THE ARM PROCESSOR:

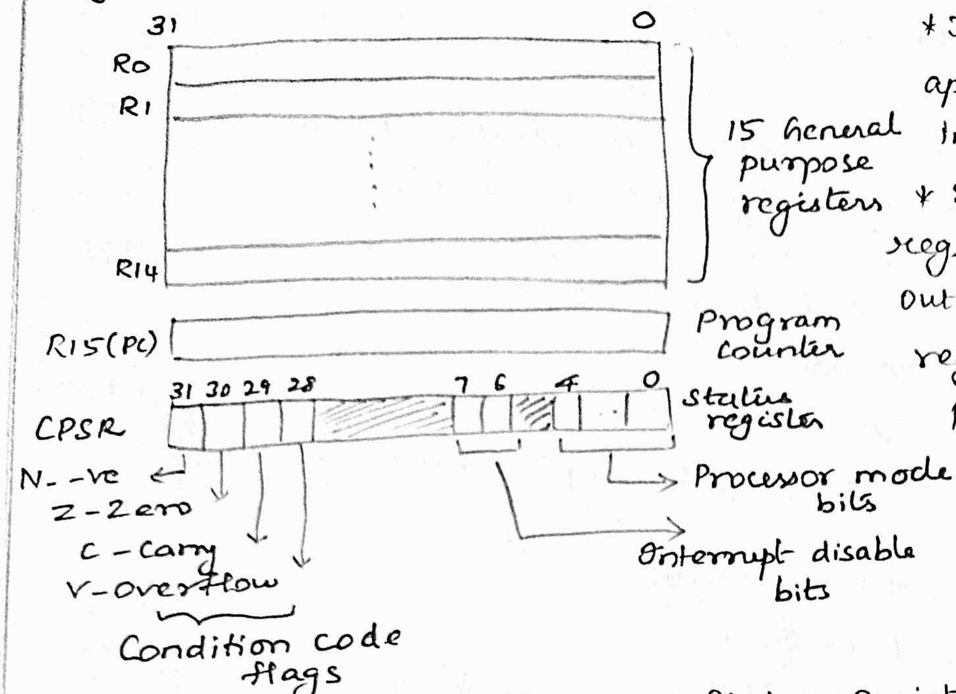
* The Advanced RISC Machines (ARM) Limited has designed a family of microprocessors referred as ARM processor. It is used in low-power and low-cost embedded applications such as mobile telephones, communication modems, automotive engine management systems and hand-held digital assistants.

PROCESSOR AND CPU CORES:

* In ARM architecture, memory is byte addressable., using 32-bit addresses and processor registers are 32 bits long.

Register Structure :

* The processor register used by application programs are given in figure.
* There are sixteen 32-bit registers labeled R0 through R15, out of which 15 general purpose registers (R0-R14) and one Program Counter (R15).

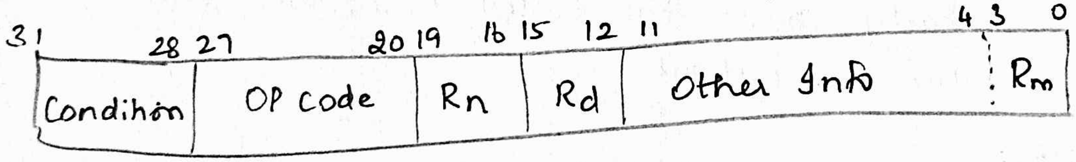


* The Current Program Status Register (CPSR) or Status Register, holds the condition code flags (N, Z, C, V), interrupt disable flags and processor mode bits.

* There are 15 additional general purpose registers called the banked registers. They are duplicates of R0 to R14 registers. They are used when the processor switches into Supervisor or Interrupt modes of operation.

Memory Access Instructions and Addressing Modes:

* Each instruction in ARM architecture is encoded into a 32-bit word. Access to memory is provided only by Load and Store instructions.



ARM Instruction Format

* An instruction specifies a conditional execution code, the OP code, 2 or 3 registers (Rn, Rd and Rm) and some other information.

* In Load instruction, the operand is transferred from memory into general-purpose register named in 4-bit Rd field.

* In store instruction, the operand is transferred from Rd into memory. If the operand is a byte, it is always located in the low-order byte position of the register.

nditional Execution of Instructions :

* The instruction in ARM processor is executed only if the current state of the processor Condition code flags satisfies the condition specified in bits b31-28 of the instruction.

Memory Addressing Mode :

* The basic method for addressing memory operands is to generate effective address, EA of the operand by adding signed offset to the contents of a base register, Rn, specified in the instruction.

eg, Load instruction

LDR Rd, [Rn;# offset]

specifies the offset in immediate mode and performs the operation

Rd ← [[Rn] + offset]

* the instruction LDR Rd,[Rn,Rm] performs the operation Rd ← [[Rn]+[Rm]]

* An offset of zero does not have to be specified explicitly. Hence, the instruction LDR Rd, [Rn] performs the operation Rd ← [[Rn]].

* The OP-code mnemonic LDR specifies that a 32-bit word is loaded from memory into register. A byte operand can be loaded into low-order byte position of a register by mnemonic LDRB. Higher order bits filled with 0.

* Store instruction have the mnemonics STR and STRB : eg, STR Rd,[Rn] performs the operation [Rn] ← [Rd] transferring a word operand into memory.

Three Addressing Modes :

→ Pre-indexed mode - The EA of operand is sum of contents of base register Rn and an offset value.

→ Pre-indexed with writeback mode - EA is calculated as in pre-indexed and EA is written back into Rn.

→ Post-indexed mode - The EA of operand is contents of Rn. The offset is added to this address and result written back into Rn.

ARM indexed Addressing Mode :

Name.	Assembler syntax	Addressing function
With immediate offset		
Pre-indexed	[Rn, #offset]	EA = [Rn] + offset

Pre-indexed with writeback	[Rn, #offset]!	EA = [Rn] + offset
		Rn ← [Rn] + offset

Post-indexed	[Rn], #offset	EA = [Rn]
		Rn ← [Rn] + offset.

With offset magnitude in Rm:

Pre-indexed	[Rn, ± Rm, shift)	EA = [Rn] ± [Rm] shifted

Pre-indexed with writeback	[Rn ± Rm, shift] !	EA = [Rn] ± [Rm] shifted
		Rn ← [Rn] ± [Rm] shifted

Post-indexed	[Rn], ± Rm, shift	EA = [Rn]
		Rn ← [Rn] ± [Rm] shifted

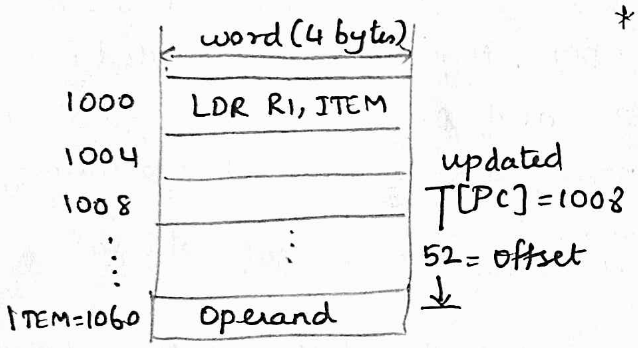
Relative (Pre-indexed with immediate offset)	Location	EA = Location = [PC] + offset

eg1, LDR R0, [R1, −R2] !

performs the operation R0 ← [[R1] − [R2]]. The effective address of the operand, [R1] − [R2) is then loaded into R1 because writeback is specified by the exclamation mark.

eg 2, LDR R0, [R1, −R2, LSL, # 4] !

This instruction perform the operation R0 ← [[R1]] − 16 × [R2]]

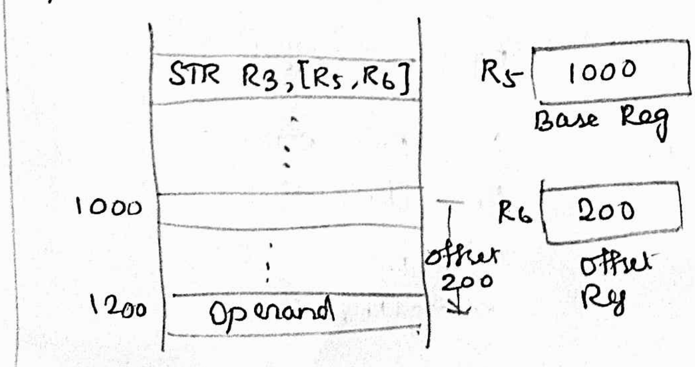


* Relative Addressing Mode:

* The address of the operand is given symbolically as ITEM in inst, is 1060. This is implemented by Pre-indexed mode with an immediate offset, using PC as base register.

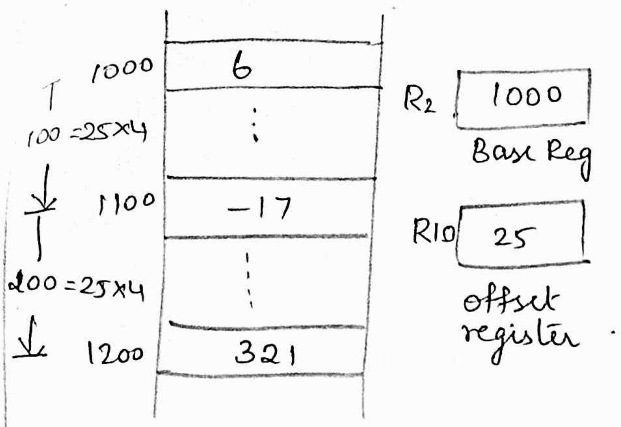
* The offset calculated by assembler is 52 because the updated PC will contain 1008 when the offset is added to it during program execution, and the EA to be generated is 1060 = 1008 + 52.

* The operand must be within the range of ±4095 bytes from the updated PC. Otherwise, an error is indicated by the assembler.



* This is an example of Pre-Indexed mode with the offset contained in register R6 and base value contained in R5.

* The Store instruction stores the Contents of R3 into memory word location 1200.



LDR R1, [R2], R10, LSL #2

*Post-indexed with writeback.

*The first row of a 25X25 matrix of no.s stored in column order. The first no. of first row of matrix is stored in word location 1000.

*The numbers at addresses 1100, 1200, ... are successive numbers of first row. The 25 memory locations 1000, 1004, 1008...1096 Contain the 1st column of the matrix.

THUMB INSTRUCTION SET.

*ARM processors are mainly intended for embedded system applications. There have been 5 major versions of ARM ISA, labelled V1 through V5.

*Version V1 and V2 supported only 26-bit memory addressing. Version V3 introduced full 32-bit addressing for byte and 32-bit word operands.

*Version V4 contains full 64-bit instructions as well as 32-bit. Version V5 and an extension of it labeled V5E, add specialized instructions for: managing software breakpoints for debugging, normalizing numbers in floating point operations, performing addition and multiplication operations on 16-bit operands.

*The ARM ISA specification includes a compact encoding of subset of V4 and V5 versions of full set of instructions. The subset is called the Thumb instruction set. and the version names are extended to V4T and V5T to denote this inclusion. All thumb instructions are encoded into a 16-bit half-word format.

*The practical motivation for the Thumb instructions is that they lead to a reduction in memory space needed to store programs used in low-cost and low-power embedded system applications...

Execution of Programs with Thumb Instruction:

*The instructions are fetched from memory and decompressed from their highly encoded 16-bit format into corresponding std 32-bit ARM instructions and then executed.

* A bit in Current Program Status Register (CPSR), labeled T, determines whether the incoming instruction stream consists of Thumb (T=1) or Std 32-bit ARM instructions (T=0). An application can contain mix of Thumb and std. instruction routines.

* Difference between Thumb and std. instructions:

(i) Thumb instructions use a 2-operand format in which destination reg. is one of source operand register.

(ii) Conditional execution, is used for only branches in Thumb set.

PROCESSOR AND CPU CORES:

* ARM designs called cores are provided in 2 different forms: hard macrocell or synthesizable.

* The hard macrocell version is a detailed physical layout, targeted to a particular chip fabrication process.

* The synthesizable form is a high-level language software module that can be synthesized using a suitable cell library in the required target technology.

* ARM designs are classified as either processor cores or CPU cores. A processor core contains only a processor and associated address and data bus connections. A CPU core contains cache and memory management components in addition to a processor.

ARM7TDMI Processor Core

* The core is commonly used for low-cost low-power applications. The processor has a 3-stage pipelining of fetch, decode and execute stages.

* It realizes version V4T of architecture, supporting both Thumb and standard instruction sets.

ARM9TDMI and ARM10TDMI Processor Core.

* They are based on 5-stage and 6-stage pipelines, respectively. They have seperate instruction and data ports to provide much higher performance levels.

* The ARM10TDMI has a wider 64-bit path to each memory post, as compared 32-bit paths for the other 2 processors.

* The ARM9TDMI and ARM10TDMI implement versions V4T and V5TE of ISA. Both decode Thumb instructions directly for execution.

ARM720T CPU Core:

* This core consists of ARM7TDMI processor core combined with an 8-K byte unified instruction and data cache

* The memory management unit uses a 64-entry associative translation lookaside buffer for holding recent translations.

* The clock rate for this integrated unit can be up to 60MHz.

ARM920T and ARM1020E CPU Cores.

* These CPU cores, based on ARM9TDMI and ARM10TDMI processor core, have seperate instruction and data caches.

* Each of the caches in the ARM920T contains 16K bytes and has 32-byte blocks, with 64-way set associativity

* The ARM1020E has 32K bytes in each cache.

Strong ARM SA-110 CPU Core

* The core was developed by ARM in collaboration with Digital Equipment Corpostion and manufactued by Intel. It implements version V4 of the architecture.

* It is comparable to ARM920T in performance, but implemented using an earlier technology and has higher power consumption.

* The Strong ARM processor has a 5-stage pipeline. There are seperate 16-K byte instruction and data caches.

INSTRUCTION ENCODING.

MEMORY LOAD AND STORE INSTRUCTIONS

Memory Addressing Modes in ARM:

1. Pre-indexed mode: The effective address of the operand is the sum of the contents of base register Rn and an offset value.
2. Pre-indexed with writeback mode - The effective address of the operand is generated in same way as in pre-indexed mode, and then the effective address is written back into Rn.

3. Post indexed mode : The effective address of the operand is the contents of Rn. The offset is then added to this address and the result is written back into Rn.

* The exclamation mark signifies writeback in pre-indexed mode. Post-indexed mode always involves writeback, so exclamation mark is not needed.

* Offset values can be directly given in instruction as immediate or can be given in register Rm.

1) With immediate offset : ⇒ offset value is in range ±4095.

Pre-indexed :

LDR Rd, [Rn, #offset]

EA = [Rn] + offset ⇒ Rd ← [[Rn] + offset]

Pre-indexed with writeback :

LDR Rd, [Rn, # offset]

EA = [Rn] + offset

Rd ← [[Rn] + offset] and Rn ← [Rn] + offset.

Post-indexed.

LDR Rd, [Rn], #offset.

EA = [Rn]

Rd ← [[Rn]]

Rn ← [Rn] + offset.

2) With offset magnitude in Rm :

Pre-indexed:

LDR Rd, [Rn ±Rm]

EA ← [Rn] ± [Rm]

Rd ← [[Rn] + [Rm]]

Pre-indexed with writeback:

LDR Rd, [Rn, ± Rm]

EA = [Rn] ± [Rm]

Rd ← [[Rn] ± [Rm]] & Rn ← [Rn] ± [Rm]

Post-indexed

LDR Rd, [Rn], [± Rm]

EA = [Rn]

Rd ← [[Rn]]

Rn ← [Rn] ± [Rm].

LDR R0, [R1, −R2] ! (Pre indexed writeback)

$R_0 \leftarrow [[R1] - [R2]]$

EA = [R1] − [R2] is loaded into R_1 (ie) $R_1 \leftarrow [R1] - [R2]$.

* When offset is given in register, it may be scaled by power of 2 by shifting to right or left. This is indicated by placing the shift direction, LSL for left shift and LSR for right shift. and the shift amount. The shift amount is in range 0 to 31.

eg. LDR R0, [R1, −R2, LSL #4] !

$R_0 \leftarrow [[R1] - 16 \times [R2]]$ (ie) the contents of [R2] multiplied by $4^2 = 16$.

$R_1 \leftarrow [R1] - 16 \times [R2]$.

Relative Addressing mode: The PC may be used in place of Base register Rn.

Instruction format for Load / Store Instruction:

31 28								19 16	15 12	11 0
Condition	0 1	I	P	U	B	W	L	Rn	Rd	offset

Rd → Source / Destination Reg

Rn → Base Register

L → Load / Store
0: Store to Memory (STR)
1: Load from Memory (LDR)

W → Writeback
0: No Writeback
1: Write address into Rn

B → Byte / Word
0: Word
1: Byte

U → Up / Down
0: Subtract offset from [Rn]
1: Add offset to [Rn]

P → Pre / Post indexing
0: Apply offset after transfer
1: Apply offset before transfer

I →
0: Offset = unsigned immediate value
1:

11 5	3	0
Shift	0	Rm

offset = shift [Rm]

* The L-bit, b_{20}, is 1 for Load (LDR) instruction and 0 for a store (STR) instruction.

* The B-bit, b_{22}, is 1 for byte operand and 0 for 32-bit word operand.

* The effective address of the memory operand is determined by adding (U=1) or Subtracting (U=0) the offset specified by the offset field with the contents of register Rn

* The P and W bits determine pre- or post-indexing and writeback operations.

* The I bit determines how the offset field is interpreted.

Eg. 1. LDR R0, [R1, #100], the operation performed is R0 ← [[R1]+100] and the bit settings are I=0, P=1, U=1, B=0, W=0 and L=1.

Eg 2 LDR R0, [R1, R2], the operation done is R0 ← [[R1]+[R2]] with I bit changed to 1 and all other settings the same.

Eg 3: When the offset is contained in a register, it can be shifted before being added to or subtracted from base register Rn. The shift can be specified by 5-bit immediate method.

LDR R0, [R1, −R2, LSL, #4]!

performs the operation

R0 ← [[R1] − 16 × [R2]]

and the effective address is written back into R1. The bit settings for this instruction are I=1, P=1, U=0, B=0, W=1 and L=1.