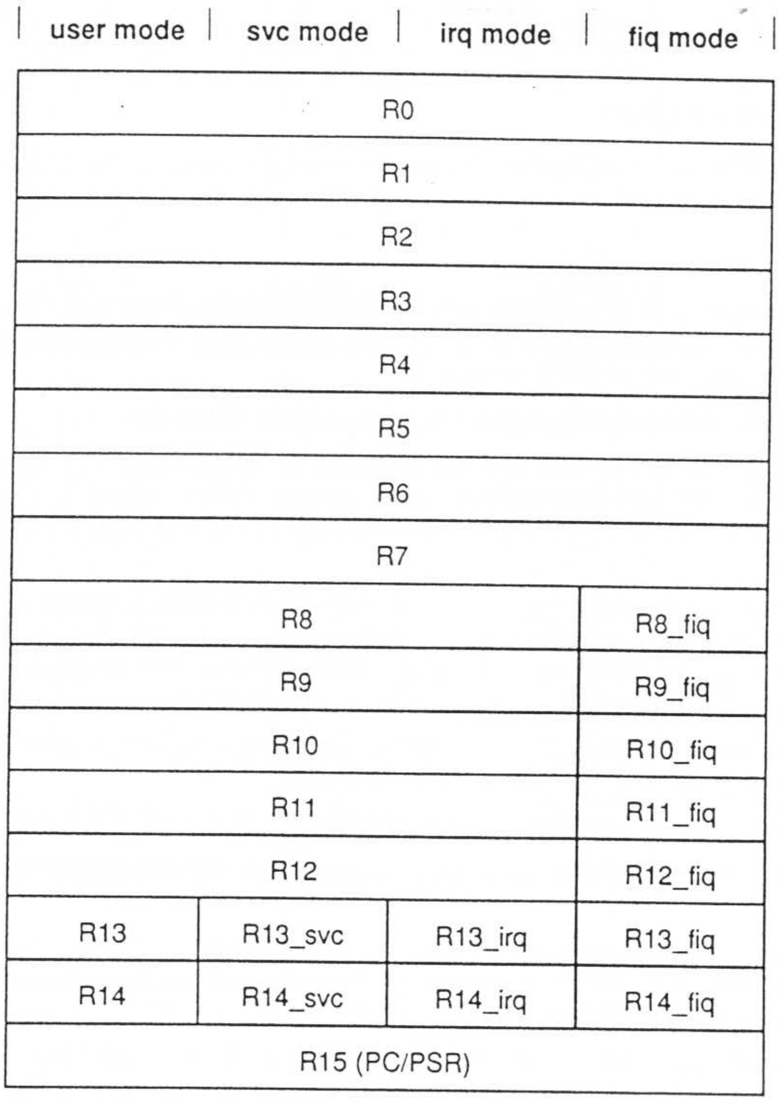


Figure 1: Register Organisation

of exceptions. In the case of a re-entrant interrupt handler, R14 should be saved onto a stack in main memory before rè-enabling the interrupt. When multiple exceptions arise simultaneously a fixed priority determines the order in which they are handled.

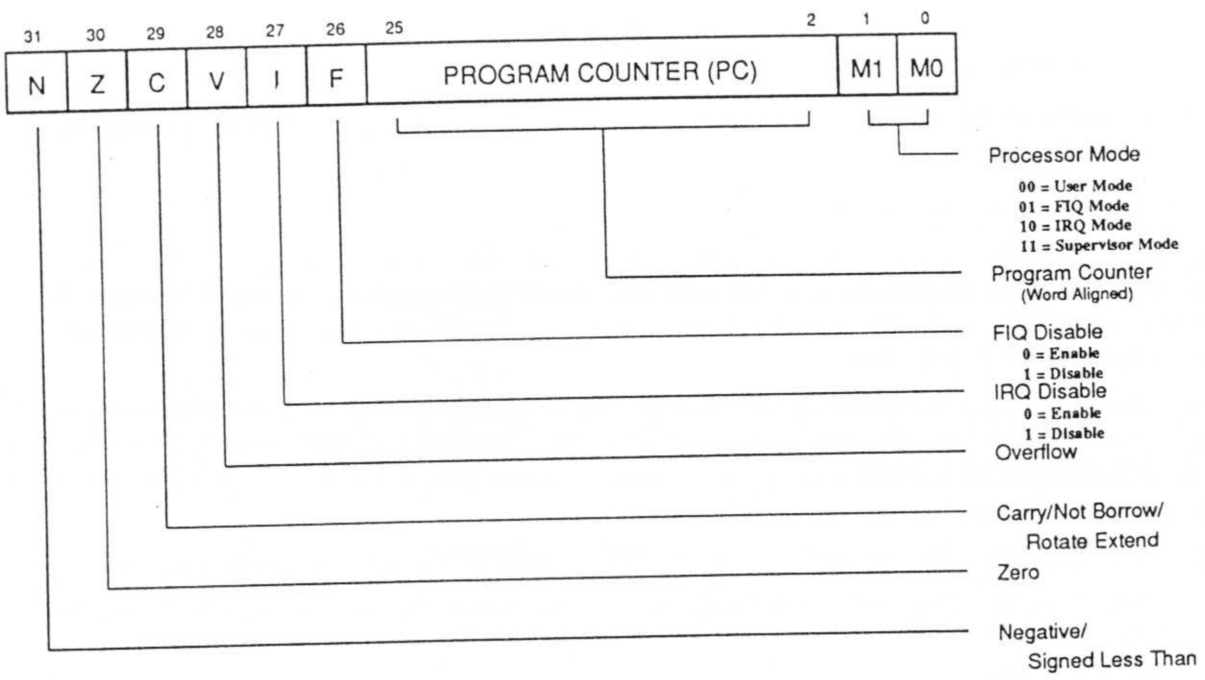


Figure 2: The Program Counter (PC) and Processor Status Register (PSR)

5.3.1 FIQ

The FIQ (Fast Interrupt reQuest) exception is externally generated by taking the $\overline{\text{FIQ}}$ pin LOW. This input can accept asynchronous transitions, and is delayed by one clock cycle for synchronisation before it can affect the processor execution flow. It 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, so that the overhead of context switching is minimised. The FIQ exception may be disabled by setting the F flag in the PSR (but note that this is not possible from user mode). If the F flag is clear ARM checks for a LOW level on the output of the FIQ synchroniser at the end of each instruction. When ARM is FIQed it will:

(1) save R15 in R14_fiq;

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

(3) force the PC to fetch the next instruction from address 1CH.

To return normally from FIQ use SUBS PC,R14_fiq,#4. This will resume execution of the interrupted code sequence, and restore the original mode and interrupt enable state.

5.3.2 IRQ

The IRQ (Interrupt ReQuest) exception is a normal interrupt caused by a LOW level on the $\overline{\text{IRQ}}$ pin. 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). If the I flag is clear ARM checks for a LOW level on the output of the IRQ synchroniser at the end of each instruction. When successfully IRQed ARM will:

(1) save R15 in R14_irq;

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

(3) force the PC to fetch the next instruction from address 18H.

To return normally from IRQ use SUBS PC,R14_irq,#4. This will restore the original processor state and thereby re-enable IRQ.

5.3.3 Address exception trap

An address exception arises whenever a data transfer is attempted with a calculated address above 3FFFFFFH. The ARM address bus is 26 bits wide, and an address calculation will have a 32-bit result. If this result has a logic "1" in any of the top 6 bits it is assumed that the address overflow is an error, and the address exception trap is taken.

Note that a branch cannot cause an address exception, and a block data transfer instruction which starts in the legal area but increments into the illegal area will not trap. The check is peformed only on the address of the first word to be transferred.

When an address exception is seen ARM will:

(1) if the data transfer was a store, force it to load. (This protects the memory from spurious writing.)

(2) complete the instruction, but prevent internal state changes where possible. The state changes are the same as if the instruction had aborted on the data transfer.

(3) save R15 in R14_svc;

(4) force M0, M1 to supervisor mode and set the I bit in the PC word;

(5) force the PC to fetch the next instruction from address 14H.

Normally an address exception is caused by erroneous code, and it is inappropriate to resume execution. If a return is required from this trap, use SUBS PC,R14_svc,#4. This will return to the instruction after the one causing the trap.

5.3.4 Abort

The Abort signal comes from an external Memory Management system, and 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. ARM checks for an Abort at the end of the first phase of each bus cycle. When successfully Aborted ARM will respond in one of three ways:

(i) if the abort occurred during an instruction prefetch (a *Prefetch Abort*), the prefetched instruction is marked as invalid; when it comes to execution, it is reinterpreted as below. (If the instruction is not executed, for example as a result of a branch being taken while it is in the pipeline, the abort will have no effect.)

(ii) if the abort occurred during a data access (a *Data Abort*), the action depends on the instruction type. Data transfer instructions (LDR, STR) are aborted as though the instruction had not executed. The LDM and STM instructions complete, and if writeback is set, the base is updated. If the instruction would normally have overwritten the base with data (i.e. 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.

(iii) if the abort occurred during an internal cycle it is ignored.

Then, in cases (i) and (ii):

(1) save R15 in R14_svc;

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

(3) force the PC to fetch the next instruction from address 0CH for Prefetch Abort, 10H for Data Abort.

To continue after a Prefetch Abort use SUBS PC,R14_svc,#4. This will attempt to re-execute the aborting instruction (which will only be effective if action has been taken to remove the cause of the original abort). A Data Abort requires any auto-indexing to be reversed before returning to re-execute the offending instruction, the return being done by SUBS PC,R14_svc,#8.

The abort mechanism allows a *demand paged virtual memory system* to be implemented when a suitable memory management unit (such as MEMC) is available. The processor is allowed to generate arbitrary addresses, and when the data at an address is unavailable the memory manager 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.

5.3.5 Software interrupt

The software interrupt is used for getting into supervisor mode, usually to request a particular supervisor function. ARM will:

(1) save R15 in R14_svc;

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

(3) force the PC to fetch the next instruction from address 08H.

To return from a SWI, use MOVS PC,R14_svc. This returns to the instruction following the SWI.

5.3.6 Undefined instruction trap

When ARM executes a Co-Processor instruction or an Undefined instruction, it offers it to any Co-Processors which may be present. If a Co-Processor can perform this instruction but is busy at that moment, ARM will wait until the Co-Processor is ready. If no Co-Processor can handle the instruction ARM will take the undefined instruction trap.

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

When the undefined instruction trap is taken ARM will:

(1) save R15 in R14_svc;

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

(3) force the PC to fetch the next instruction from address 04H.

To return from this trap (after performing a suitable emulation of the required function), use MOVS PC,R14_svc. This will return to the instruction following the undefined instruction.

5.3.7 Reset

When Reset goes HIGH ARM will:

(1) stop the currently executing instruction and start executing no-ops. When Reset goes LOW again it will:

(2) save R15 in R14_svc;

(3) force M0, M1 to supervisor mode and set the F and I bits in the PC word;

(4) force the PC to fetch the next instruction from address 0H.

5.3.8 Vector Summary

```
Address

0000000  Reset
0000004  Undefined instruction
0000008  Software interrupt
000000C  Abort (prefetch)
0000010  Abort (data)
0000014  Address exception
0000018  IRQ
000001C  FIQ
```

These are byte addresses, and will normally contain a branch instruction pointing to the relevant routine. The FIQ routine might reside at 000001CH onwards, and thereby avoid the need for (and execution time of) a branch instruction.

5.3.9 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) Address exception, 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. Address exception and data abort are mutually exclusive, since if an address is illegal the ARM will ignore the **ABORT** input. Undefined instruction and software interrupt are also mutually exclusive since they each correspond to particular (non-overlapping) decodings of the current instruction.

If an address exception or data abort occurs at the same time as a FIQ, and FIQs are enabled (ie the F flag in the PSR is clear), ARM will enter the address exception or data abort handler and then immediately proceed to the FIQ vector. A normal return from FIQ will cause the address exception or data abort handler to resume execution. Placing address exception and data abort at a higher priority than FIQ is necessary to ensure that the transfer error does not escape detection, but the time for this exception entry should be added to worst case FIQ latency calculations.

5.3.10 Interrupt Latencies

The worst case latency for FIQ, assuming that it is enabled, consists of the longest time the request can take to pass through the synchroniser (*Tsyncmax*), plus the time for the longest instruction to complete (*Tldm*, the longest instruction is load multiple registers), plus the time for address exception or data abort entry (*Texc*), plus the time for FIQ entry (*Tfiq*). At the end of this time ARM will be executing the instruction at 1CH.

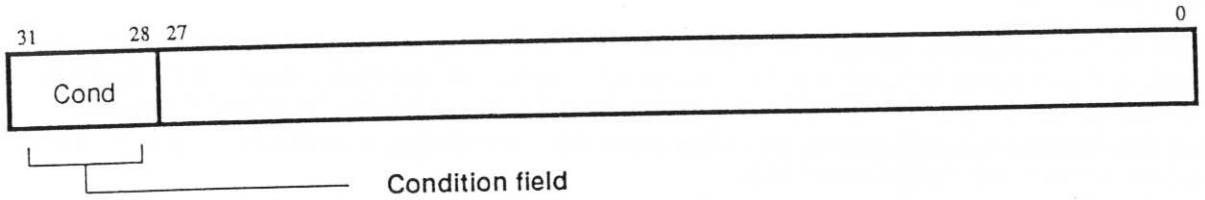
Tsyncmax is 2.5 processor cycles, *Tldm* is 18 cycles, *Texc* is 3 cycles, and *Tfiq* is 2 cycles. The total time is therefore 25.5 processor cycles, which is just over 2.5 microseconds in a system which uses a continuous 10 MHz processor clock. In a DRAM based system running at 4 and 8 MHz, for example using MEMC, this time becomes 4.5 microseconds, and if bus bandwidth is being used to support video or other DMA activity, the time will increase accordingly.

The maximum IRQ latency calculation is similar, but must allow for the fact that FIQ has higher priority and could delay entry into the IRQ handling routine for an arbitrary length of time.

The minimum latency for FIQ or IRQ consists of the shortest time the request can take through the synchroniser (*Tsyncmin*) plus *Tfiq*. This is 3.5 processor cycles.

6. Instruction Set

6.1 The condition field



Condition field

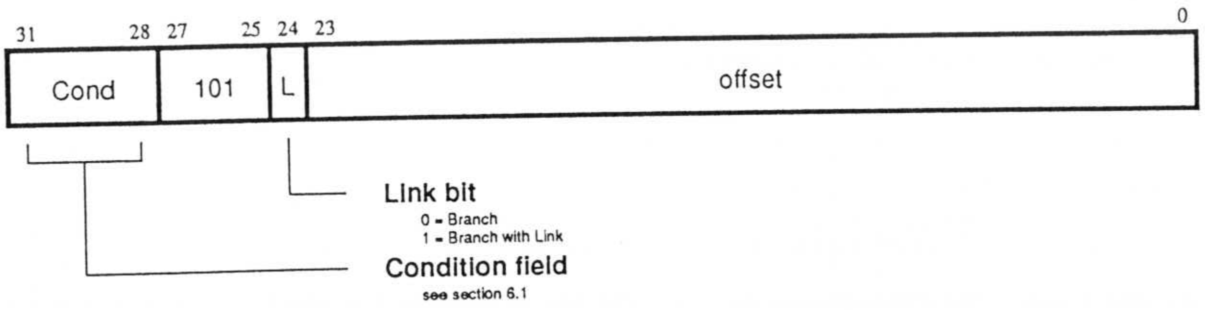
0000 - EQ - Z set (equal)
0001 - NE - Z clear (not equal)
0010 - CS - C set (unsigned higher or same)
0011 - CC - C clear (unsigned lower)
0100 - MI - N set (negative)
0101 - PL - N clear (positive or zero)
0110 - VS - V set (overflow)
0111 - VC - V clear (no overflow)
1000 - HI - C set and Z clear (unsigned higher)
1001 - LS - C clear or Z set (unsigned lower or same)
1010 - GE - N set and V set, or N clear and V clear (greater or equal)
1011 - LT - N set and V clear, or N clear and V set (less than)
1100 - GT - Z clear, and either N set and V set, or N clear and V clear (greater than)
1101 - LE - Z set, or N set and V clear, or N clear and V set (less than or equal)
1110 - AL - always
1111 - NV - never

All ARM 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 PSR at the end of the preceding instruction.

If the ALways condition is specified, the instruction will be executed irrespective of the flags, and likewise the NeVer condition will cause it not to be executed (it will be a no-op, ie take one cycle and have no effect on the processor state).

The other condition codes have meanings as detailed above, 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.

6.2 Branch and branch with link (B, BL)



The instruction is only executed if the condition specified in the condition field is true (see section 6.1).

All branches take a 24 bit offset. This is shifted left two bits and added to the PC, with any overflow being ignored. The branch can therefore reach any word aligned address within the address space. The branch offset must take account of the prefetch operation, which causes the PC to be 2 words ahead of the current instruction.

6.2.1 The link bit

Branch with Link writes the old PC and PSR into R14 of the current bank. The PC value written into the link register (R14) is adjusted to allow for the prefetch, and contains the address of the instruction following the branch and link instruction.

To return and restore the PSR use MOVS PC,R14 if the link register is still valid or LDM Rn!,{PC}^ if the link register has been saved onto a stack. To return without restoring the PSR 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.

6.2.2 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-char mnemonic as shown in section 6.1 (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.

6.2.3 Examples

```
here BAL here ; assembles to EAFFFFFE
              ; (note effect of PC offset)

 B there      ; ALways condition used as default

 CMP R1,#0    ; compare register 1 with zero
 BEQ fred     ; branch to fred if register 1 was zero
              ; otherwise continue to next instruction

 BL sub + ROM ; unconditionally call subroutine at computed address

 ADDS R1,#1   ; add 1 to register 1, setting PSR flags on the result
 BLCC sub     ; call subroutine if the C flag is clear, which will be
              ; the case unless R1 contained FFFFFFFFH
              ; otherwise continue to next instruction

 BLNV sub     ; NeVer call subroutine (this is a NO-OP)
```