

## 3.0 Programmer's Model

ARM60 supports a variety of operating configurations. Some are controlled by inputs and are known as the *hardware configurations*. Others may be controlled by software and these are known as *operating modes*.

### 3.1 Hardware Configuration

The ARM60 processor provides 4 hardware configurations which may be changed while the processor is running and which are detailed in *Chapter 4.0 Instruction Set*.

The **BIGEND** input sets whether the ARM60 treats words in memory as being stored in Big Endian or Little Endian format. Memory is viewed as a linear collection of bytes numbered upwards from zero. Bytes 0 to 3 hold the first stored word, bytes 4 to 7 the second and so on.

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

In the Big Endian scheme the most significant byte of a word is stored at the lowest numbered byte and the least significant byte is stored at the highest numbered byte. Byte 0 of the memory system should therefore be connected to data lines 31 through 24 (**D[31:24]**).

The **LATEABT** input sets the processor's behaviour when a data abort exception occurs. It only affects the behaviour of load/store register instructions and is discussed more fully in *Chapter 3.0 Programmer's Model* and *Chapter 4.0 Instruction Set*.

The other two inputs, **PROG32** and **DATA32** are used for backward compatibility with earlier ARM processors (see *13.0 Appendix - Backward Compatibility*) but should normally be set to 1. This configuration extends the address space to 32 bits, introduces major changes in the programmer's model as described below and provides support for running existing 26 bit programs in the 32 bit environment. This mode is recommended for compatibility with future ARM processors and all new code should be written to use only the 32 bit operating modes.

Because the original ARM instruction set has been modified to accommodate 32 bit operation there are certain additional restrictions which programmers must be aware of. These are indicated in the text by the words shall and shall not. Reference should also be made to the *ARM Application Notes "Rules for ARM Code Writers"* and *"Notes for ARM Code Writers"* available from your supplier.

### 3.2 Operating Mode Selection

ARM60 has a 32 bit data bus and a 32 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.

# ARM60 Data Sheet

---

ARM60 supports six modes of operation:

- (1) User mode (usr): the normal program execution state
- (2) FIQ mode (fiq): designed to support a data transfer or channel process
- (3) IRQ mode (irq): used for general purpose interrupt handling
- (4) Supervisor mode (svc): a protected mode for the operating system
- (5) Abort mode (abt): entered after a data or instruction prefetch abort
- (6) Undefined mode (und): entered when an undefined instruction is executed

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

## 3.3 Registers

The processor has a total of 37 registers made up of 31 general 32 bit registers and 6 status registers. At any one time 16 general registers (R0 to R15) and one or two status registers are visible to the programmer. The visible registers depend on the processor mode and the other registers (the *banked registers*) are switched in to support IRQ, FIQ, Supervisor, Abort and Undefined mode processing. The register bank organisation is shown in *Figure 3: Register Organisation*. The banked registers are shaded in the diagram.

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

R14 is used as the subroutine link register and receives a copy of R15 when a Branch and Link instruction is executed. It may be treated as a general purpose register at all other times. R14\_svc, R14\_irq, R14\_fiq, R14\_abt and R14\_und are used similarly to hold the return values of R15 when interrupts and exceptions arise, or when Branch and Link instructions are executed within interrupt or exception routines.

## General Registers and Program Counter Modes

User32	FIQ32	Supervisor32	Abort32	IRQ32	Undefined32
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8_fiq	R8	R8	R8	R8
R9	R9_fiq	R9	R9	R9	R9
R10	R10_fiq	R10	R10	R10	R10
R11	R11_fiq	R11	R11	R11	R11
R12	R12_fiq	R12	R12	R12	R12
R13	R13_fiq	R13_svc	R13_abt	R13_irq	R13_und
R14	R14_fiq	R14_svc	R14_abt	R14_irq	R14_und
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)

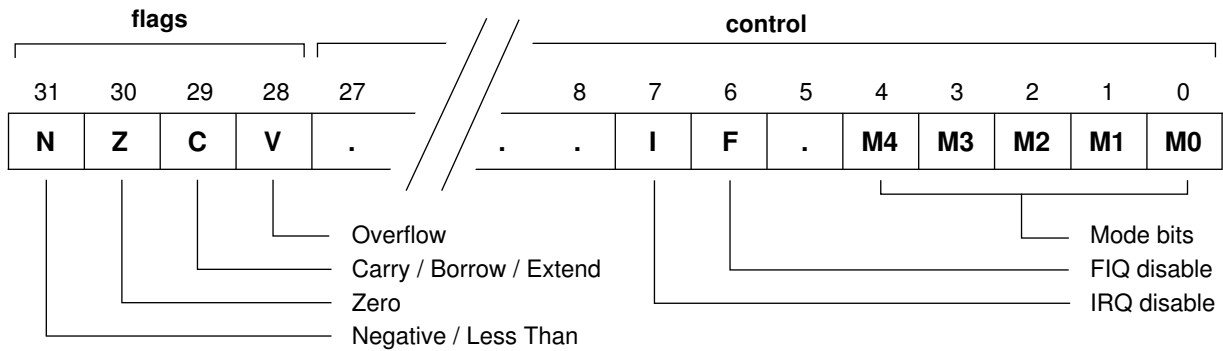
## Program Status Registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

**Figure 3: Register Organisation**

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

# ARM60 Data Sheet



**Figure 4: Format of the Program Status Registers (PSRs)**

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

The I and F bits are the *interrupt disable bits*. The I bit disables IRQ interrupts when it is set and the F bit disables FIQ interrupts when it is set. The M0, M1, M2, M3 and M4 bits (M[4:0]) are the *mode bits*, and these determine the mode in which the processor operates. The interpretation of the mode bits is shown in *Table 2: The Mode Bits*. Not all combinations of the mode bits define a valid processor mode. Only those explicitly described shall be used.

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

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

**Table 2: The Mode Bits**

## 3.4 Exceptions

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

ARM60 handles exceptions by making use of the banked registers to save state. The old PC and CPSR contents are copied into the appropriate R14 and SPSR and the PC and mode bits in the CPSR bits are forced to a value which depends on the exception. Interrupt disable flags are set where required to prevent otherwise unmanageable nestings of exceptions. In the case of a re-entrant interrupt handler, R14 and the SPSR should be saved onto a stack in main memory before re-enabling the interrupt; when transferring the SPSR register to and from a stack, it is important to transfer the whole 32 bit value, and not just the flag or control fields. When multiple exceptions arise simultaneously, a fixed priority determines the order in which they are handled. The priorities are listed later in this chapter.

### 3.4.1 FIQ

The FIQ (Fast Interrupt reQuest) exception is externally generated by taking the **nFIQ** input 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 (thus minimising the overhead of context switching). The FIQ exception may be disabled by setting the F flag in the CPSR (but note that this is not possible from User mode). If the F flag is clear, ARM60 checks for a LOW level on the output of the FIQ synchroniser at the end of each instruction.

When a FIQ is detected, ARM60 performs the following:

- (1) Saves the address of the next instruction to be executed plus 4 in R14\_fiq; saves CPSR in SPSR\_fiq
- (2) Forces M[4:0]=10001 (FIQ mode) and sets the F and I bits in the CPSR
- (3) Forces the PC to fetch the next instruction from address 0x1C

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

### 3.4.2 IRQ

The IRQ (Interrupt ReQuest) exception is a normal interrupt caused by a LOW level on the **nIRQ** input. It has a lower priority than FIQ, and is masked out when a FIQ sequence is entered. Its effect may be masked out at any time by setting the I bit in the CPSR (but note that this is not possible from User mode). If the I flag is clear, ARM60 checks for a LOW level on the output of the IRQ synchroniser at the end of each instruction. When an IRQ is detected, ARM60 performs the following:

- (1) Saves the address of the next instruction to be executed plus 4 in R14\_irq; saves CPSR in SPSR\_irq
- (2) Forces M[4:0]=10010 (IRQ mode) and sets the I bit in the CPSR
- (3) Forces the PC to fetch the next instruction from address 0x18

# ARM60 Data Sheet

---

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

## 3.4.3 Abort

An ABORT can be signalled by the external **ABORT** input. ABORT indicates that the current memory access cannot be completed. For instance, in a virtual memory system the data corresponding to the current address may have been moved out of memory onto a disc, and considerable processor activity may be required to recover the data before the access can be performed successfully. ARM60 checks for ABORT during memory access cycles. When successfully aborted ARM60 will respond in one of two ways:

- (1) If the abort occurred during an instruction prefetch (a *Prefetch Abort*), the prefetched instruction is marked as invalid but the abort exception does not occur immediately. If the instruction is not executed, for example as a result of a branch being taken while it is in the pipeline, no abort will occur. An abort will take place if the instruction reaches the head of the pipeline and is about to be executed.
- (2) If the abort occurred during a data access (a *Data Abort*), the action depends on the instruction type.
  - (a) Single data transfer instructions (LDR, STR) are aborted as though the instruction had not executed if the processor is configured for Early Abort. When configured for Late Abort, these instructions are able to write back modified base registers and the Abort handler must be aware of this.
  - (b) The swap instruction (SWP) is aborted as though it had not executed, though externally the read access may take place.
  - (c) Block data transfer instructions (LDM, STM) complete, and if write-back is set, the base is updated. If the instruction would normally have overwritten the base with data (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.

When either a prefetch or data abort occurs, ARM60 performs the following:

- (1) Saves the address of the aborted instruction plus 4 (for prefetch aborts) or 8 (for data aborts) in R14\_abt; saves CPSR in SPSR\_abt.
- (2) Forces M[4:0]=10111 (Abort mode) and sets the I bit in the CPSR.
- (3) Forces the PC to fetch the next instruction from either address 0x0C (prefetch abort) or address 0x10 (data abort).

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

The abort mechanism allows a *demand paged virtual memory system* to be implemented when suitable memory management software is available. The processor is allowed to generate arbitrary addresses, and when the data at an address is unavailable the MMU signals an abort. The processor traps into system software which must work out the cause of the abort, make the requested data available, and retry the aborted instruction. The application program needs no knowledge of the amount of memory available to it, nor is its state in any way affected by the abort.

## 3.4.4 Software interrupt

The software interrupt instruction (SWI) is used for getting into Supervisor mode, usually to request a particular supervisor function. When a SWI is executed, ARM60 performs the following:

- (1) Saves the address of the SWI instruction plus 4 in R14\_svc; saves CPSR in SPSR\_svc
- (2) Forces M[4:0]=10011 (Supervisor mode) and sets the I bit in the CPSR
- (3) Forces the PC to fetch the next instruction from address 0x08

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

## 3.4.5 Undefined instruction trap

When the ARM60 comes across an instruction which it cannot handle (see *Chapter 4.0 Instruction Set*), it offers it to any coprocessors which may be present. If a coprocessor can perform this instruction but is busy at that time, ARM60 will wait until the coprocessor is ready or until an interrupt occurs. If no coprocessor can handle the instruction then ARM60 will take the undefined instruction trap.

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

When ARM60 takes the undefined instruction trap it performs the following:

- (1) Saves the address of the Undefined or coprocessor instruction plus 4 in R14\_und; saves CPSR in SPSR\_und.
- (2) Forces M[4:0]=11011 (Undefined mode) and sets the I bit in the CPSR
- (3) Forces the PC to fetch the next instruction from address 0x04

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

# ARM60 Data Sheet

---

## 3.4.6 Vector Summary

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

**Table 3: Vector Summary**

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

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

The reserved entry is for an Address Exception vector which is only operative when the processor is configured for a 26 bit program space. See *13.0 Appendix - Backward Compatibility*

## 3.4.7 Exception Priorities

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

- (1) Reset (highest priority)
- (2) Data abort
- (3) FIQ
- (4) IRQ
- (5) Prefetch abort
- (6) Undefined Instruction, Software interrupt (lowest priority)

Note that not all exceptions can occur at once. Undefined instruction and software interrupt are mutually exclusive since they each correspond to particular (non-overlapping) decodings of the current instruction.

If a data abort occurs at the same time as a FIQ, and FIQs are enabled (i.e. the F flag in the CPSR is clear), ARM60 will enter the data abort handler and then immediately proceed to the FIQ vector. A normal return from FIQ will cause the data abort handler to resume execution. Placing data abort at a higher priority than FIQ is necessary to ensure that the transfer error does not escape detection; the time for this exception entry should be added to worst case FIQ latency calculations.



## 3.4.8 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 ( $T_{syncmax}$ ), plus the time for the longest instruction to complete ( $T_{ldm}$ , the longest instruction is an LDM which loads all the registers including the PC), plus the time for the data abort entry ( $T_{exc}$ ), plus the time for FIQ entry ( $T_{fiq}$ ). At the end of this time ARM60 will be executing the instruction at 0x1C.

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

## 3.5 Reset

When the **nRESET** signal goes LOW, ARM60 abandons the executing instruction and then continues to fetch instructions from incrementing word addresses.

When **nRESET** goes HIGH again, ARM60 does the following:

- (1) Overwrites R14\_svc and SPSR\_svc by copying the current values of the PC and CPSR into them. The value of the saved PC and CPSR is not defined.
- (2) Forces M[4:0]=10011 (Supervisor mode) and sets the I and F bits in the CPSR.
- (3) Forces the PC to fetch the next instruction from address 0x00

## ARM60 Data Sheet

---