3 Programmer's Model

This chapter describes the two operating states of the ARM7TDMI-S.

3.1 Processor Operating States

From the programmer’s point of view, the ARM7TDMI-S can be in one of two states:

ARM state	which executes 32-bit, word-aligned ARM instructions.
THUMB state	which operates with 16-bit, halfword-aligned THUMB instructions. In this state, the PC uses bit 1 to select between alternate halfwords.

Note *Transition between these two states does not affect the processor mode or the contents of the registers.*

3.2 Switching State

Entering THUMB state

Entry into THUMB state can be achieved by executing a BX instruction with the state bit (bit 0) set in the operand register.

Transition to THUMB state will also occur automatically on return from an exception (IRQ, FIQ, UNDEF, ABORT, SWI etc.), if the exception was entered with the processor in THUMB state.

Entering ARM state

Entry into ARM state happens:

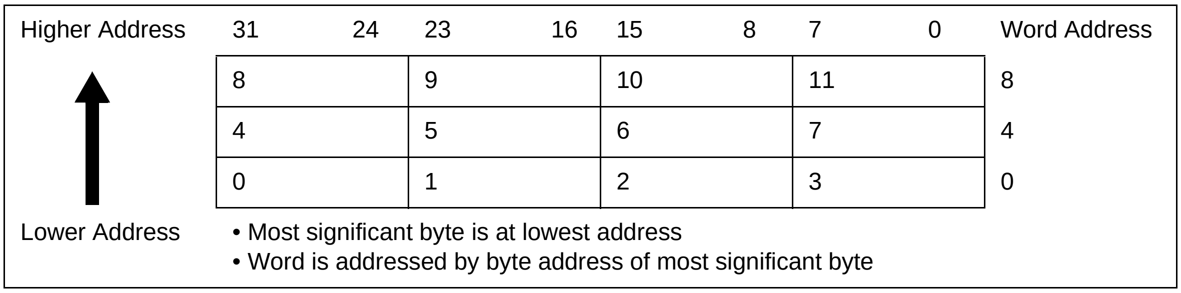
1. On execution of the BX instruction with the state bit clear in the operand register.
2. On the processor taking an exception (IRQ, FIQ, RESET, UNDEF, ABORT, SWI etc.).
   In this case, the PC is placed in the exception mode's link register, and execution commences at the exception's vector address.

3.3 Memory Formats

ARM7TDMI-S views memory 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. ARM7TDMI-S can treat words in memory as being stored either in *Big-endian* or *Little-endian* format.

3.3.1 Big-endian format

In big-endian format, the most significant byte of a word is stored at the lowest numbered byte and the least significant byte at the highest numbered byte. Byte 0 of the memory system is therefore connected to data lines 31 through 24.



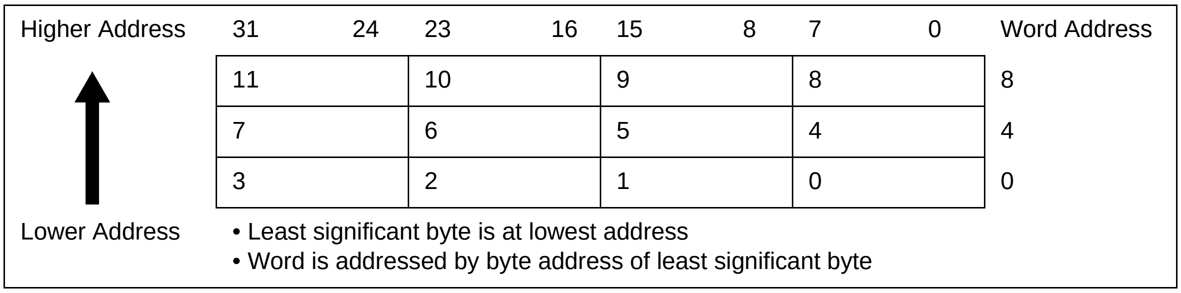
31 24	23 16	15 8	7 0	Word Address
8	9	10	11	8
4	5	6	7	4
0	1	2	3	0

- Most significant byte is at lowest address
- Word is addressed by byte address of most significant byte

Figure 3-1: Big-endian addresses of bytes within words

3.3.2 Little-endian format

In little-endian format, the lowest numbered byte in a word is considered the word’s least significant byte, and the highest numbered byte the most significant. Byte 0 of the memory system is therefore connected to data lines 7 through 0.



31 24	23 16	15 8	7 0	Word Address
11	10	9	8	8
7	6	5	4	4
3	2	1	0	0

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

Figure 3-2: Little-endian addresses of bytes within words

3.4 Instruction Length

Instructions are either 32 bits long (in ARM state) or 16 bits long (in THUMB state).

3.5 Data Types

ARM7TDMI-S supports byte (8-bit), halfword (16-bit) and word (32-bit) data types. Words must be aligned to four-byte boundaries and half words to two-byte boundaries.

3.6 Operating Modes

ARM7TDMI-S supports seven modes of operation:

User (usr):	The normal ARM program execution state
FIQ (fiq):	Designed to support a data transfer or channel process
IRQ (irq):	Used for general-purpose interrupt handling
Supervisor (svc):	Protected mode for the operating system
Abort mode (abt):	Entered after a data or instruction prefetch abort
System (sys):	A privileged user mode for the operating system
Undefined (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 non-user modes—known as *privileged modes*—are entered in order to service interrupts or exceptions, or to access protected resources.

3.7 Registers

ARM7TDMI-S has a total of 37 registers—31 general-purpose 32-bit registers and six status registers—but these cannot all be seen at once. The processor state and operating mode dictate which registers are available to the programmer.

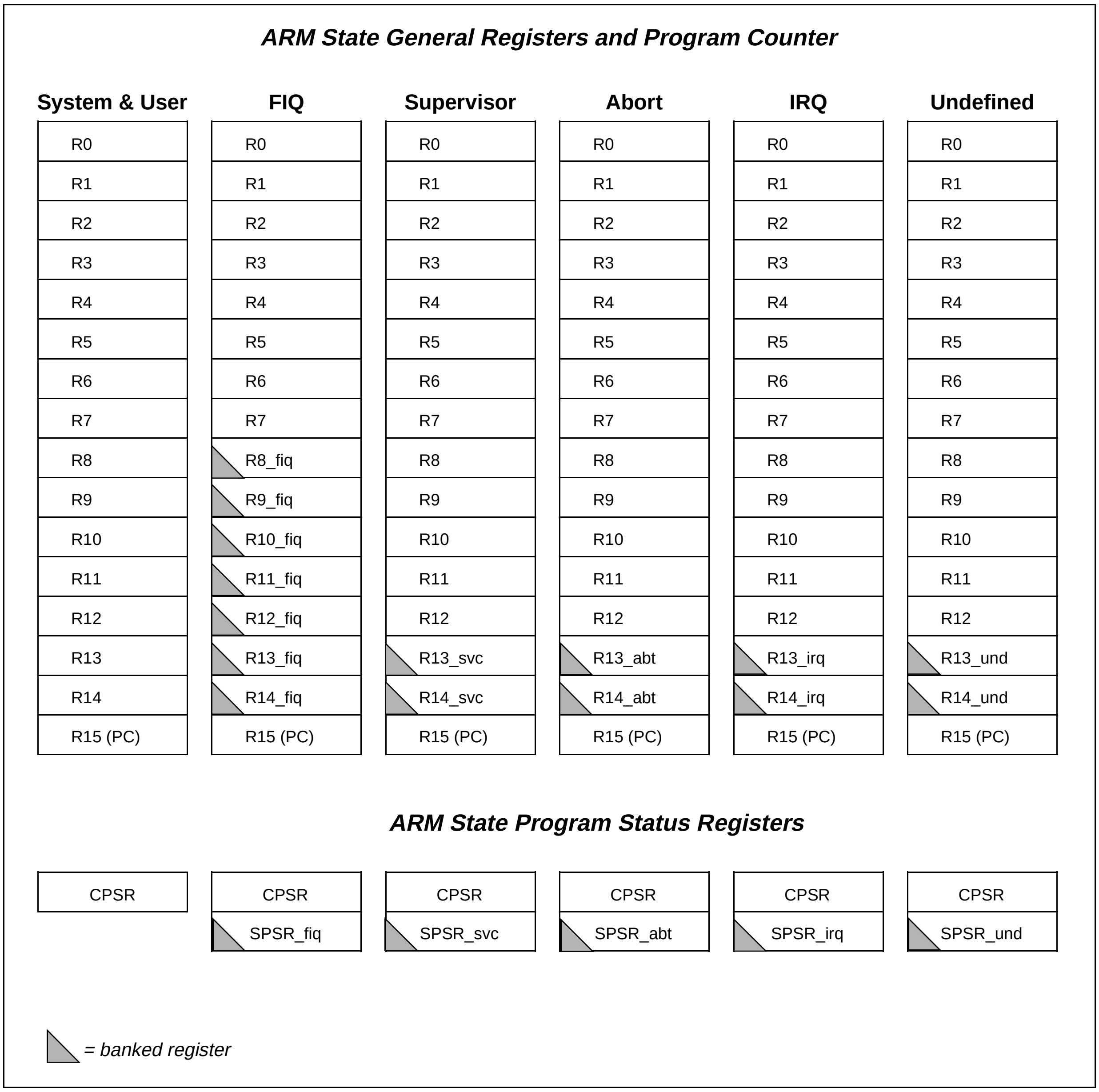
3.7.1 The ARM state register set

In ARM state, 16 general registers and one or two status registers are visible at any one time. In privileged (non-User) modes, mode-specific banked registers are switched in. ***Figure 3-3: Register organization in ARM state*** on page 3-9 shows which registers are available in each mode: the banked registers are marked with a shaded triangle.

The ARM state register set contains 16 directly accessible registers: R0 to R15. All of these except R15 are general-purpose, and may be used to hold either data or address values. In addition to these, there is a seventeenth register used to store status information

Register 14	is used as the subroutine link register. This receives a copy of R15 when a Branch and Link (BL) instruction is executed. At all other times it may be treated as a general-purpose register. The corresponding banked registers R14_svc, R14_irq, R14_fiq, R14_abt and R14_und are similarly used 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.
Register 15	holds the Program Counter (PC). In ARM state, bits [1:0] of R15 are zero and bits [31:2] contain the PC. In THUMB state, bit [0] is zero and bits [31:1] contain the PC.
Register 16	is the CPSR (Current Program Status Register). This contains condition code flags and the current mode bits.

FIQ mode has seven banked registers mapped to R8q–14 (R8_fiq–R14_fiq). In ARM state, many FIQ handlers do not need to save any registers. User, IRQ, Supervisor, Abort and Undefined each have two banked registers mapped to R13 and R14, allowing each of these modes to have a private stack pointer and link registers.



ARM State General Registers and Program Counter

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
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)

ARM State Program Status Registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	◣ SPSR_fiq	◣ SPSR_svc	◣ SPSR_abt	◣ SPSR_irq	◣ SPSR_und

◣ *= banked register*

Figure 3-3: Register organization in ARM state

3.7.2 The THUMB state register set

The THUMB state register set is a subset of the ARM state set. The programmer has direct access to eight general registers, R0–R7, as well as the Program Counter (PC), a stack pointer register (SP), a link register (LR), and the CPSR. There are banked Stack Pointers, Link Registers and Saved Process Status Registers (SPSRs) for each privileged mode. This is shown in ***Figure 3-4: Register organization in THUMB state*** on page 3-10.

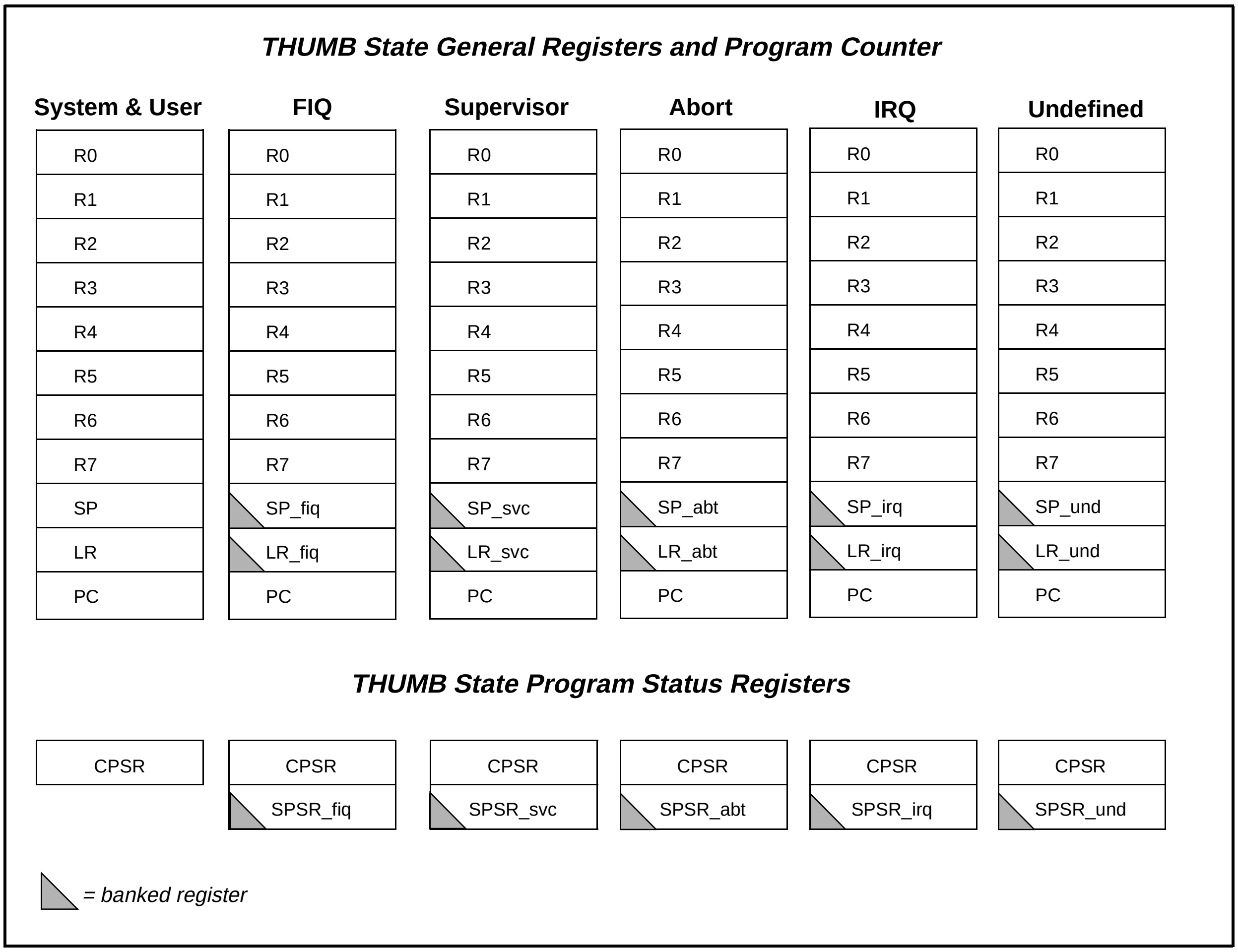


Figure 3-4: Register organization in THUMB state

3.7.3 The relationship between ARM and THUMB state registers

The THUMB state registers relate to the ARM state registers in the following way:

- THUMB state R0–R7 and ARM state R0–R7 are identical
- THUMB state CPSR and SPSRs and ARM state CPSR and SPSRs are identical
- THUMB state SP maps onto ARM state R13
- THUMB state LR maps onto ARM state R14
- The THUMB state Program Counter maps onto the ARM state Program Counter (R15)

This relationship is shown in ***Figure 3-5: Mapping of THUMB state registers onto ARM state registers*** on page 3-11.

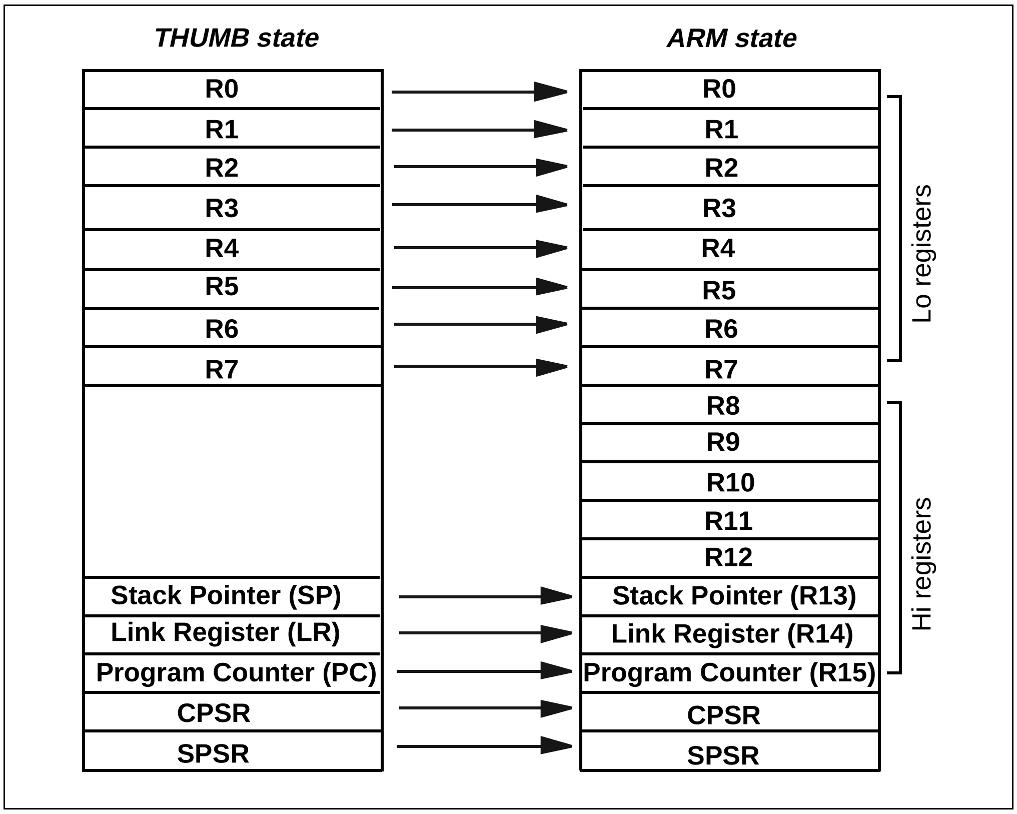


Figure 3-5: Mapping of THUMB state registers onto ARM state registers

3.7.4 Accessing Hi registers in THUMB state

In THUMB state, registers R8–R15 (the *Hi registers*) are not part of the standard register set. However, the assembly language programmer has limited access to them, and can use them for fast temporary storage.

A value may be transferred from a register in the range R0–R7 (a *Lo register*) to a Hi register, and from a Hi register to a Lo register, using special variants of the `MOV` instruction. Hi register values can also be compared against or added to Lo register values with the `CMP` and `ADD` instructions. See ***5.5 Format 5: Hi register operations/ branch exchange*** on page 5-13.

3.8 The Program Status Registers

The ARM7TDMI-S contains a Current Program Status Register (CPSR), plus five Saved Program Status Registers (SPSRs) for use by exception handlers. These registers

- hold information about the most recently performed ALU operation
- control the enabling and disabling of interrupts
- set the processor operating mode

The arrangement of bits is shown in ***Figure 3-6: Program status register format***.

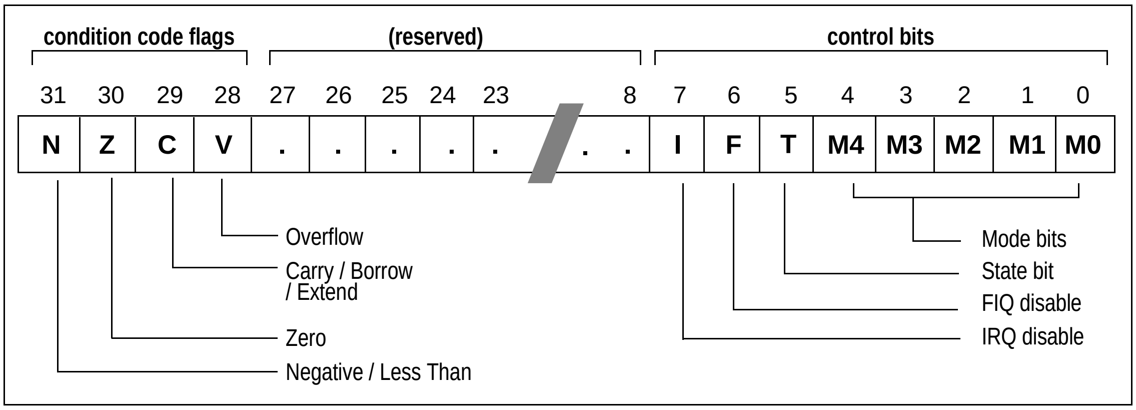


Figure 3-6: Program status register format

3.8.1 The condition code flags

The N, Z, C and V bits are the condition code flags. These may be changed as a result of arithmetic and logical operations, and may be tested to determine whether an instruction should be executed.

In ARM state, all instructions may be executed conditionally: see ***4.2 The Condition Field*** on page 4-5 for details.

In THUMB state, only the Branch instruction is capable of conditional execution: see ***5.17 Format 17: software interrupt*** on page 5-38.

3.8.2 The control bits

The bottom 8 bits of a PSR (incorporating I, F, T and M[4:0]) are known collectively as the control bits. These will change when an exception arises. If the processor is operating in a privileged mode, they can also be manipulated by software.

The T bit	This reflects the operating state. When this bit is set, the processor is executing in THUMB state, otherwise it is executing in ARM state. This is reflected on the **TBIT** external signal.
	Note that the software must never change the state of the T bit in the CPSR. If this happens, the processor will enter an unpredictable state.
Interrupt disable bits	The I and F bits are the interrupt disable bits. When set, these disable the IRQ and FIQ interrupts respectively.

The mode bits

The M4, M3, M2, M1 and M0 bits (M[4:0]) are the mode bits. These determine the processor’s operating mode, as shown in ***Table 3-1: PSR mode bit values***. Not all combinations of the mode bits define a valid processor mode. Only use the bit combinations described in the table. An illegal value programmed into M[4:0] will cause the processor to enter an unrecoverable state. If this occurs, apply reset.

M[4:0]	Mode	Visible THUMB state registers	Visible ARM state registers
10000	User	R7–R0, LR, SP PC, CPSR	R14–R0, PC, CPSR
10001	FIQ	R7–R0, LR_fiq, SP_fiq PC, CPSR, SPSR_fiq	R7–R0, R14_fiq–R8_fiq, PC, CPSR, SPSR_fiq
10010	IRQ	R7–R0, LR_irq, SP_irq PC, CPSR, SPSR_irq	R12–R0, R14_irq–R13_irq, PC, CPSR, SPSR_irq
10011	Supervisor	R7–R0, LR_svc, SP_svc, PC, CPSR, SPSR_svc	R12–R0, R14_svc–R13_svc, PC, CPSR, SPSR_svc
10111	Abort	R7–R0, LR_abt, SP_abt, PC, CPSR, SPSR_abt	R12–R0, R14_abt–R13_abt, PC, CPSR, SPSR_abt
11011	Undefined	R7–R0 LR_und, SP_und, PC, CPSR, SPSR_und	R12–R0, R14_und–R13_und, PC, CPSR
11111	System	R7–R0, LR, SP PC, CPSR	R14–R0, PC, CPSR

Table 3-1: PSR mode bit values

Reserved bits

The remaining bits in the PSRs are reserved. When changing a PSR’s flag or control bits, you must ensure that these unused bits are not altered. Also, your program should not rely on them containing specific values, since in future processors they may read as one or zero.

3.9 Exceptions

Exceptions arise whenever the normal flow of a program has to be halted temporarily, for example to service an interrupt from a peripheral. Before an exception can be handled, the current processor state must be preserved so that the original program can resume when the handler routine has finished.

It is possible for several exceptions to arise at the same time. If this happens, they are dealt with in a fixed order—see ***3.9.10 Exception priorities*** on page 3-17.

3.9.1 Action on entering an exception

When handling an exception, the ARM7TDMI-S:

1. Preserves the address of the next instruction in the appropriate Link Register. If the exception has been entered from ARM state, the address of the next instruction is copied into the Link Register (that is, current PC + 4 or PC + 8 depending on the exception. Refer to ***Table 3-2: Exception entry/exit*** on page 3-15 for details). If the exception has been entered from THUMB state, the value written into the Link Register is the current PC offset by a value such that the program resumes from the correct place on return from the exception. This means that the exception handler need not determine which state the exception was entered from. For example, in the case of SWI, `MOVS PC, R14_svc` will always return to the next instruction regardless of whether the SWI was executed in ARM or THUMB state.
2. Copies the CPSR into the appropriate SPSR
3. Forces the CPSR mode bits to a value which depends on the exception
4. Forces the PC to fetch the next instruction from the relevant exception vector

It may also set the interrupt disable flags to prevent otherwise unmanageable nestings of exceptions.

If the processor is in THUMB state when an exception occurs, it will automatically switch into ARM state when the PC is loaded with the exception vector address.

3.9.2 Action on leaving an exception

On completion, the exception handler:

1. Moves the Link Register, minus an offset where appropriate, to the PC. (The offset will vary depending on the type of exception.)
2. Copies the SPSR back to the CPSR.
3. Clears the interrupt disable flags, if they were set on entry.

Note *An explicit switch back to THUMB state is never needed, because restoring the CPSR from the SPSR automatically sets the T bit to the value it held immediately prior to the exception.*

3.9.3 Exception entry/exit summary

Table 3-2: Exception entry/exit summarizes the PC value preserved in the relevant R14 on exception entry, and the recommended instruction for exiting the exception handler.

Exception or Entry	Return Instruction	Previous State ARM R14_x	Previous State THUMB R14_x	Notes
BL	MOV PC, R14	PC + 4	PC + 2	1
SWI	MOVS PC, R14_svc	PC + 4	PC + 2	1
UDEF	MOVS PC, R14_und	PC + 4	PC + 2	1
FIQ	SUBS PC, R14_fiq, #4	PC + 4	PC + 4	2
IRQ	SUBS PC, R14_irq, #4	PC + 4	PC + 4	2
PABT	SUBS PC, R14_abt, #4	PC + 4	PC + 4	1
DABT	SUBS PC, R14_abt, #8	PC + 8	PC + 8	3
RESET	NA	-	-	4

Table 3-2: Exception entry/exit

Notes

1 Where PC is the address of the BL/SWI/Undefined Instruction fetch that had the prefetch abort.

2 Where PC is the address of the instruction that did not get executed since the FIQ or IRQ took priority.

3 Where PC is the address of the Load or Store instruction that generated the data abort.

4 The value saved in R14_svc upon reset is unpredictable.

3.9.4 FIQ

The *FIQ (Fast Interrupt Request)* exception is designed to support a data transfer or channel process, and in ARM state has sufficient private registers to remove the need for register saving (thus minimizing the overhead of context switching).

FIQ is externally generated by taking the **nFIQ** input LOW.

Irrespective of whether the exception was entered from ARM or THUMB state, a FIQ handler should leave the interrupt by executing:

```
SUBS PC,R14_fiq,#4
```

FIQ may be disabled by setting the CPSR's F flag (but note that this is not possible from User mode). If the F flag is clear, ARM7TDMI-S checks for a LOW level on the output of the FIQ synchronizer at the end of each instruction.

3.9.5 IRQ

The *IRQ (Interrupt Request)* exception is a normal interrupt caused by a LOW level on the **nIRQ** input. IRQ has a lower priority than FIQ and is masked out when a FIQ sequence is entered. It may be disabled at any time by setting the I bit in the CPSR, though this can only be done from a privileged (non-User) mode.

Irrespective of whether the exception was entered from ARM or Thumb state, an IRQ handler should return from the interrupt by executing

```
SUBS PC,R14_irq,#4
```

3.9.6 Abort

An abort indicates that the current memory access cannot be completed. It can be signalled by the external **ABORT** input. ARM7TDMI-S checks for the abort exception during memory access cycles.

There are two types of abort:

Prefetch abort	occurs during an instruction prefetch.
Data abort	occurs during a data access.

If a prefetch abort occurs, the prefetched instruction is marked as invalid, but the exception will not be taken until the instruction reaches the head of the pipeline. If the instruction is not executed—for example because a branch occurs while it is in the pipeline—the abort does not take place.

If a data abort occurs, the action taken depends on the instruction type:

1. Single data transfer instructions (LDR, STR) write back modified base registers: the Abort handler must be aware of this.
2. The swap instruction (SWP) is aborted as though it had not been executed.
3. Block data transfer instructions (LDM, STM) complete. If write-back is set, the base is updated. If the instruction would have overwritten the base with data (ie. it has the base in the transfer list), the overwriting is prevented. All register overwriting is prevented after an abort is indicated, which means in particular that R15 (always the last register to be transferred) is preserved in an aborted LDM instruction.

The abort mechanism allows the implementation of a demand paged virtual memory system. In such a system the processor is allowed to generate arbitrary addresses. When the data at an address is unavailable, the Memory Management Unit (MMU) signals an abort. The abort handler must then 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.

After fixing the reason for the abort, the handler should execute the following irrespective of the state (ARM or Thumb):

`SUBS PC,R14_abt,#4` for a prefetch abort, or

`SUBS PC,R14_abt,#8` for a data abort

This restores both the PC and the CPSR, and retries the aborted instruction.

3.9.7 Software interrupt

The *software interrupt instruction (SWI)* is used for entering Supervisor mode, usually to request a particular supervisor function. A SWI handler should return by executing the following irrespective of the state (ARM or THUMB):

```
MOV PC, R14_svc
```

This restores the PC and CPSR, and returns to the instruction following the SWI.

3.9.8 Undefined instruction

When ARM7TDMI-S comes across an instruction which it cannot handle, it takes the undefined instruction trap. This mechanism may be used to extend either the THUMB or ARM instruction set by software emulation.

After emulating the failed instruction, the trap handler should execute the following irrespective of the state (ARM or Thumb):

```
MOVS PC,R14_und
```

This restores the CPSR and returns to the instruction following the undefined instruction.

The ARM7TDMI-S is fully compliant with the ARM Instruction Set Architecture version v4T in the trapping of all instruction bit patterns which are classified as undefined.

3.9.9 Exception vectors

The following table shows the exception vector addresses.

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*	*Reserved*
0x00000018	IRQ	IRQ
0x0000001C	FIQ	FIQ

Table 3-3: Exception vectors

3.9.10 Exception priorities

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

Highest priority:

1	Reset
2	Data abort
3	FIQ
4	IRQ
5	Prefetch abort

Lowest priority:

6	Undefined Instruction, Software interrupt.

Not all exceptions can occur at once:

Undefined Instruction and Software Interrupt are mutually exclusive, since they each correspond to particular (non-overlapping) decodings of the current instruction.

If a data abort occurs at the same time as a FIQ, and FIQs are enabled (ie. the CPSR's F flag is clear), ARM7TDMI-S enters the data abort handler and then immediately proceeds 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.

3.10 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 synchronizer (*Tsyncmax*), plus the time for the longest instruction to complete (*Tldm*, the longest instruction is an LDM which loads all the registers including the PC), plus the time for the data abort entry (*Texc*), plus the time for FIQ entry (*Tfiq*). At the end of this time ARM7TDMI-S will be executing the instruction at 0x1C.

Tsyncmax is 3 processor cycles, *Tldm* is 20 cycles, *Texc* is 3 cycles, and *Tfiq* 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 synchronizer (*Tsyncmin*) plus *Tfiq*. This is 4 processor cycles.

3.11 Reset

When the **nRESET** signal goes LOW, ARM7TDMI-S abandons the executing instruction.

When **nRESET** goes HIGH again, ARM7TDMI-S:

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 SPSR is not defined.
2. Forces M[4:0] to 10011 (Supervisor mode), sets the I and F bits in the CPSR, and clears the CPSR's T bit.
3. Forces the PC to fetch the next instruction from address 0x00.
4. Execution resumes in ARM state.