

UNIT-IV

ARM

ARM PROCESSOR FUNDAMENTALS :-

ARM's are 16 or 32 bit processors that provide precise computation facilities, minimum power dissipation, high speed, and support small code size.

The architecture features of an ARM processor are,

1. It possess HEMOS technology with small die size that results in low power consumption and low voltage operations.
2. It has a large register set consists of sixteen 32-bit registers.
3. It performs a three stage pipelining. Figure illustrates the pipelining process.

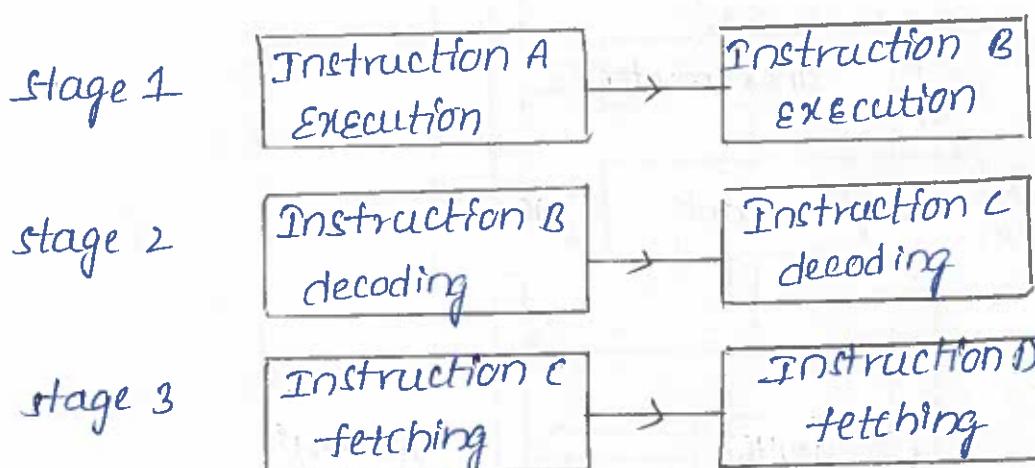


Fig : Three stage pipelining in ARM processors

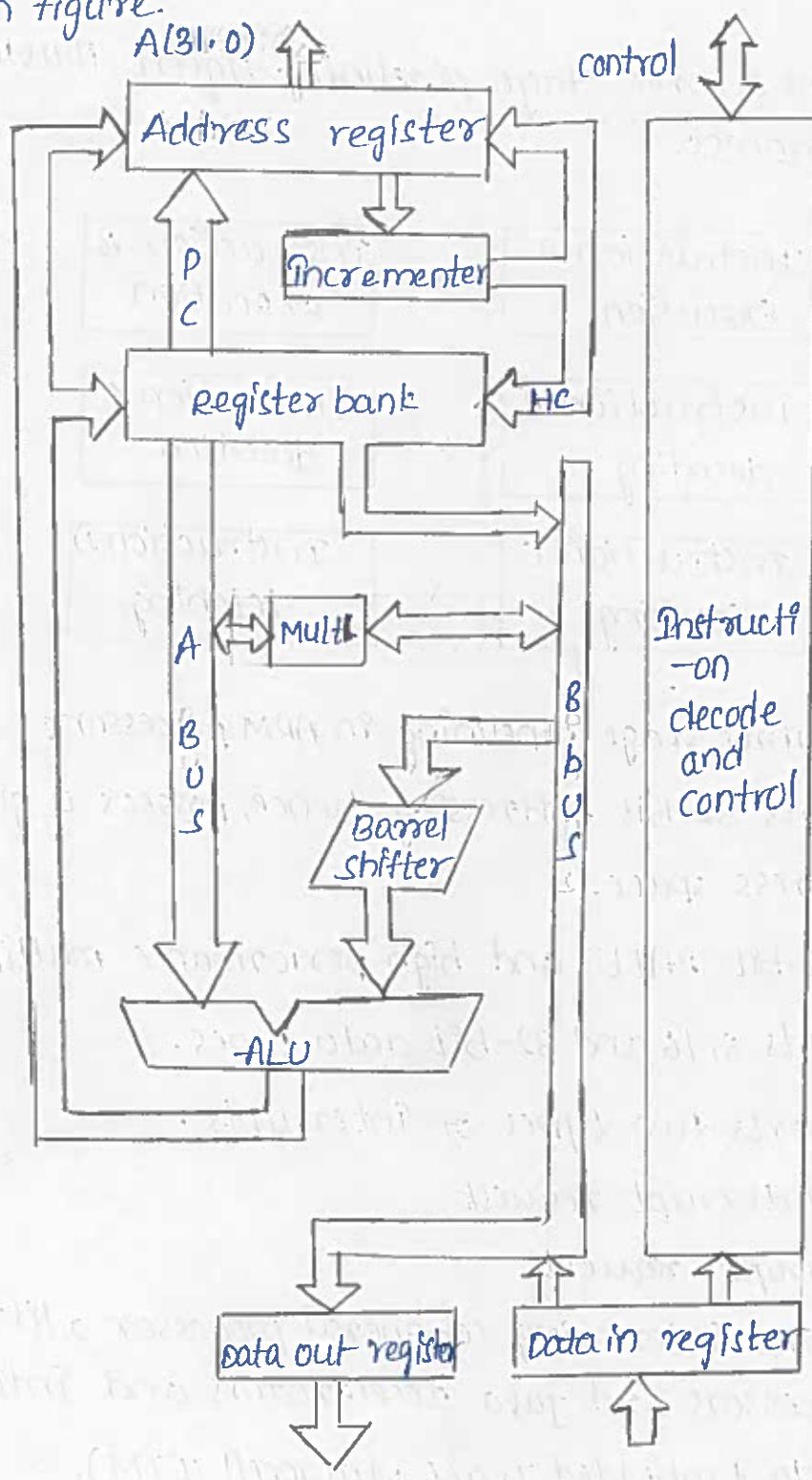
4. It supports 32-bit addressing hence, possess 4 giga bytes linear address space.
5. It has 32-bit RALU and high performance multiplier.
6. It supports 8, 16 and 32-bit data types.
7. It supports two types of interrupts.
 - i) fast interrupt request
 - ii) interrupt request
8. It supports interfacing co-processor, like DSP core processors and java accelerators and interfacing facilities to Embedded Trace Macrocell (ETM).

9. It provides good debug facilities like RT (real-time) debug.
10. ARM processor core architecture is a 16/32-bit RISC architecture.

ARM CORE ARCHITECTURE

ARM (Advanced RISC Machine) is a 32-bit microcontroller. It performs several functions such as addition, multiplication, multiplication with accumulation etc., on 32-bit data. As it facilitates advanced features, it is called used in wide variety of applications.

The simplified architecture of ARM core processor is shown in figure.



Figure

The general architecture of ARM has,

- * Barrel shifter
- * Register bank
- * Incrementer
- * Multiplier
- * ALU

Barrel shifter is one of the most important feature of ARMCore. It checks for the values of the register before participating in operations related to ALU block. This helps in evaluating large expressions and addresses easily.

Register bank of ARM consists of total 32 registers, each of 32-bit wide. These registers also supports 8-bit and 16-bit data by converting them to 32-bit. All these registers are classified as counter register, status registers and general purpose registers.

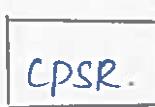
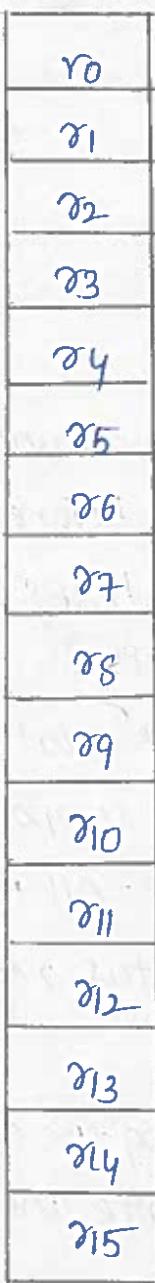
Incrementer is used for updating the address register, whenever the processor reads to write the data in it.

ALU until performs all register-based arithmetic and logical operations.

Registers of ARM

The ARM consists of 37 registers. All registers are 32-bit in size. There are 30 general purpose registers among which 16 (r0 to r15) are active registers and are accessible in the main execution mode i.e., user mode. The current program status register (CPSR) is also an active register located below the r15 register.

From figure (1) depicts the registers of ARM.

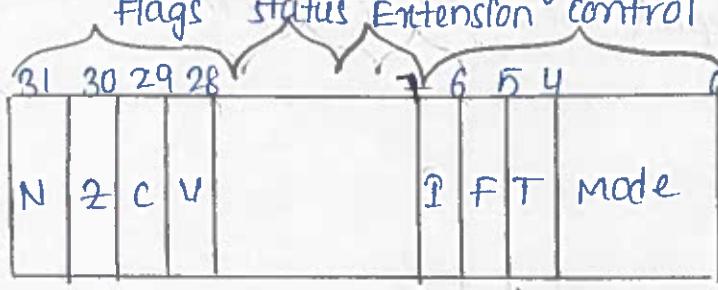


Figure(1) : Registers of ARM

The register R13, R14 and R15 have special functions.

The register R13 serves as stack pointer (SP) register and the registers R14 and R15 serve as Link Register (LR) and program counter (PC) respectively.

The CPSR is a 32-bit register. It is used to execute status of the processor, it operation and interrupt enables bit station. The bit format of CPSR register is shown in figure(2).



Figure(2) : Format of CPSR.

The bit format of CPSR is divided into four fields, namely,

1. Flag
2. Status
3. Extension
4. control.

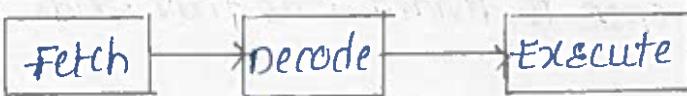
The bits and their corresponding function is depicted in table.

Bits	Name	Function
0-4	processor mode(M)	selects the operating mode
5	Thumb(T)	Decides whether processor is using 32 bit instruction i.e, $T=0 \rightarrow$ process is in ARM state $T=1 \rightarrow$ process is in thumbstate
6	Fast interrupt(F)	$F=1$, disables FIQ Mode
7	Interrupt(I)	$I=1$, disables IRQ Mode
28	overflow(V)	$V=1$, ALU operation overflow
29	carry out(C)	$C=1$, ALU operation carried out
30	zero(Z)	$Z=1$, ALU operation is zero
31	Negative(N)	$N=1$, ALU operation is negative

Pipelining in ARM processors:-

The mechanism of obtaining or getting or fetching the next instruction to be executed, while the other instruction is in the process of execution, is called the "pipelining". The main aim of this concept is to increase or double the program execution speed. The ARM processor also uses this pipelining concept in order to raise its processing speed.

A simple 3-stage pipeline concept is used by ARM 7 and is shown in figure (1).



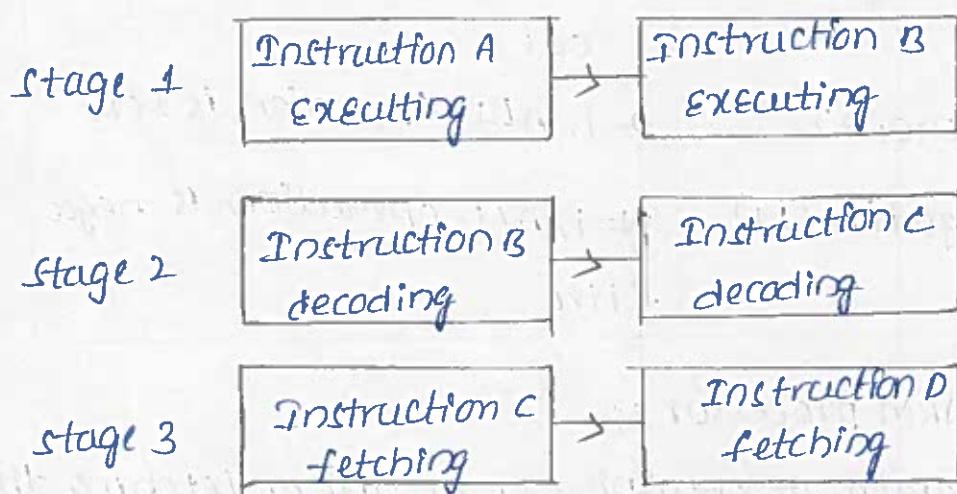
figure(1): 3 stage pipeline of ARM7

Here, the processor performs fetching of instructions from the memory at 'Fetch' stage at decode stage, the processor finds the instruction to be executed and finally it performs processing of instructions in the execute stage.

The processor will store the result in a register for further and future purpose.

By means of combining all the above stated execution speed can be increased which is very much desired for efficient operation of the processor. Each instruction which is very much desired for efficient operation of the processor. Each instruction which is filled in the pipeline consumes single cycle to complete execution, which leads to increase or growth in the throughput of processor.

Figure(2) illustrates 3-stage pipelined instruction execution



figure(2) : Three stage pipelining in ARM processor

From figure (2),

cycle I : The processor fetches instruction-1 from memory.

cycle II : The processor fetches instruction-2 from memory and decodes instruction-1.

cycle III : The processor fetches instruction-3 from memory, decodes instruction-2 and execute instruction-3.

The increase in the length of pipeline, increase the speed of execute and thus the processor performance gets increased. The ARM9 processor uses 5-stage pipeline and as shown in figure (3).



Figure (3): 5-stage pipeline of ARM9 processor

By adopting the above 5-stage pipeline concept, the ARM9 will reach higher core frequency than that of ARM7.

The ARM10 processor uses 6-stage pipeline concept is shown in figure (4).

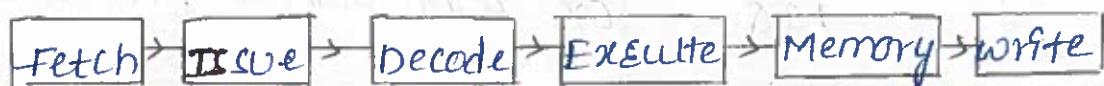


Figure (4): 6-stage pipeline of ARM10 processor

EXCEPTIONS AND INTERRUPTS, INTERRUPT VECTOR TABLE

If an exception or an interrupt occurs in a program, the PC (program counter) is directed to a particular memory address by the processor. The directed address lies in a special range of address known as the "vector table". Vector table consists of a number of instructions (branch of specific num routines) that are used to control a specific exception or interrupt. The address range of vector table is from 0x 0000 0000 to 0x 0000 001C of 32-bit words. But some processors allocate offset higher address range i.e., from 0x ffff 0000 to 0x ffff 0001C.

If an exception or an interrupt arises, the processor stops the current execution and loads instructions from the vector table. The entire or instructions in the vector table are shown in table.

Exception/Interrupt	shorthand	Address	High Address
Reset	RESET	0x 0000 0000	0x ffff 0000
undefined instruction	UNDEF	0x 0000 0004	0x ffff 0004
Software interrupt	SWI	0x 0000 0008	0x ffff 0008
prefetch abort	PABT	0x 0000 000C	0x ffff 000C
Data abort	DABT	0x 0000 0010	0x ffff 0010
Received	-	0x 0000 0014	0x ffff 0014
Interrupt request	IRQ	0x 0000 0018	0x ffff 0018
Fast interrupt request	FIQ	0x 0000 001C	0x ffff 001C

Table

Reset vector

This is the first instruction in the table. It is executed if power is applied and branches to the initialization code.

undefined instruction vector

It is employed if an instruction is not decoded by the processor.

Software Interrupt vector

This instruction is called by the processor when SWI instruction is executed.

prefetch Abort vector

It occurs if an instruction is fetched from an address with incorrect access permissions. Generally, an abort appears in the decode stage.

Data Abort vector

This is same as prefetch abort vector. It occurs if data memory is accessed by an instruction with incorrect access permissions.

Interrupt Request vector

This instruction is employed to interrupt the flow

of current execution by an external hardware. It appears only if IRQs are not masked in the current program status register (CPSR).

Fast Interrupt Request vector

An external hardware uses this instruction if there is a necessity of faster response time. It appears only if FIQs are not masked in the CPSR.

DATA PROCESSING :-

There are seven types of data-transfer instruction and are discussed below,

1. 'Move' Instruction

ARM / THUMB instruction set supports two types of move instructions. They are,

'MOV' - It is used to transfer the content of one register to other.

Example : MVN r_1, r_2

'MVN' - It is used to move the complement content of one register to the other.

Example : MOV r_1, r_2

MVN - Move with logical AND

2. Barrel shifter operations

ARM processor includes a barrel shifter which shifts the 32-bit binary number either in left or right direction to the specified positions. This feature boosts the power and flexibility of various data processing operations.

The syntax of barrel shift operation for data processing instructions is shown in table.

N shift operations	syntax
Immediate	# Immediate
Register	Rm
Logical shift left by immediate	Rm, LSL #shift-imm
Logical shift left by register	Rm, LSL Rs
Logical shift right by immediate	Rm, LSL #shift-9mm
Logical shift right with register	Rm, LSL Rs
Arithmetic shift right by immediate	Rm, ASR #shift-9mm
Arithmetic shift right by register	Rm, ASR Rs
Rotate right by immediate	Rm, ROR#shift-9mm
Rotate right by register	Rm, ROR Rs
rotate right b with extend	Rm, RRX

Table

3. Arithmetic operation Instructions

- i) "ADD, ADC (Addition with carry-flag)" are the two instructions used to perform addition operation
- ii) "SUB, SBC (subtraction with carry-flag)" are used for subtraction operation.
- iii) "RSB (reverse subtraction), RSC (reverse subtraction with carry-flag)" are used for reverse subtraction operation.

4. Barrel shifter with Arithmetic operations

This feature provides a shift of second operand on arithmetic and logical instructions.

Example :

$$r_1 = 0x\ 0000\ 0000$$

$$r_2 = 0x\ 0000\ 0001$$

ADD r₁, r₂, r₂, LSL #1; Adds and left shifts the contents of r₂ and stores the result in r₁.

5. Logical operation Instructions

- i) "EOR" is a logical instruction used to perform NOR between two source operand registers and stores the result in a destination operand register.
- ii) The logical instruction "ORR" is used for a bit-wise logical OR operation between two source operand registers.
- iii) "AND" instruction is used for bit-wise logical AND between two source operand registers.
- iv) "BIC" instruction is used to perform AND NOT operation i.e., logical bit clear.

6. Comparison Instructions

- i) "CMP" instruction is used for operands. It effects C, N, Z and V flags.
- ii) "CMN" instruction is also used for comparison of two source operands but, after performing negation operation on second operand.
- iii) "TST" instruction is used for hypothetical bit-wise AND operation.
- iv) "TEQ" instruction is used for hypothetical bit-wise XOR operation.

7. Multiply Instructions

- i) "MUL" instruction is used to multiply the two operands.
- ii) "MULL" is used to multiply a long instruction.
- iii) "MLA" is the instruction used for multiplication followed by an addition. Similarly, MLAL instruction is used to multiply and add a long instruction.

Example: MLA r1, r2, r3/r4.

The above instruction multiplies r2 and r3 and adds the results with r4 and accumulates the result in r1.

Branch Instructions

The most basic technique employed by ARM to change the flow of control is a "Branch instruction". The mnemonic used for branch instruction is 'B'. This mnemonic is followed by a destination address which gives the location where the control is to be branched and it is called "Branch target".

The address in the branch instruction specifies the offset, which is a value to be added to the current PC value to reach the branch target.

Example : B #250

This instruction multiplies 250 with 4 and result is added to current PC value to reach the branch target. For instance, if the current PC value is 50 then, the branch target is $250 \times 4 + 50 = 1150$. Here, the offset is multiplied with 4, to convert it into bytes because the offset specified in the instruction is in words and ARM is byte addressable.

Load Store Instructions

1. 'Load' Instructions

The instructions which are used to perform load operation are,

- i) LDRB - Load byte instruction
- ii) LDRH - Load Half-word instruction
- iii) LDRSH - Load Half-word signed instruction
- iv) LDR - word Load instruction.

2. 'Store' Instructions

The instructions which are used to perform store operation are,

- i) STRB - store Byte instruction
- ii) STRH - store Half-word instruction
- iii) STRSH - store Half-word signed instruction

iv) STR - word store instruction.

LDM, STM are the instructions used for block data-load and store from or to the memory.

Software Interrupt Instructions (SWI)

A software interrupt instruction creates a software exception to call operating system routines. For calling operating system routines, parameter passing is required. This parameter passing is obtained by employing registers.

Every SWI has a number to indicate a specific call or feature. This number is calculated by a code known as handle from the link register, lr.

Syntax

SWI {<cond>} SWI-number

SWI	software interrupt	lr-svc = address of instruction following the SWI spsr-svc = CPSR pc = vectors to x8 cpsr mode = SVC cpsr I=1 (mask IRQ interrupts)
-----	--------------------	---

Before executing SWI instruction, processor directs program counter (pc) to the offset ox8 in the vector table and processor mode to supervisor mode (i.e., SVC mode).

PSR Instructions

The ARM instruction set employs two instructions namely MRS and MSR to control, read and write into a program status register (PSR).

MRS Instruction: It moves the data from CPSR/SPSR to a register.

MSR Instruction: It moves the data from register to CPSR/SPSR.

Syntax

MRS {<conds>} Rd, <cpsr/spsr>

MSR {<conds>} <cpsr/spsr> -<fields>, Rm

MSR {<conds>} <cpsr/spsr> -<fields>, #immediate

Execution

MRS: moves data from PSR to general-purpose register Rd.

MSR: moves data from a general-purpose register, Rm to PSR.

MSR: moves an immediate value to PSR.

In syntax, label field is a combination of control (C), Extension (X), status (S) and flags (F). The byte regions of these fields in a program status register (PSR) is shown in figure.

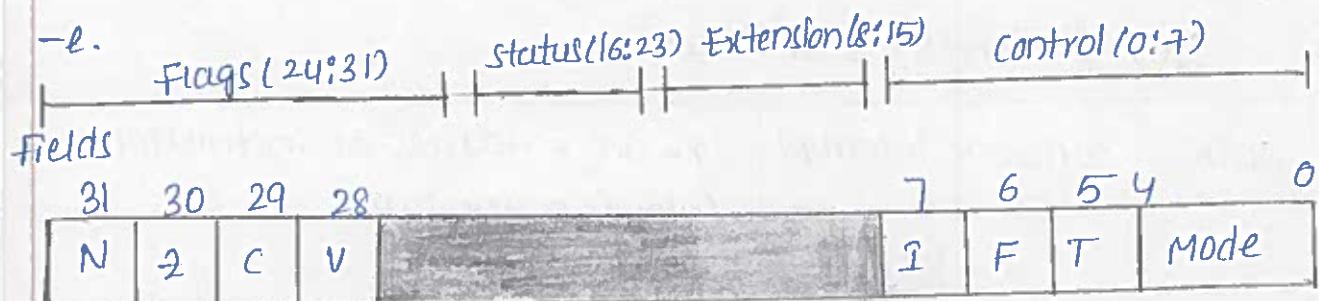


Figure: PSR Byte Fields

control (C) field controls the interrupt mask, & thumb state and processor mode.

Loading constants:

Usually, loading a 32-bit size constant into any register is not possible by using ARM instruction set. So, to achieve this two pseudoinstructions i.e., LDR and ADR are introduced.

Syntax

LDR Rd, =constant

ADD Rd, label

LDR: It loads constant of 32-bit into register, Rd.

ADR: It loads relative address of 32-bit constant into register, Rd.

Loading constants always require a memory access which is very economical. Thus, tools such as compiler and assembler are employed to load constants without memory access. These tools generate a constant in the register with optimal number of instructions. If the tools fail in generating constants then the constants load from memory.

LDR pseudoinstruction conversion is shown in table.

Pseudoinstruction	Actual instruction
LDR r0, =0x ff	MOV r0, #0xff
LDR r0, =0x 5555 5555	LDR r0, [PC, #offset-12]

Table

In the table, the first conversion gives a simple MOV instruction and the second conversion gives a pc-relative load.

In pseudoinstructions, it is possible to determine the handling capability of tools and number of instructions selected by the tools in loading constants.

conditional Execution

Majority of the ARM instruction are executed conditionally. This execution depends upon the status of the condition field and condition flags. Condition field is a two-letter mnemonic placed in the instruction and conditional flags are placed in the current program status register (CPSR).

The default mnemonic "AL" is used for data processing and compare instruction mnemonic "S" is used for changing the status of condition flags in CPSR.

Example

Greatest common divisor algorithm

I. code with partial condition Execution (i.e., condition execution only on the branch instructions)

gcd

```
CMP R1, R2  
BEQ complete  
BLT lessthan  
SUB R1, R1, R2  
B gcd
```

lessthan

```
SUB R2, R2, R1  
B gcd
```

complete

2. code with full condition execution

gcd

```
CMP R1, R2  
SUBGT R1, R1, R2  
SUBLT R2, R2, R1  
BNE gcd
```

Advantages

1. conditional execution maximizes the performance by reducing the number of branches, pipelines flushes and instructions
2. It also maximizes the code density.

Thumb state

The thumb state is one of the state of ARM processor that execute the 16-bit thumb instructions.

Thumb instruction set is developed by Advanced RISC Machines (ARM) for reducing the memory requirements. ARM TTDMI is a chip in which thumb instruction set is included where T stands for 'Thumb'. A typical application is provided with required functionality by the thumb instruction set. The different operations that are supported by thumb instruction set are,

- * Arithmetic and logic operations,

- * Load / store data transfer
- * conditional / unconditional branches.

Most of the programs written in 'C' language can be executed in thumb state successfully depending on the instruction set availability. The register sets that are available (f.e, accessible) in 16-bit thumb state are R₀-R₇, stack pointer (SP), LR, PC (program counter).

Thumb Instruction set

The instruction set in which each instruction is of 16-bit for the 32-bit ARM processor. The instruction can be executed in two ways. They are,

1. By expanding the instructions to the original 32-bit ARM instructions.
2. By employing a 16-bit instruction decoding unit.

The instructions of thumb are half word aligned i.e., at a time it can read two 8-bit memory locations. As a result, the bits 1 to 31 of program counter are important whereas, the lower bit (bit 0) does not hold any significance.

Thumb instruction-set Extensions of ARM controllers

The 'Thumb' is of 16-bit instruction subset and is the subset of widely used 32-bit instructions. The ARM microcontroller has included this 16-bit thumb instruction set. The expansion of run time of the thumb instruction is carried into 32-bit ARM instructions.

Assumptions of Thumb instruction set for programming

1. Lower register set includes registers r₀ to r₇ and are used by many instructions. The higher register set involves registers r₈ to r₁₅. Some of the instructions such as MOV, LDM, PUSH and POP uses registers r₀ to r₁₅ GPRs containing 4-bits per register.

2. some instructions makes use of PC as source or destination for operands.
3. few instructions user r14 as LR (link register) for the return address after BL instruction.
4. Almost all the instruction use r13 as stack pointer in thumb.
5. The CPSR is varied by data processing, conditional branch, and condition control instructions.

The ARM thumb instruction set supports an 8-bit, 16-bit and 32-bit data types and also the memory is half-word aligned in which each offset gets increases or decrease by two addresses.