An Introduction to the ARM 7 Architecture

Trevor Martin CEng, MIEE
Technical Director

This article gives an overview of the ARM 7 architecture and a description of its major features for a developer new to the device. Future articles will examine other aspects of the ARM architecture.

Basic Characteristics

The principle feature of the ARM 7 microcontroller is that it is a register based load-and-store architecture with a number of operating modes. While the ARM7 is a 32 bit microcontroller, it is also capable of running a 16-bit instruction set, known as "THUMB". This helps it achieve a greater code density and enhanced power saving. While all of the register-to-register data processing instructions are single-cycle, other instructions such as data transfer instructions, are multi-cycle. To increase the performance of these instructions, the ARM 7 has a three-stage pipeline. Due to the inherent simplicity of the design and low gate count, ARM 7 is the industry leader in low-power processing on a watts per MIP basis. Finally, to assist the developer, the ARM core has a built-in JTAG debug port and on-chip "embedded ICE" that allows programs to be downloaded and fully debugged in-system.

In order to keep the ARM 7 both simple and cost-effective, the code and data regions are accessed via a single data bus. Thus while the ARM 7 is capable of single-cycle execution of all data processing instructions, data transfer instructions may take several cycles since they will require at least two accesses onto the bus (one for the instruction one for the data). In order to improve performance, a three stage pipeline is used that allows multiple instructions to be processed simultaneously.

The pipeline has three stages; FETCH, DECODE and EXECUTE. The hardware of each stage is designed to be independent so up to three instructions can be processed simultaneously. The pipeline is most effective in speeding up sequential code. However a branch instruction will cause the pipeline to be flushed marring its performance. As we shall see later the ARM 7 designers had some clever ideas to solve this problem.

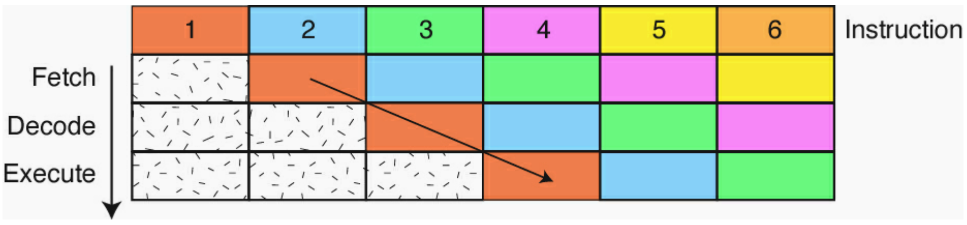


Fig 1 ARM 3-Stage Pipeline

ARM7 Programming Model

The programmer's model of the ARM 7 consists of 15 user registers, as shown in Fig. 3, with R15 being used as the Program Counter (PC). Since the ARM 7 is a load-and-store architecture, an user program must load data from memory into the CPU registers, process this data and then store the result back into memory. Unlike other processors no memory to memory instructions are available.

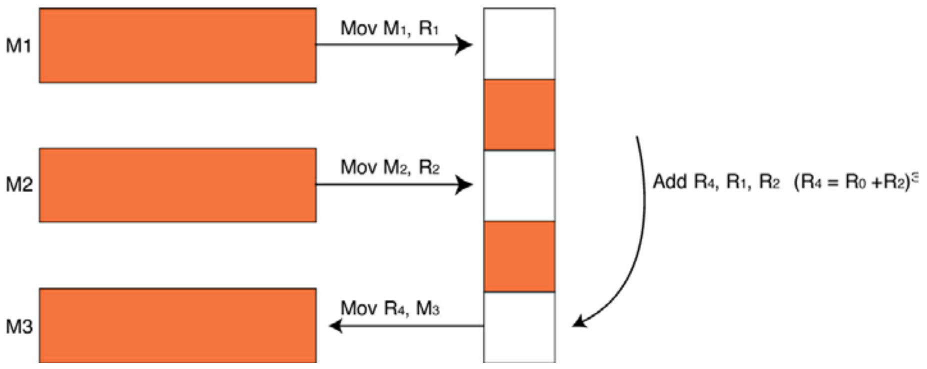


Fig 2 Load And Store Architecture

As stated above R15 is the Program Counter. R13 and R14 also have special functions; R13 is used as the stack pointer, though this has only been defined as a programming convention. Unusually the ARM instruction set does not have PUSH and POP instructions so stack handling is done via a set of instructions that allow loading and storing of multiple registers in a single operation. Thus it is possible to PUSH or POP the entire register set onto the stack in a single instruction. R14 has special significance and is called the "link register". When a call is made to a procedure, the return address is automatically placed into R14, rather than onto a stack, as might be expected. A return can then be implemented by moving the contents of R14 into R15, the PC. For multiple calling trees, the contents of R14 (the link register) must be placed onto the stack.

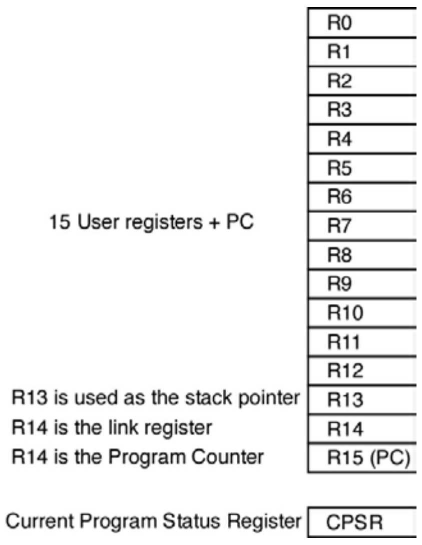


Fig 3 User Mode Register Model

In addition to the 16 CPU registers, there is a current program status register (CPSR). This contains a set of condition code flags in the upper four bits that record the result of a previous instruction, as shown in Fig 4. In addition to the condition code flags, the CPSR contains a number of user-configurable bits that can be used to change the processor mode, enter Thumb processing and enable/disable interrupts.

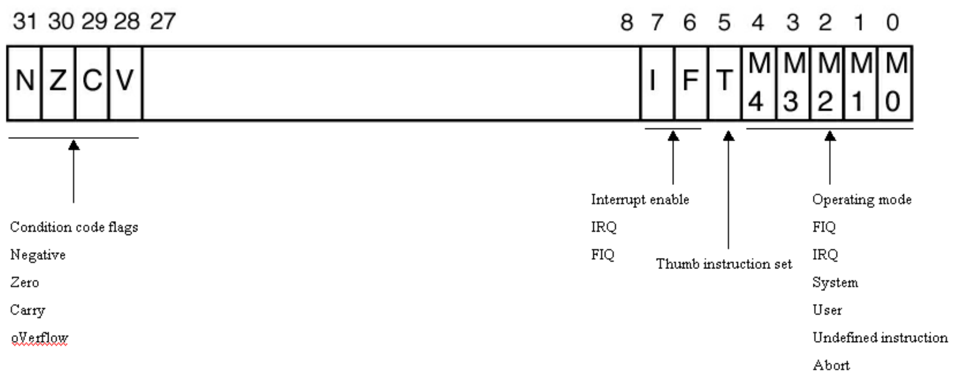


Fig 4 Current Program Status Register and Flags

Exception And Interrupt Modes

The ARM 7 architecture has a total of six different operating modes, as shown below. These modes are protected or exception modes which have associated interrupt sources and their own register sets.

User: This mode is used to run the application code. Once in user mode the CPSR cannot be written to and modes can only be changed when an exception is generated.

FIQ: (Fast Interrupt reQuest) This supports high speed interrupt handling. Generally it is used for a single critical interrupt source in a system

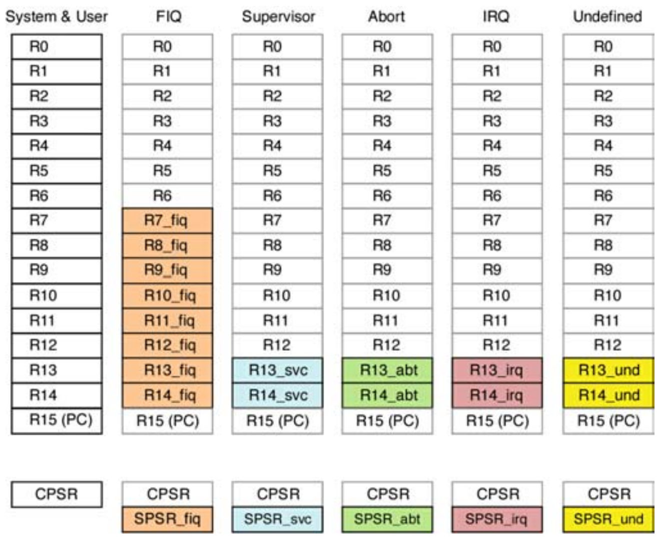
IRQ: (Interrupt ReQuest) This supports all other interrupt sources in a system

Supervisor: A “protected” mode for running system level code to access hardware or run OS calls. The ARM 7 enters this mode after reset.

Abort: If an instruction or data is fetched from an invalid memory region, an abort exception will be generated

Undefined Instruction: If a FETCHED opcode is not an ARM instruction, an undefined instruction exception will be generated.

The User registers R0-R7 are common to all operating modes. However FIQ mode has its own R8 –R14 that replace the user registers when FIQ is entered. Similarly, each of the other modes have their own R13 and R14 so that each operating mode has its own unique Stack pointer and Link register. The CPSR is also common to all modes. However in each of the exception modes, an additional register - the saved program status register (SPSR), is added. When the processor changes the current value of the CPSR stored in the SPSR, this can be restored on exiting the exception mode.



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_fiq	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)
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

Fig 5 Full Register Set For ARM 7

Entry to the Exception modes is through the interrupt vector table. Exceptions in the ARM processor can be split into three distinct types.

(i) Exceptions caused by executing an instruction, these include software interrupts, undefined instruction exceptions and memory abort exceptions

(ii) Exceptions caused as a side effect of an instruction such as a abort caused by trying to fetch data from an invalid memory region.

(iii) Exceptions unrelated to instruction execution, this includes reset, FIQ and IRQ interrupts.

In each case entry into the exception mode uses the same mechanism. On generation of the exception, the processor switches to the privileged mode, the current value of the PC+4 is saved into the Link register (R14) of the privileged mode and the current value of CPSR is saved into the privileged mode's SPSR. The IRQ interrupts are also disabled and if the FIQ mode is entered, the FIQ interrupts are also disabled Finally the Program Counter is forced to the exception vector address and processing of the exception can start. Usually the first action of the exception routine will be to push some or all of the user registers onto the stack.

Exeption type	Mode	Meaning
Reset	Supervisor	0x00000000
Undefined instruction	Undefined	0x00000004
Software interrupt (SWI)	Supervisor	0x00000008
Prefetch Abort (instruction fetch memory abort)	Abort	0x0000000C
Data Abort (data access memory abort)	Abort	0x00000010
IRQ (interrupt)	IRQ	0x00000018
FIQ (fast interrupt)	FIQ	0x0000001C

Fig 6 ARM 7 Vector Table

A couple of things are worth noting on the vector table. Firstly, there is a missing vector at 0x000000014. This was used on an earlier ARM architecture and is left empty on ARM 7 to allow backward compatibility. Secondly, the FIQ interrupt is at the highest address so the FIQ routines could start from this address, removing the need for a jump instruction to reach the routine. It helps make entry into the FIQ routine as fast as possible.

Once processing of the exception has finished, the processor can leave the privileged mode and return to the user mode. Firstly the contents of any registers previously saved onto the stack must be restored. Next the CSPR must be restored from the SPSR and finally the Program Counter is restored by moving the contents of the link register to R15, (i.e. the Program Counter). The interrupted program flow can then restart.

Data Types

The ARM instruction set supports six data types namely 8 bit signed and unsigned, 16 bit signed and unsigned plus 32 bit signed and unsigned. The ARM processor instruction set has been designed to support these data types in Little or Big-endian formats. However most ARM silicon implementations use the Little-endian format.

ARM instructions typically have a three-operand format, as shown below

```
ADD R1, R2, R3        ;        R1 = R2+R3
```

ARM7 Program Flow Control

In all processors there is a small group of instructions that are conditionally executed depending on a group of processor flags. These are branch instructions such as branch not equal. Within the ARM instruction set, all instructions are conditionally executable.

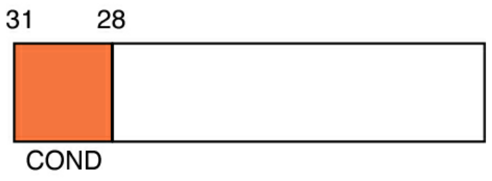
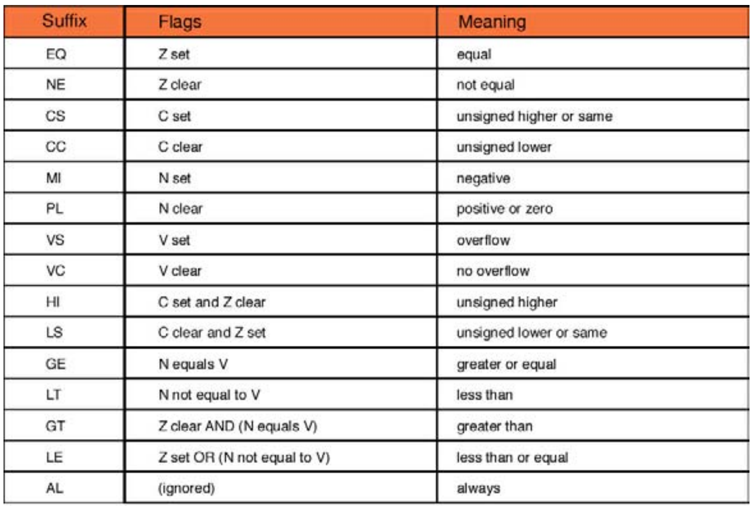


Fig. 7 Instruction Condition Code Bits

The top four bits of each instruction contain a condition code that must be satisfied if the instruction is to be executed. This goes a long way to eliminating small branches in the program code and eliminating stalls in the pipeline so increasing the overall program performance. Thus for small conditional branches of three instructions or less, conditional execution of instructions should be used. For larger jumps, normal branching instructions should be used.



Suffix	Flags	Meaning
EQ	Z set	equal
NE	Z clear	not equal
CS	C set	unsigned higher or same
CC	C clear	unsigned lower
MI	N set	negative
PL	N clear	positive or zero
VS	V set	overflow
VC	V clear	no overflow
HI	C set and Z clear	unsigned higher
LS	C clear and Z set	unsigned lower or same
GE	N equals V	greater or equal
LT	N not equal to V	less than
GT	Z clear AND (N equals V)	greater than
LE	Z set OR (N not equal to V)	less than or equal
AL	(ignored)	always

Fig. 8 Instruction Condition Codes

Thus our ADD instruction below could be prefixed with a condition code, as shown. This adds no overhead to instruction execution

```
EQADD R1, R2,R3  ; If ( Zero flag = 1) then R1 = R2+R3
```

The ARM7 processor also has a 32-bit barrel shifter that allows it to shift or rotate one of the operands in a data processing instruction. This takes place in the same cycle as the instruction. The ADD instruction could be expanded as follows

```
EQADD R1,R2 R3, LSL #2  ; If ( Zero flag = 1) then R1 = R2+(R3 x 4)
```

Finally the programmer may decide if a particular instruction can set the condition code flags in the CPSR.

```
EQADDS R1,R2 R3, LSL #2 ;      If ( Zero flag = 1) then R1 = R2+(R3 x
                               4) and set condition code flags
```

In the ARM instruction set there are no dedicated call or return instructions. Instead these functions are created out of a small group of branching instructions.

The standard branch (B) instruction allows a jump of around +- 32Mb. A conditional branch can be formed by use of the condition codes. For example, a “branch not equal” would be the branching instruction B and the condition code “NE” for not equal giving “BNE”. The next form of the branch instruction is the branch with link. This is the branch instruction but the current value of the PC +4 is saved into R14, the link register. This acts as a CALL instruction by saving the return address into R14. A return instruction is not necessary since a return can be implemented by moving R14 into the PC. The return is more complicated in the case of an interrupt routine. Depending on the type of exception, it may be necessary to modify the contents of the link register to get the correct return address. For example, in the case of an IRQ or FIQ interrupt, the processor will finish its current instruction, increment the PC to the next instruction and then jumping to the vector table. This means that the value in the link register is PC+4 or one instruction ahead of the return address. This means we need to subtract 4 from the value in the link register to get the correct return address. This can be done in a single instruction thus:

SUBS pc, r14, #4 // PC = Link register – 4

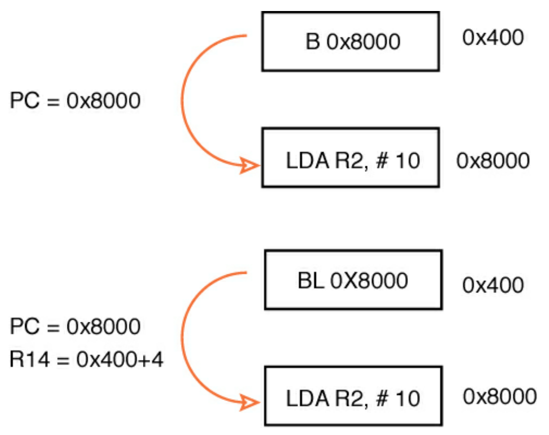


Fig 9 Branch and Branch Link Instruction Operation

Branching instructions are also used to enter the 16-bit Thumb instruction set. Both the branch and branch-with-link may perform an exchange between 32-bit and 16-bit instruction sets and vice versa.

The Branch exchange will jump to a location and start to execute 16-bit Thumb instructions. Branch link exchange will jump to a location, save PC+4 into the link register and start execution of 16-bit Thumb instructions. In both cases, the T bit is set in the CPSR. An equivalent instruction is implemented in the Thumb instruction set to return to 32-bit ARM instruction processing.

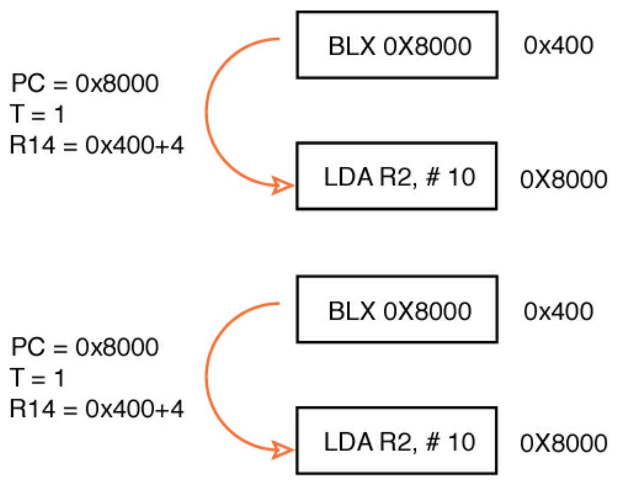


Fig. 10 Branch Exchange and Branch Link Exchange Instruction Operation

Software Interrupts

The ARM instruction set has a software interrupt instruction. Execution of this instruction forces an exception as described above; the processor will enter supervisor mode and jump to the SWI vector at 0x00000008.

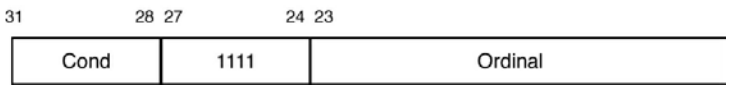


Fig. 11 Software Interrupt Instruction

The bit field 0-23 of the SWI instruction is empty and can be used to hold an ordinal. On execution of an SWI instruction, this ordinal can be examined to determine which SWI procedure to run and gives over 16 million possible SWI functions.

```
…
Swi, #1  ; call swi function one
…

In the swi handler

register unsigned * link_ptr asm ("r14");          // define a pointer to the link register

Switch ((*(link_ptr-1)) & 0x00FFFFFF)              //calculate the number of the swi function
{
Case 0x01 :  SWI_Function1();                      //Call the function
….
}
```

This can be used to provide a hardware abstraction layer. In order to access OS calls or SFR registers, the user code must make a SWI call. All these functions are then running in a supervisor mode, with a separate stack and link register.

As well as instructions to transfer data to and from memory and to CPU registers, the ARM 7 has instructions to save and load multiple registers. It is possible to load or save all 16 CPU registers or a selection of registers in a single instruction. Needless to say, this is extremely useful when entering or exiting a procedure.

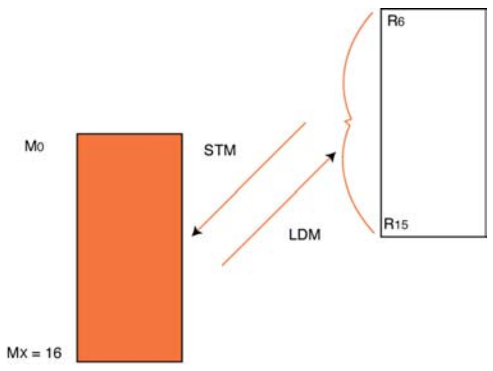


Fig. 12 Load and Store Multiple Instruction Operation

The CPSR and SPSR are only accessed by two special instructions to move their contents to and from a CPU register. No other instruction can act on them directly.

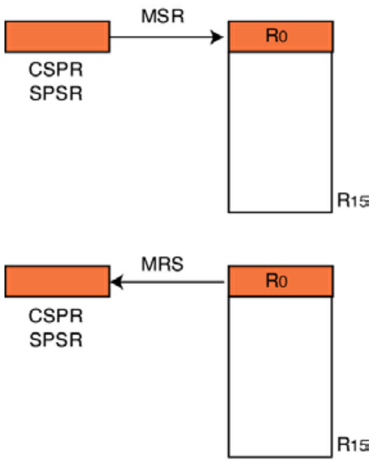


Fig. 13 Programming The SPSR And CPSR Registers

THUMB Support

The ARM processor is capable of executing both 32-bit (ARM) instructions and 16-Bit (Thumb instructions). The Thumb instruction set must always be entered by running a Branch exchange or branch link exchange instruction and NOT by setting the T bit in the CPSR. Thumb instructions are essentially a mapping of their 32 bit cousins but unlike the ARM instructions, they are unconditionally executed except though for branch instructions.

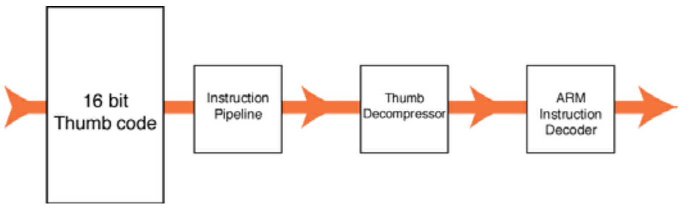


Fig. 14 Thumb Instruction Processing

Thumb instructions only have unlimited access to registers R0-R7 and R13 – R15. A reduced number of instructions can access the full register set.

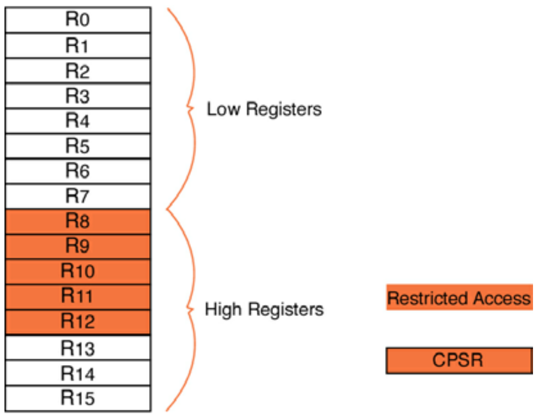


Fig.15 Thumb programmers model

The Thumb instruction set has the same load and store multiple instructions as ARM and in addition, has a modified version of these instructions in the form of PUSH and POP that implement a full descending stack in the conventional manner. The Thumb instruction set also supports the SWI instruction, except that the ordinal field is only 8 bits long to support 256 different SWI calls. When the processor is executing Thumb code and an exception occurs, it will switch to ARM mode in order to process the exception. When the CPSR is restored the, Thumb bit will be reset and the processor continues to run Thumb instructions

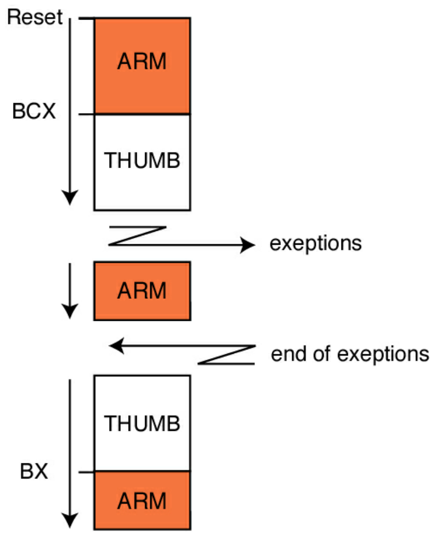


Fig.16 Thumb Exception Processing

Thumb has a much higher code density than ARM code, needing some 70% of the space of the latter. However in a 32-bit memory, ARM code is some 40% faster than Thumb. However it should be noted that if you only have 16-bit wide memory then Thumb code will be faster than ARM code by about 45%. Finally the other important aspect of Thumb is that it can use up to 30% less power than ARM code.