mood-book

UNIT – IV

ARM Architecture

ARM Processor fundamentals:

The first ARM processor was developed on 3-micron technology in 1983-1985. For simplicity, we will be using the ARM6/7 architecture first developed between 1990-1995.

Features of ARM Processor:

- The ARM is a 32-bit architecture.
- When used in relation to the ARM:
  - Byte means 8 bits
  - Half word means 16 bits (two bytes)
  - Word means 32 bits (four bytes)
- Most ARM's implement two instruction setsν 32-bit ARM Instruction Setν 16-bit Thumb Instruction Setν
- Jazelle cores can also execute Java byte code.
- ARM has 37 registers all of which are 32-bits long.
  - 1 dedicated program counter
  - 1 dedicated current program status register
  - 5 dedicated saved program status registers
  - 30 general purpose registersν
- The current processor mode governs which of several banks isν accessible. Each mode can access
  - a particular set of r0-r12 registers
  - a particular r13 (the stack pointer, sp) and r14 (the link register, lr)
  - the program counter, r15 (pc)
  - the current program status register, cpsr

- Privileged modes (except System) can also access a particular spsr (saved program status register)

The ARM is **load/store** architecture in that only **load** and **store** instructions can access memory. All other instructions are register-to-register accesses. This speeds up overall operation as register accesses are much faster than memory accesses.

All ARM instructions are 32 bits wide and are subtle variations on the same regular formatting pattern. This makes decoding the instructions easier, simplifying some of the necessary circuitry and again speeding up the fetch-decode-execute cycle.

All input/output peripherals (such as printers, hard drive disks, and the network) are memory-mapped devices. There is also hardware support for high-bandwidth data transfer.

Most ARM processors are 32-bit architectures throughout. (Those that aren't are 64-bit architectures instead!) Thus they can address a maximum of 232 bytes (or 4 Gigabytes of memory). The word size is 32 bits, with half-words being 16 bits wide. Words are aligned on 4-byte boundaries; half words are aligned on even byte boundaries.

Registers:

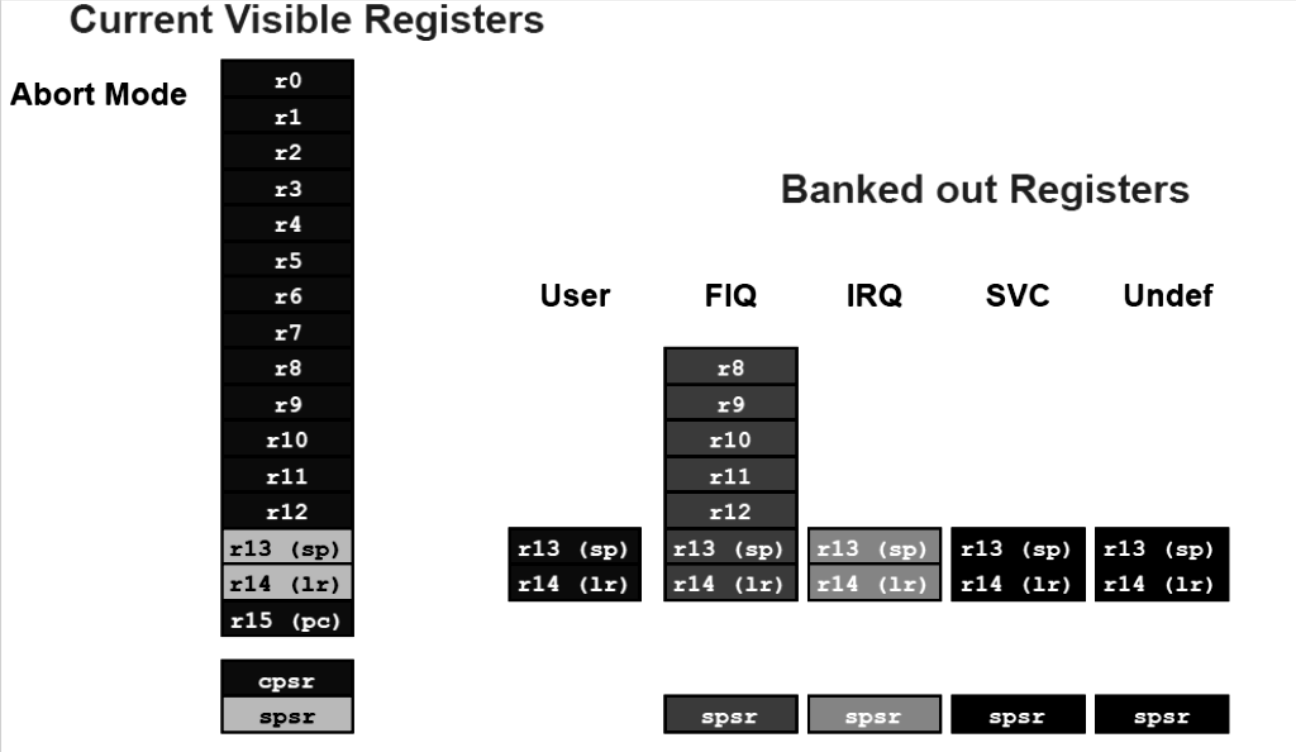
ARM processors provide general-purpose and special-purpose registers. Some additional registers are available in privileged execution modes.

In all ARM processors, the following registers are available and accessible in any processor mode:

- 13 general-purpose registers R0-R_{12}.
- One Stack Pointer (SP).
- One Link Register (LR).
- One Program Counter (PC).
- One Application Program Status Register (APSR).

With the exception of ARMv6-M and ARMv7-M based processors, there are 30 (or 32 if Security Extensions are implemented) general-purpose 32-bit registers, that include the banked SP and LR registers. Fifteen general-purpose registers are visible at any one time, depending on the current processor mode. These are R0-R12, SP, LR. The PC (R15) is not considered a general-purpose register.

The ARM Register Set:



Application Program Status Register:

The Application Program Status Register (APSR) holds the program status flags that are accessible in any processor mode.

It holds copies of the N, Z, C, and V condition flags. The processor uses them to determine whether or not to execute conditional instructions.

Saved Program Status Register:

A *Saved* Program Status Register (SPSR) stores the current value of the CPSR when an exception is taken so that the CPSR can be restored after handling the exception.

Each exception handling mode can access its own SPSR. User mode and System mode do not have an SPSR because they are not exception handling modes.

Current Program Status Register:

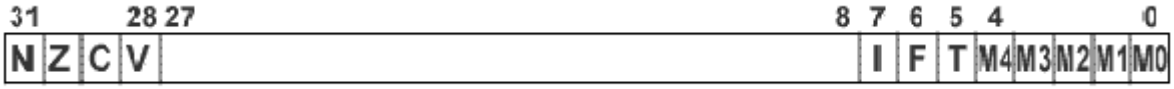
The Current Program Status Register is a 32-bit wide register used in the ARM architecture to record various pieces of information regarding the state of the program being

executed by the processor and the state of the processor. This information is recorded by setting or clearing specific bits in the register. The Current Program Status Register (CPSR) holds the same program status flags as the APSR, and some additional information.

The CPSR holds:

- The APSR flags.
- The processor mode.
- The interrupt disable flags.
- The instruction set state (ARM, Thumb, ThumbEE, or Jazelle®).
- The endianness state (on ARMv4T and later).
- The execution state bits for the IT block (on ARMv6T2 and later).

The format of CPSR register is shown below.

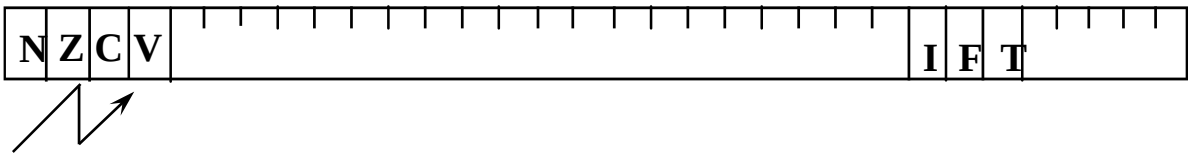


The top four bits (bits 31, 30, 29, and 28) are the condition code (**cc**) bits and are of most interest to us. Condition code bits are sometimes referred to as "flags". The lowest 8 bits (bit 7 through to bit 0) store information about the processor's own state. The remaining bits (i.e. bit 27 to bit 8) are currently unused in most ARM processors.

- The N bit is the "negative flag" and indicates that a value is negative. The Z bit is the "zero flag" and is set when an appropriate instruction produces a zero result.
- The C bit is the "carry flag" but it can also be used to indicate "borrows" (from subtraction operations) and "extends" (from shift instructions (LINK)).
- The V bit is the "overflow flag" which is set if an instruction produces a result that overflows and hence may go beyond the range of numbers that can be represented in 2's complement signed format.

For completeness, the other state bits are:

- The I and F bits which determine whether interrupts (such as requests for input/output) are enabled or disabled.
- The T bit which indicates whether the processor is in "Thumb" mode, where the processor can execute a subset of the assembly language as 16-bit compact instructions. As Thumb code packs more instructions into the same amount of memory, it is an effective solution to applications where physical memory is at a premium.
- The M4 to M0 bits are the mode bits. Application programs normally run in user mode (where the mode bits are 10000). Whenever an interrupt or similar event occurs, the processor switches into one of the alternative modes allowing the software handler greater privileges with regard to memory manipulation.



Copies of the ALU status flags
(latched if the instruction has the "S" bit set).

* **Condition Code Flags**

N = **N**egative result from ALU flag. Z = **Z**ero result from ALU flag.
C = ALU operation **C**arried out V = ALU operation o**V**erflowed

* **Mode Bits**

M[4:0] define the processor mode

* **Interrupt Disable bits. I** = 1, disables the IRQ. **F** = 1, disables the FIQ.

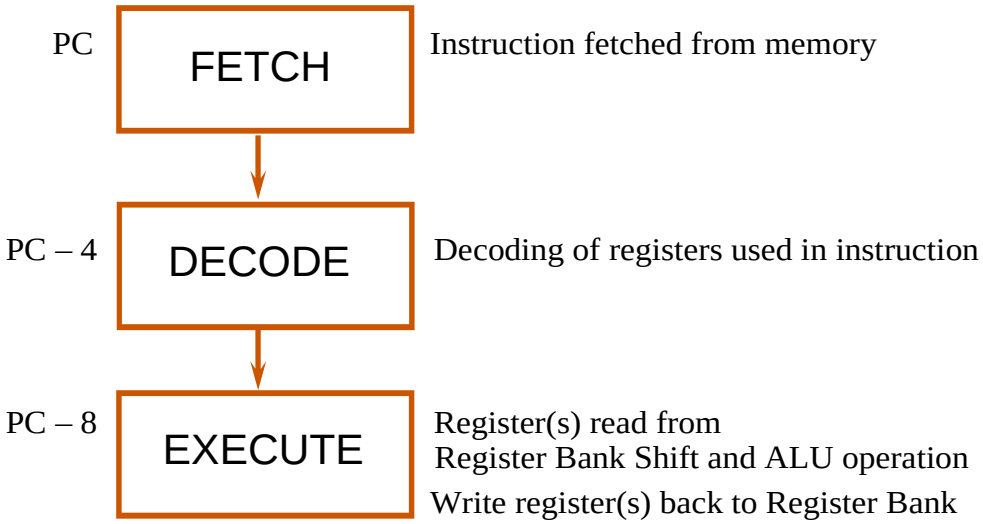
* **T Bit (Architecture v4T only)**

  T = 0, Processor in ARM state

  T = 1, Processor in Thumb state

Program Counter (r15):

- When the processor is executing in ARM state:
  - All instructions are 32 bits wide
  - All instructions must be word aligned
  - Therefore the pc value is stored in bits [31:2] with bits [1:0] undefined (as instruction cannot be half word or byte aligned).
- When the processor is executing in Thumb state:
  - All instructions are 16 bits wide
  - All instructions must be half word aligned
  - Therefore the pc value is stored in bits [31:1] with bit [0] undefined (as instruction cannot be byte aligned).
- When the processor is executing in Jazelle state:
  - All instructions are 8 bits wide
  - Processor performs a word access to read 4 instructions at once



Pipeline:

A pipeline is the mechanism a RISC processor uses to execute instructions. Using a pipeline speeds up execution by fetching the next instruction while other instructions are being decoded and executed. One way to view the pipeline is to think of it as an automobile assembly line, with each stage carrying out a particular task to manufacture

the vehicle.

ARM is having three-stage pipeline.

- Fetch loads an instruction from memory.
- Decode identifies the instruction to be executed.
- Execute processes the instruction and writes the result back to a register.

- The ARM uses a pipeline in order to increase the speed of the flow of instructions to the processor.
  - Allows several operations to be undertaken simultaneously, rather than serially.

Processor Modes:

The ARM has seven basic operating modes:

- **User :** unprivileged mode under which most tasks run
- **FIQ :** entered when a high priority (fast) interrupt is raised
- **IRQ :** entered when a low priority (normal) interrupt is raised
- **Supervisor :** entered on reset and when a Software Interrupt
- **instruction is executed Abort :** used to handle memory access violations
- **Undef :** used to handle undefined instructions
- **System:** privileged mode using the same registers as user mode.

Exceptions, Interrupts, and the Vector Table:

When an exception or interrupt occurs, the processor sets the pc to a specific memory address. The address is within a special address range called the vector table.

The entries in the vector table are instructions that branch to specific routines designed to handle a particular exception or interrupt.

The memory map address 0x00000000 is reserved for the vector table, a set of 32-bit words. On some processors the vector table can be optionally located at a higher address

in memory (starting at the offset 0xffff0000). Operating systems such as Linux and Microsoft's embedded products can take advantage of this feature.

When an exception or interrupt occurs, the processor suspends normal execution and starts loading instructions from the exception vector table (see Table 2.6). Each vector table entry contains a form of branch instruction pointing to the start of a specific routine:

Reset vector is the location of the first instruction executed by the processor when power is applied. This instruction branches to the initialization code.

Undefined instruction vector is used when the processor cannot decode an instruction. Software interrupt vector is called when you execute a SWI instruction. The SWI instruction is frequently used as the mechanism to invoke an operating system routine.

Prefetch abort vector occurs when the processor attempts to fetch an instruction from an address without the correct access permissions. The actual abort occurs in the decode stage.

Data abort vector is similar to a prefetch abort but is raised when an instruction attempts to access data memory without the correct access permissions.

Interrupt request vector is used by external hardware to interrupt the normal execution flow of the processor. It can only be raised if IRQs are not masked in the CPSR, which is explained below.

When an exception occurs, the core:

- Copies CPSR into SPSR_<mode>
- Sets appropriate CPSR bits
    - If core implements ARM Architecture 4T and is currently in Thumb state, then

0x00000000	Reset
0x00000004	Undefined Instruction
0x00000008	Software Interrupt
0x0000000C	Prefetch Abort
0x00000010	Data Abort
0x00000014	Reserved
0x00000018	IRQ
0x0000001C	FIQ

- ARM state is entered.
- Mode field bits
- Interrupt disable flags if appropriate.

– Maps in appropriate banked registers

– Stores the “return address” in LR_<mode>

– Sets PC to vector address

- To return, exception handler needs to:

– Restore CPSR from SPSR_<mode>

– Restore PC from LR_<mode>

ARM Instruction Set:

Main features of the ARM Instruction Set are

- All instructions are 32 bits long.
- Most instructions execute in a single cycle.
- Most instructions can be conditionally executed.

- A load/store architecture

– Data processing instructions act only on registers

– Three operand format

– Combined ALU and shifter for high speed bit manipulation

– Specific memory access instructions with powerful auto- indexing addressing modes.

- 32 bit and 8 bit data types
- and also 16 bit data types on ARM Architecture v4.

– Flexible multiple register load and store instructions

- Instruction set extension via coprocessors

Very dense 16-bit compressed instruction set (Thumb)

Data processing Instructions:

- Largest family of ARM instructions, all sharing the same instruction format.
- Contains:
  - Arithmetic operations
  - Comparisons (no results - just set condition codes)
  - Logical operations
  - Data movement between registers
- Remember, this is a load / store architecture
  - These instructions only work on registers, NOT memory.
- They each perform a specific operation on one or two operands.
  - First operand always a register - Rn
  - Second operand sent to the ALU via barrel shifter.

Arithmetic Operations:

Operations are:

- ADD operand1 + operand2 ; Add
- ADC operand1 + operand2 + carry ; Add with carry
- SUB operand1 - operand2 ; Subtract
- SBC operand1 - operand2 + carry -1 ; Subtract with carry
- RSB operand2 - operand1 ; Reverse subtract
- RSC operand2 - operand1 + carry – 1; Reverse subtract with carry

- Syntax:
  - <Operation>{<cond>}{S} Rd, Rn, Operand2
- Examples

- ADD r0, r1, r2
- SUBGT r3, r3, #1
- RSBLES r4, r5, #5

Comparison instructions:

- The only effect of the comparisons is to update the condition flags. Thus no need to set S bit.
- Operations are:
  - CMP operand1 - operand2 ; Compare
  - CMN operand1 + operand2 ; Compare negative
  - TST operand1 AND operand2 ; Test
  - TEQ operand1 EOR operand2 ; Test equivalence
- Syntax:
  - <Operation>{<cond>} Rn, Operand2
- Examples:
  - CMP r0, r1
  - TSTEQ r2, #5

Logical Operations:

Operations are:

AND operand1 AND operand2
EOR operand1 EOR operand2
ORR operand1 OR operand2
ORN operand1 NOR operand2
BIC operand1 AND NOT operand2 [ie bit clear]

Syntax:

- <Operation>{<cond>}{S} Rd, Rn, Operand2

Examples:

AND r0, r1, r2

BICEQ r2, r3, #7

EORS r1,r3,r0

Data Movement:

Operations are:

MOV operand2

MVN NOT operand2

Note that these make no use of operand1.

Syntax:

<Operation>{<cond>}{S} Rd, Operand2

Examples:

MOV r0, r1

MOVS r2, #10

MVNEQ r1,#0

Barrel Shifter:

Data processing instructions are processed within the arithmetic logic unit (ALU). Unique and powerful feature of the ARM processor is the ability to shift the 32-bit binary pattern in one of the source registers left or right by a specific number of positions before it enters the ALU. This shift increases the power and flexibility of many data processing operations.

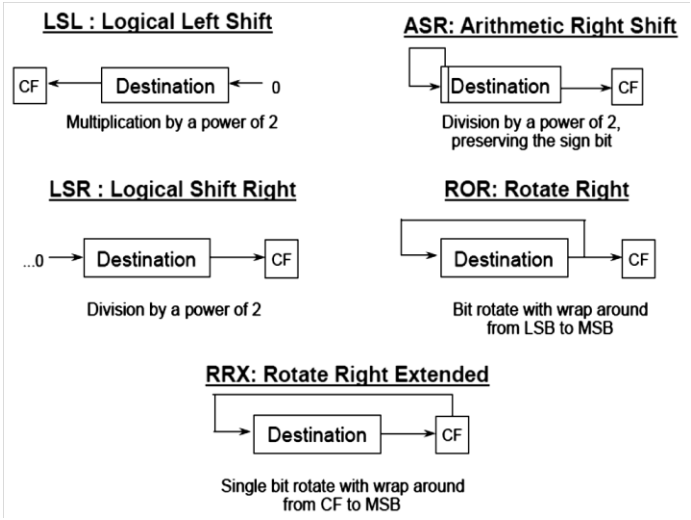
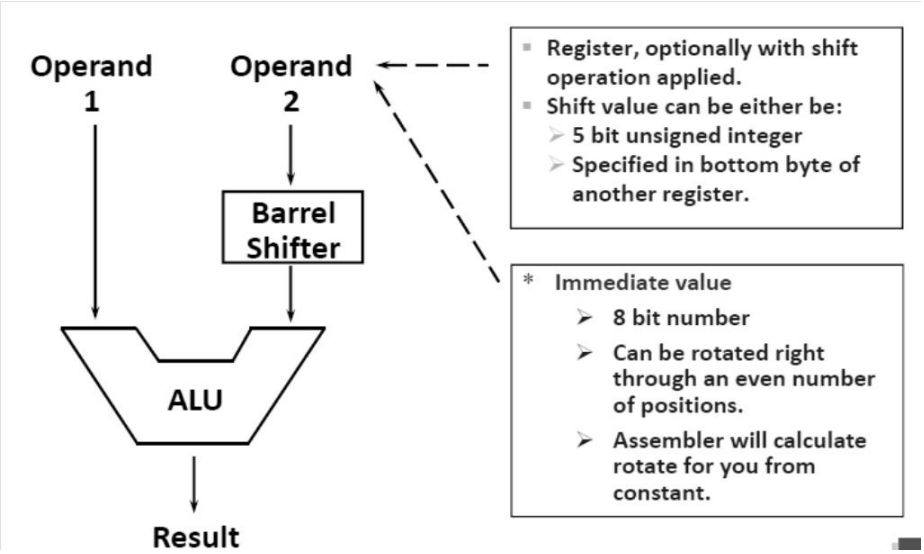


Fig. Barrel shifter operations.



Multiplication Instructions:

- The Basic ARM provides two multiplication instructions.
- Multiply
  - MUL{<cond>}{S} Rd, Rm, Rs ; Rd = Rm * Rs
- Multiply Accumulate ; does addition for free
  - MLA{<cond>}{S} Rd, Rm, Rs,Rn ; Rd = (Rm * Rs) + Rn
- Restrictions on use:
  - Rd and Rm cannot be the same register
    - Can be avoided by swapping Rm and Rs around. This works because multiplication is commutative.
  - Cannot use PC.
- These will be picked up by the assembler if overlooked.
- Operands can be considered signed or unsigned

- Up to user to interpret correctly.

Branch Instructions:

A branch instruction changes the flow of execution or is used to call a routine. This type of instruction allows programs to have subroutines, if-then-else structures, and loops.

The change of execution flow forces the program counter pc to point to a new address. The ARMv5E instruction set includes four different branch instructions.

Syntax: B{<cond>} label

BL{<cond>}

label

BX{<cond>} Rm

BLX{<cond>}

label | Rm

B	branch	pc = label
BL	branch with link	pc = label lr = address of the next instruction after the BL
BX	branch exchange	pc = Rm & 0xfffffffe, T = Rm & 1
BL X	Branch exchange with link	pc = label, T = 1 pc = Rm & 0xfffffffe, T = Rm & 1 lr = address of the next instruction after the BLX

The address label is stored in the instruction as a signed pc-relative offset and must be within approximately 32 MB of the branch instruction. T refers to the Thumb bit in the CPSR. When instructions set T, the ARM switches to Thumb state.

Conditional Branches:

Branch	Interpretation	Normal uses
B	Unconditional	Always take this branch
BAL	Always	Always take this branch
BEQ	Equal	Comparison equal or zero result
BNE	Not equal	Comparison not equal or non-zero result
BPL	Plus	Result positive or zero
BMI	Minus	Result minus or negative
BCC	Carry clear	Arithmetic operation did not give carry-out
BLO	Lower	Unsigned comparison gave lower
BCS	Carry set	Arithmetic operation gave carry-out
BHS	Higher or same	Unsigned comparison gave higher or same
BVC	Overflow clear	Signed integer operation; no overflow occurred
BVS	Overflow set	Signed integer operation; overflow occurred
BGT	Greater than	Signed integer comparison gave greater than
BGE	Greater or equal	Signed integer comparison gave greater or equal
BLT	Less than	Signed integer comparison gave less than
BLE	Less or equal	Signed integer comparison gave less than or equal
BHI	Higher	Unsigned comparison gave higher
BLS	Lower or same	Unsigned comparison gave lower or same

Load-Store Instructions:

Load-store instructions transfer data between memory and processor registers. There are three types of load-store instructions: single-register transfer, multiple-register transfer, and swap.

Single-Register Transfer:

These instructions are used for moving a single data item in and out of a register. The data types supported are signed and unsigned words (32-bit), halfwords (16-bit), and bytes. Here are the various load-store single-register transfer instructions.

Syntax: <LDR|STR>{<cond>}{B} Rd,addressing[1]

LDR{<cond>}SB|H|SH Rd, addressing

STR{<cond>}H Rd, addressing

Multiple-Register Load-Store Instructions:

The Thumb versions of the load-store multiple instructions are reduced forms of the ARM load- store multiple instructions. They only support the increment after (IA) addressing mode.

Syntax : <LDM|STM>IA Rn!, {low Register list}

Stack Instructions:

The Thumb stack operations are different from the equivalent ARM instructions because they use the more traditional POP and PUSH concept.

Syntax: POP {low_register_list{, pc}}

PUSH {low_register_list{, lr}}

Software Interrupt Instruction:

Similar to the ARM equivalent, the Thumb software interrupt (SWI) instruction causes a software interrupt exception. If any interrupt or exception flag is raised in Thumb state, the processor automatically reverts back to ARM state to handle the exception.

Syntax: SWI immediate

Swap Instruction:

The swap instruction is a special case of a load-store instruction. It swaps the contents of memory with the contents of a register. This instruction is an atomic operation—it reads and writes a location in the same bus operation, preventing any other instruction from reading or writing to that location until it completes.

Syntax :

SWP{B}{<cond>} Rd,Rm,[Rn]

When the processor executes an SWI instruction, it sets the program counter pc to the offset 0x8 in the vector table. The instruction also forces the processor mode to SVC, which allows an operating system routine to be called in a privileged mode.Each SWI instruction has an associated SWI number, which is used to represent a particular function call or feature.

Program Status Register Instructions:

The ARM instruction set provides two instructions to directly control a program status register (psr). The MRS instruction transfers the contents of either the cpsr or spsr into register; in the reverse direction, the MSR instruction transfers the contents of a register into the cpsr or spsr. Together these instructions are used to read and write the cpsr and spsr.

In the syntax you can see a label called fields. This can be any combination of control (c), extension (x), status (s), and flags (f). These fields relate to particular byte regions in

Syntax: MRS{<cond>} Rd,<cpsr|spsr>

MSR{<cond>} <cpsr|spsr>_<fields>,Rm

MSR{<cond>} <cpsr|spsr>_<fields>,#immediate

Loading Constants:

You might have noticed that there is no ARM instruction to move a 32-bit constant into register. Since ARM instructions are 32 bits in size, they obviously cannot specify a general 32- bit constant.

To aid programming there are two pseudoinstructions to move a 32 - bit value into Register.

Syntax: LDR Rd, =constant

ADR Rd, label

Conditional Execution:

Most ARM instructions are conditionally executed—you can specify that the instruction only executes if the condition code flags pass a given condition or test. By using conditional execution instructions you can increase performance and code density.

The condition field is a two-letter mnemonic appended to the instruction mnemonic. The default mnemonic is AL, or always execute.

Conditional execution reduces the number of branches, which also reduces the number of pipeline flushes and thus improves the performance of the executed code. Conditional execution depends upon two components: the condition field and condition flags. The condition field is located in the instruction, and the condition flags are located inthe cpsr.

Introduction to the Thumb Instruction Set:

Thumb encodes a subset of the 32-bit ARM instructions into a 16-bit instruction set space. Since Thumb has higher performance than ARM on a processor with a 16-bit data bus, but lower performance than ARM on a 32-bit data bus, use Thumb for memory-constrained systems.

Thumb has higher code density—the space taken up in memory by an executable program—than ARM. For memory-constrained embedded systems, for example, mobile phones and PDAs, code density is very important. Cost pressures also limit memory size, width, andspeed.

- Thumb is a 16-bit instruction set
  - Optimized for code density from C code
  - Improved performance form narrow memory
  - Subset of the functionality of the ARM instruction set
- Core has two execution states – ARM and Thumb
  - Switch between them using BX instruction
- Thumb has characteristic features:
  - Most Thumb instruction are executed unconditionally
  - Many Thumb data process instruction use a 2-address format
  - Thumb instruction formats are less regular than ARM instruction formats, as a result of the dense encoding.

Thumb Instruction Set: The following table summarizes the THUMB instruction set.

Mnemonic	Instruction
ADC	Add with Carry
ADD	Add
AND	AND
ASR	Arithmetic Shift Right
B	Unconditional branch
B*xx*	Conditional branch
BIC	Bit Clear
BL	Branch and Link
BX	Branch and Exchange
CMN	Compare Negative
CMP	Compare
EOR	EOR
LDMIA	Load multiple
LDR	Load word
LDRB	Load byte
LDRH	Load halfword
LSL	Logical Shift Left
LDSB	Load sign-extended byte
LDSH	Load sign-extended halfword
LSR	Logical Shift Right
MOV	Move register
MUL	Multiply
MVN	Move Negative register

Mnemonic	Instruction
NEG	Negate
ORR	OR
POP	Pop registers
PUSH	Push registers
ROR	Rotate Right
SBC	Subtract with Carry
STMIA	Store Multiple
STR	Store word
STRB	Store byte
STRH	Store halfword
SWI	Software Interrupt
SUB	Subtract
TST	Test bits