15

32-Bit ARM7, ARM9 and ARM MCUs: Architecture, Programming and Development Tools

Chapter Outline

- Learn novel 32-bit ARM processor and ARM based MCUs
- Learn ARM® 32-bit instruction set, Thumb® 16-bit set and inter-working between them
- Learn development tools for the ARM MCUs based application systems
- Learn porting of Linux in ARM

15.1 INTRODUCTION TO 16/32 BIT PROCESSORS

Requirement of High- speed Computations as well as Minimum Power Dissipation at the Same Time

A 32-bit processor with an instruction set facilitating a subset of instructions for 16-bit coding is called a 16/32-bit processor. Many embedded systems need precise computing, 16/32-bit processing, high-speed computations as well as at the same time minimum power dissipation and smaller code-size for a given set of computations. MCUs based on 32-bit ARM architecture and 32-bit ARM and 16-bit Thumb® Thumb-2 instruction sets provide these features.

Example of Applications of ARM

ARM is considered important for many applications. ARM finds applications in mobile phones, image processing, video games, robotics and adaptive control systems. Just as most desktop personal computers use the Intel® Pentium microprocessors, most mobile phones (Nokia phones) and pocket PC's use the MCUs based on ARM®. The MCU forms a processing core in the SoC (system-on-chip) used in the mobiles.

15.2 ARM ARCHITECTURE

ARM® stands for Advance RISC Machines (microprocessors), who designed a family of RISC supersalar processor architecture for VLSI implementation. RISC stands for reduced instruction set computer. Figure 15.1(a) shows the basic features of ARM processors.

ARM Instruction set retains the best of CISC (Complex Instruction Set Computer) features. It gives simplicity for programming. The ARM® VLSIs are used widely as core or chip. ARM® is a registered trademark of ARM Limited. These machines offer high computational performance at very low power consumption.

ARM Processor Core

ARM has 32-bit instruction set and supports a 16-bit compact code, called Thumb® instruction set code. The processor functions as a 16/32-bit processor. The ARM7 TDMI ™ embedded processor core is a member of the ARM-7 family of Advanced RISC Machines. The processor uses Thumb™ 16-bit subset extension of ARM7 32-bit instructions.

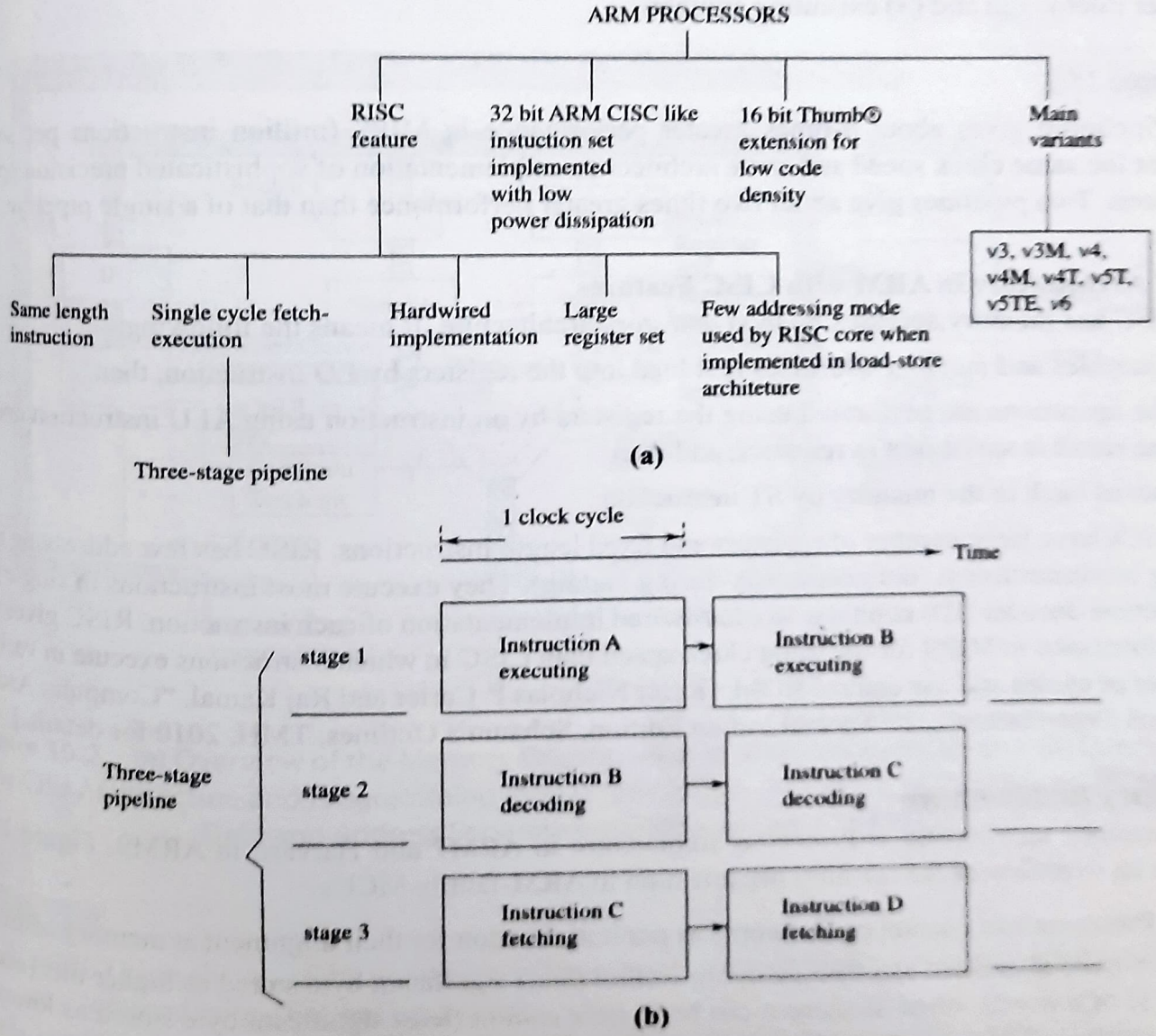

Figure 15.1 (a) ARM Processor Features and Variants of ARM Processor and (b) Features of Pipeline Processing Shown by Three Stage Pipeline in ARM7 Family of Processors

Single Cycle Operations in ARM

ARM performs thirty-two-bit processing for addition, subtraction, multiplication, multiplication with accumulation in the sum in the single cycle in sophisticated precision applications. It performs a single cycle multiplication. It completes division in 2 to 12 clock cycles.

Parallel Pipelines in ARM

ARM is a superscalar processor. It means that it has parallel pipelines. The performance becomes nearly m-times in m-parallel pipelines in the superscalar processor. Instructions are executed in the parallel pipelines. The performance becomes nearly n-times in an n-stage pipeline. Therefore, performance improves nearly $n \times m$ times in the superscalar processor.

Pipelining is facilitated when each instruction fetch is in a single cycle, decoding in single cycle and executing in the single cycle. A pipeline is used for processing such that at an instance more than one instruction is the different stages of the pipeline. Figure 15.1(b) shows a three-stage pipeline as in the ARM7 processor. The stages in a three-stage pipeline are (1) fetching an instruction, (2) decoding another instruction and (3) executing another.

Example 15.1

Pipelining gives about n-times greater performance in MIPS (million instructions per second) for the same clock speed and same architecture implementation of sophisticated precision applications. Two pipelines give about two times greater performance than that of a single pipeline.

RISC Architecture in ARM with CISC Features

An *RISC* has memory accesses by *load and store* architecture. It means the following:

- Variables and memory addresses first load into the registers by LD instruction, then
- the operations are performed using the registers by an instruction using ALU instruction and then the result is saved first in registers, and then
- stored back in the memory by ST instruction.

RISCs have large number of registers and fixed length instructions. RISC has few addressing modes during implementation (not necessarily during coding). They execute most instructions in single cycle. Instruction decoder (ID) connects to a hardwired implementation of each instruction. RISC gives greater performance in MIPS for the same clock speed than CISC in which instructions execute in variable a number of cycles and use control ROM. [Refer Nicholas P. Carter and Raj Kamal, "Computer Architecture and Organisation", 2nd Special Indian Edition, Schaum's Outlines, TMH, 2010 for details.]

Memory Architecture

The memory architecture is Princeton architecture in ARM7 and Harvard in ARM9. Figure 15.2(a) shows an overview of the memory organization in ARM-family MCUs.

1. Processor can operate on the words as per initialization for their alignment at memory addresses.
2. A word alignment can then be in *big endian* (least significant byte stored as higher bits (address 3) of a word). Word alignment can be in *little endian* (least significant byte stored as lower bits (address 0 of a word). Address 0 is an even address and the address 4 means the address of word that is next to next even address. Address 3 means address before the next to next even address (next to next odd address).

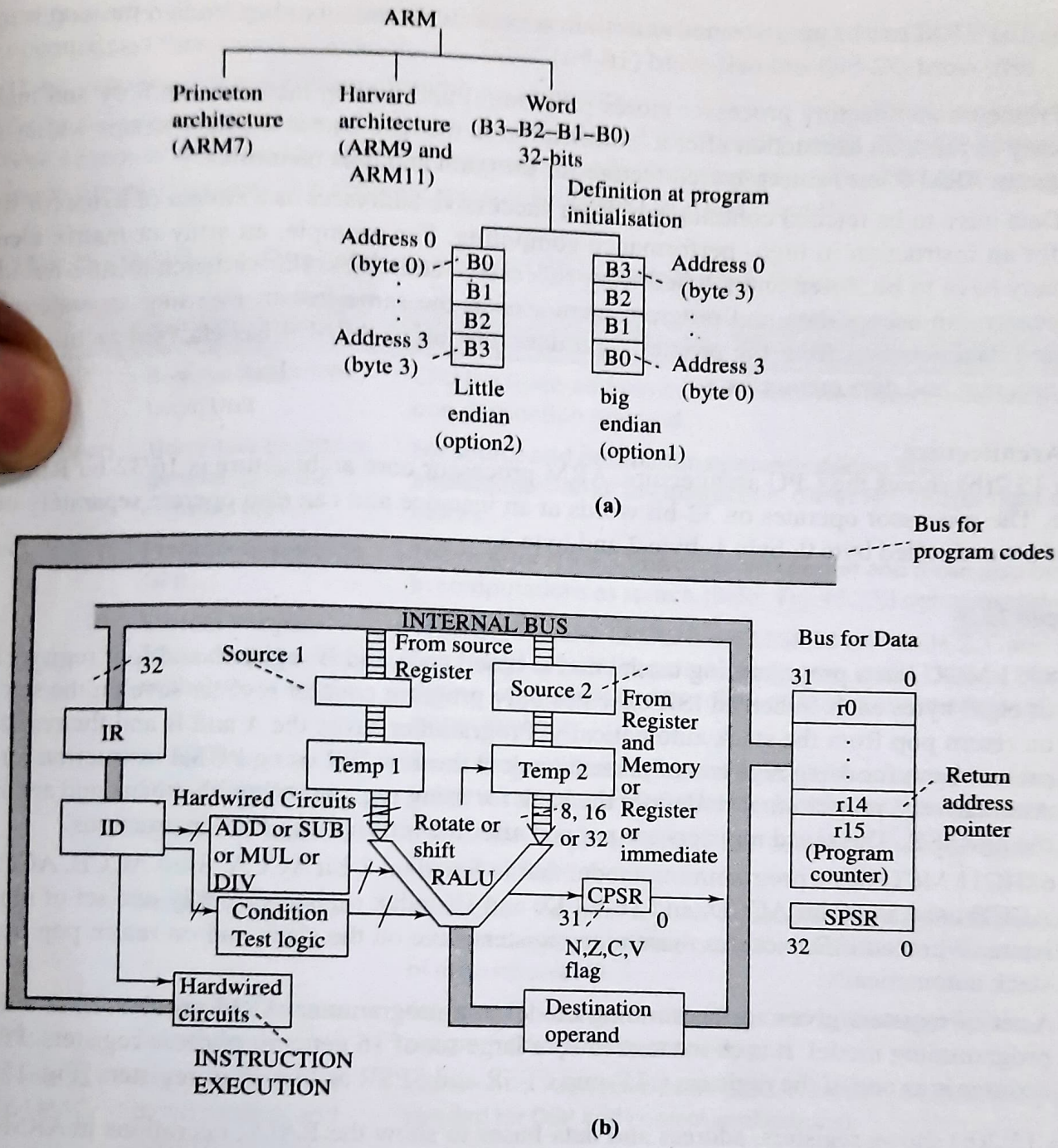

Figure 15.2 (a) Overview of the Memory Organization in ARM Family MCUs and (b) Overview of the CPU Architecture and Programming Model of Registers, Address and Data Buses. [Buses for Data and Address Separate in ARM9 and Same in ARM7]

Example 15.2

1. 68HC11 has always 16-bit word byte 0 (lower byte) at a higher address and byte 1 (higher byte) at a lower address. [Big-endian mode.]
2. The 8051 has always word alignment in the little endian mode—16-bit word byte 0 at a lower address and byte 1 at a higher address and 80196 always has little endian mode for double word (32-bit) and word (16-bit).

3. The ARM can be programmed at initialization as little endian or big -endian for long word (64-bit), word (32-bit) and half-word (16-bit).

3. Princeton architecture processor stores program(s) and data in the same memory and makes it easy to reuse an instruction after a modification in the data part at the instructions within a program. ARM 7 has Princeton architecture for program and data memories.
4. Data have to be fetched continuously from successive addresses as a stream of bytes (or words) for an instruction in high- performance computing. For example, an array or matrix elements may have to be stored and fetched from successive addresses. Harvard architecture-based processor can access data and program memories at the same instant by using separate address and data accesses from the program and data memories. ARM 9 has Harvard architecture for program and data memories.

CPU Architecture

Figure 15.2(b) shows the CPU architecture. ARM processor core architecture is 16/32-bit RISC architecture. The processor operates on 32-bit words at an instance and can also operate separately on each byte of a word called byte 0, byte 1, byte 2 and byte 3.

Example 15.3

1. 8051 MCU has a programming model that is based on A and B registers and four register banks of eight bytes each. When an ISR executes only program counter register save on the stack and on return pop from the stack automatically. Programmer saves the A and B and the registers of present bank for using registers of present bank in the new ISR using PUSH instruction for each. Alternatively, programmer re-defines the bank for using registers of another bank and are used in the new ISR. The saved registers are popped after the return using POP instructions.
2. 68HC11 MCU has a programming model that is based on 8-bit ACCA, 8-bit ACCB, ACCA and ACCB used as 16-bit ACCD, and 16-bit IX and IY index registers in only one set of nine registers. When an ISR executes these nine registers save on the stack and on return pop from the stack automatically.
3. A set of registers gives a programming model to a programmer. ARM processor has a distinct programming model. It is characterized by a large set of 16 general- purpose registers. Program counter is as one of the registers (r15) and CPSR and SPSR as two other registers [Fig. 15.2(b)].

Figure 15.2(b) shows registers, address and data buses to show the RALU operations in ARM-family MCUs [RALU means *ALU with operations on registers* only]. Table 15.1 gives the structural units and their uses in ARM architecture. Following are the model features:

1. ARM7 and ARM9 support 300 MIPS when the die size is 0.13 μm. CPU supports sixteen 32-bit registers, r0 to r15. These form a general- purpose register (GPR) set.
2. ALU operations are the register-based arithmetic and logic unit (RALU) operations.
3. RALU does 32-bit add, subtract, reverse subtract, multiplying and multiplying with accumulate operations. It has a barrel shifter (multi-bits left or right shift or rotate).
4. A GPR can function for computations as well as for index register for indirect addressing to memory in an instruction.
5. 32/16-bit ARM instruction set feature is CISC with load and store architecture. Implementation is RISC. No RALU operations take place through a direct reference to the memory but through

indirect reference. Operand in load and store architecture must first be loaded in CPU registers, operated and then saved to memory.

6. 32-bit memory space is continuous with no segments.
7. There are two modes of the processor—user mode and supervisory mode. The OS (supervisory SVC) mode is facilitated by the SWI instruction (software interrupt instruction). For supervisory mode functioning there is a separate register set (bank) in ARM7 and 9.

TABLE 15.1 The Meaning and Uses of the Structural Units Shown in Block Diagram

Term	Full form	Use
RALU	Register Arithmetic Logic Unit	CPU is a three-address machine with two source operands and one destination operand. [a]
r0 to r14 fifteen registers	Thirty-two bit fifteen general- purpose registers (GPRs)	For source and destination operands during RISC implementation of the instruction. (Refer Fig. 15.2(b) right side blocks.)
r15 (PC)	Program Counter as GPR	The register r15 is 32-bit program counter and it can also be used in computations as source. (Refer Fig. 15.2(b) rightmost side.)
CPSR	Current Program Status Register for each Process Mode	CPSR.31, CPSR.30, CPSR.29 and CPSR.28 bits are N, Z, C and V flags (flags set on a negative result, zero result, carry or borrow during operations and overflow on arithmetic result overflow, respectively).
SPSR	Saved Program Status Register	Saves Program Status Register from CPSR on branch and link (routine call) and SPSR can be stacked for each processor mode.
IR	Instruction Register	Instruction Register to hold the current instruction opcode from dour byte queue.
ID	Instruction Decoder	Instruction Decoder register to decode the IR opcode bits and activate the appropriate signals for hardwired implementation of instruction at IR.
Condition Test Branch Logic	Condition Test and Branch Logic	Additional circuit to test the flags and initiate appropriate program flow path by branching.
32 x 32 and 32 x 32 + 64 MAC	32-bit multiply and 32-bit multiply and accumulate (MAC)	Variant M of ARM have these units. (32 x 32 + 64 MAC units are needed for DSP and control applications).

[a] A source operand (one RALU input) in RALU operations is first loaded, can also be first operated with rotate or shift instructions, can be either from register or register + memory or an 8-bit immediate operand. [A register indirectly indexes the word at the memory.] Another RALU input is from a source operand from a register. Result of RALU returns to a destination register. [Refer Fig. 15.2(b) middle for RALU and common bus for sources and destination from the general- purpose register set r0 to r15.]

Context Switching

Real-time interrupt response is fast because of provision for fast context switching. The fast context switching is present in ARM because of the sets of large registers. A new set (or bank) is used by called routine. Only the set or subset pointer needs to be used when switching to a routine. ARM has large register sets.

Example 15.4

Assume that 8 registers r0–r7 and a status register are being used by a routine. The routine is interrupted and an ISR starts; now if the ISR uses another fresh set (or bank) of r0–r7 and status register, then context switching is fast. This is because there is no need to save previous routine parameters in r0–r7 and status register.

Interrupt Vectors

Interrupt vectors start from memory address 0 in ARM7. Four-byte address of each interrupt vector is stored there. Interrupt vector table starts from address 0 in ARM7 and ARM9. Interrupt vector table can be moved to another address using a control register in Cortex M3.

Interrupt Service

Interrupt servicing by IHR (interrupt handling routine) takes place after saving of return address in a register (r14), SPSR saves the CPSR bits, the CPSR bits to show which interrupt servicing taking place and PC becomes equal to the address at the interrupt vector. IHR restores CPSR from SPSR and clears the interrupt-disable flags and restores PC using instructions when return takes place to the foreground program.

Example 15.5

ARM7 follows banked shadow register exception (interrupt) model. It means that in supervisory mode, a separate bank is allocated for the supervisory mode programs r0–r7, current and saved program status register (CPSR and SPSR), link register (r14) and program counter (r15). The SPSRs are stacked if new ISR calls another routine. The SPSR returns to CPSR when there is return from the ISR.

ARM Variants in architecture

ARM has the following variants of architectures:

(a) *v1* – ARM1, *v2* – ARM2, *v2as* – ARM3, and ARM250, *v3* – full 32-bit addressing for both data and code ARM6, ARM7, ARM8 and Amulet 1, *v3M*–ARM6, ARM7 and ARM8, *v4*–StrongARM, ARM9, *v5*–ARM10, VFP1–ARM10, *v6*–ARM11 some variants for high floating point performance (single-precision and double-precision floating point arithmetic) on typical algorithms used in graphics and DSP.

(b) Thumb (T variants), which uses 16-bit Thumb® subset of instructions for high code density. It loses the conditional instruction execution part of the opcode in the instructions. It can address the first eight registers of the processor only. Thumb-2 version gives mixed mode capability. It means CPU can inter-work between Thumb and ARM modes.

(c) Long multiply instructions (M variants), four extra instructions that perform 32×32: 64 bit multiplication result when multiply instruction executes. 32×32 + 64:64 bit instruction means that the result (add multiplication result with 64-bit) when a multiply-accumulate instruction executes.

(d) Enhanced DSP instructions (E variants). [32×32: 64 operation means a 32-bit register value multiplies with another 32-bit register value to get 64-bit multiplication.]

(e) Jazelle version ARM architecture executes Java codes fast due to Java accelerator core as internal coprocessor.

(f) ARM 11 core used in ARM1176JZ(F)-S has an 8-stage pipeline and deploys ARMv6KZ architecture. It has Thumb, Jazelle DBX, (VFP) and Enhanced DSP instructions. It uses a variable

amount of caches. It uses MMU and TrustZone memory protection unit. It finds applications in Apple iPhone. Samsung's ARM11 versions are S3C6410 and S3C6430.

15.3 ARM7

ARM 7 TDMI™ architecture has the following features:

1. Uses 0.25-µm and less die-size. Very small die size facilitates low voltage operations, and low power consumption. (A very small power consumption measure is MIPS/Watt.)
2. Gives high performance of 300 MIPS at die size = 0.13 µm.
3. Uses HCMOS technology. Has fully static operations, which means MCU clock can be reduced to 0 since it is fully complementary MOSFET based.
4. There is a large register set consisting of sixteen 32-bit registers.
5. There is a three-stage pipeline (pipelining means processing, memory and other systems can operate continuously) (Fig. 15.1(b)). (A Pentium has a five-stage pipeline.)
6. There are 2^{32} addresses for 4G bytes linear address space.
7. There is a 32/16-bit RISC architecture (called ARM v4T), which has 32-bit ARM instruction set and a 16-bit Thumb instruction subset as extension.
8. Thirty two-bit RALU and high-performance multiplier.
9. There is a common internal bus for the address and data of thirty-two bit in ARM7. Thirty-two bit Princeton architecture bus and bus interface for the 32-bit data and instructions (ARM9 has Harvard architecture).
10. Instructions process data with 8, 16 and 32-bit data types (byte, half-word and word).
11. There are two types of requests for interrupts—Fast interrupt request (FIQ) and interrupt request (IRQ). Here, fast means high priority.
12. Banked shadow registers are present. When the system interrupts to supervisory mode, a separate bank is allocated for the supervisory mode programs r0–r7, current and saved program status register (CPSR and SPSR), link register (r14) and program counter (r15),
13. Coprocessor interface exists to connect coprocessors, like DSP processors and Java accelerators.
14. The ARM memory interface has the speed-critical control signals, which are pipelined (control signals implement system control functions with standard low-power logic and facilitate fast local access modes by using the DRAMs).
15. Extensive debug facilities [like embedded ICE (in circuit emulator), RT (real-time) debug and on-chip JTAG interface units exist.] JTAG provides direct high-speed access to the MCUs internal units (registers and control unit).
16. Interface for direct connection to embedded trace macrocell (ETM).
17. It supports Thumb instructions for 16-bit operations and for enhancing code density.

15.4 ARM9

ARM9 has Harvard architecture. ARM9 has the separate instruction and data cache memory. It has separate instruction and data buses. It has a five- stage pipeline compared to three in ARM7. It

provides faster load and store due to more stages in pipeline. ARM9 gives higher performance than ARM7.

ARM9 supports the Thumb instructions like ARM7. ARM 9 T, E and J versions support. ARM 926 has 16 kB instruction cache and 16 kB data cache. ARM 940 T has memory protection unit. ARM supports Jazelle DBX (Direct Byte-code eXecution). The byte code in Java execute directly.

15.5 ARM-BASED MCUS

ARM Core-based MCUs

ARM embedded microcontrollers (MCUs) are single-chip microcomputers with the ARM architecture at CPU. ARM core-based MCUs based on ARM7 and ARM9 are widely accepted for applications needing high speed and performance in MIPS per Watt. The instructions use the byte, half-word (16-bit) and word (32-bit) for operations.

Example 15.6

AT91ASM9263EJ-S™ (E means DSP instructions extended instructions, J mean Jazelle Java accelerator version. It executes ARM/Thumb instructions. It has 16 kB plus 16 kB instruction an data caches. It has MMU (memory management unit). It gives performance of 220 MIPS for 200 MHz clock. It has embedded ICE, embedded trace macro-cell (ETM) and nine 32-bit bus-matrix. 28.8 Gbps is total on-chip bus bandwidth. It has 128 kB internal ROM and 80 kB SRAM. It has single cycle access and maximum bus matrix speed. It has DMAC, LCD controller, system controller, advanced interrupt controller, debug unit, two real- time timers, USB2.0, programmable frame capture, image sensor interface, two graphic accelerators, two synchronous serial control channels and three channel 16-bit timers. It operates at 1.08 V to 1.32 V. The peripherals derive at 2.7 V to 3.6 V. Further details are at Atmel Web site http://www.atmel.com/products/AT91/.

There are a number of ARM- based MCUs. Table 15.2 lists the ARM7TDMI ™ architecture feature MCUs.

Analog Devices ARM- based MCU is ADuC7026. ARM Cortex M3 based MCU is ARM Cortex-M1 from Altera. Atmel (http://www.atmel.com/products/AT91/) manufactures AT91 single chip controller, which has ARM9 as core in AT91RM3400. AT91 industrial controllers are AT91SAM7A11x CAN and AT91SAM7A24x CAN. [CAN stands for standard bus called Controller Area Network bus. The CAN is widely used in industry and in automobiles for networking the embedded controllers.] Refer http://www.e-lab.de/ARM7/AT91SAM7A2.pdf for a description of architecture.

An Atmel MCU is Atmel AT91SAM7S128. AT91RM3400 MCU from Atmel has ARM9 architecture plus SRAM + boot ROM, loads from an external SPI data flash. ARM9 with advanced RTOS compatibility like in STR720 is in AT91RM9200. It has 16 kB SRAM, 128 kB preprogrammed (AT91 boot ROM, SDRAM controller, JTAG interface, debugger, four UARTs, I^2C master, SPI Master/Slave, USB Host and USB/Client, Ethernet, 3 x SSC (Synchronous Serial Controller), 6x timers, Compact Flash, Smartmedia MMC Card, SD Card and Burst Flash Controller.

A manufacturer of ARM7TDMI™ based MCUs is ST Microelectronics (referred as ST in the following descriptions in this Chapter). (http://st.com and http://www.arm.com/news/5135.html). ST manufacturers ARM based MCU chips (and cores), it integrates ARM7 TDMI ™ in STR7x (STR710, STR711, STR712) and STR720 derivatives, which are the high- end 32-bit single-VLSI MCUs.

An ARM9-based MCU from ST Microelectronics is ST STR912FW44. STR 10 and STR20 MCUs are also listed. Figure 15.3 shows 16/32-bit ARM-based flash STR710 microcontrollers with USB and CAN.

The ARM-based low cost and high performance Philips MCUs are low power LPC2114 and LPC2124. Philips MCUs are now made by Nexperia. Their architectural features and devices are also listed in Table 15.2. Another MCU is from Nexperia (Philips) LPC2148.

An ARM-based MCU S3C2410X01 is manufactured by Samsung. [http://www.pocket pccentral.net/ help/samsung2410.htm]. It is widely used in PocketPC. [PocketPC means a PC that can be kept in a pocket and has mobile applications.] The S3C2410X01 is developed using a low-power, simple, elegant and fully static design. It is particularly suitable for cost-sensitive and power-sensitive applications. Also, S3C2410X01 adopts a new bus architecture. An outstanding feature of the S3C2410X01 is its CPU core, which is a 16/32-bit ARM920T RISC. (The ARM920T implements MMU, AMBA BUS and Harvard cache architecture with separate 16 kB instruction and 16 kB data caches, each with an eight-word line length.) By providing a complete set of common system peripherals, the S3C2410X01 minimizes the overall system costs and eliminates the need to configure additional components. A Samsung S3 based MCU is S3C44B0X. Table 15.2 lists the MCU features.

ARM MCUs are also available from Freescale and Texas Instruments. These are MAC7101 and TI TMS470R1B1M, respectively.

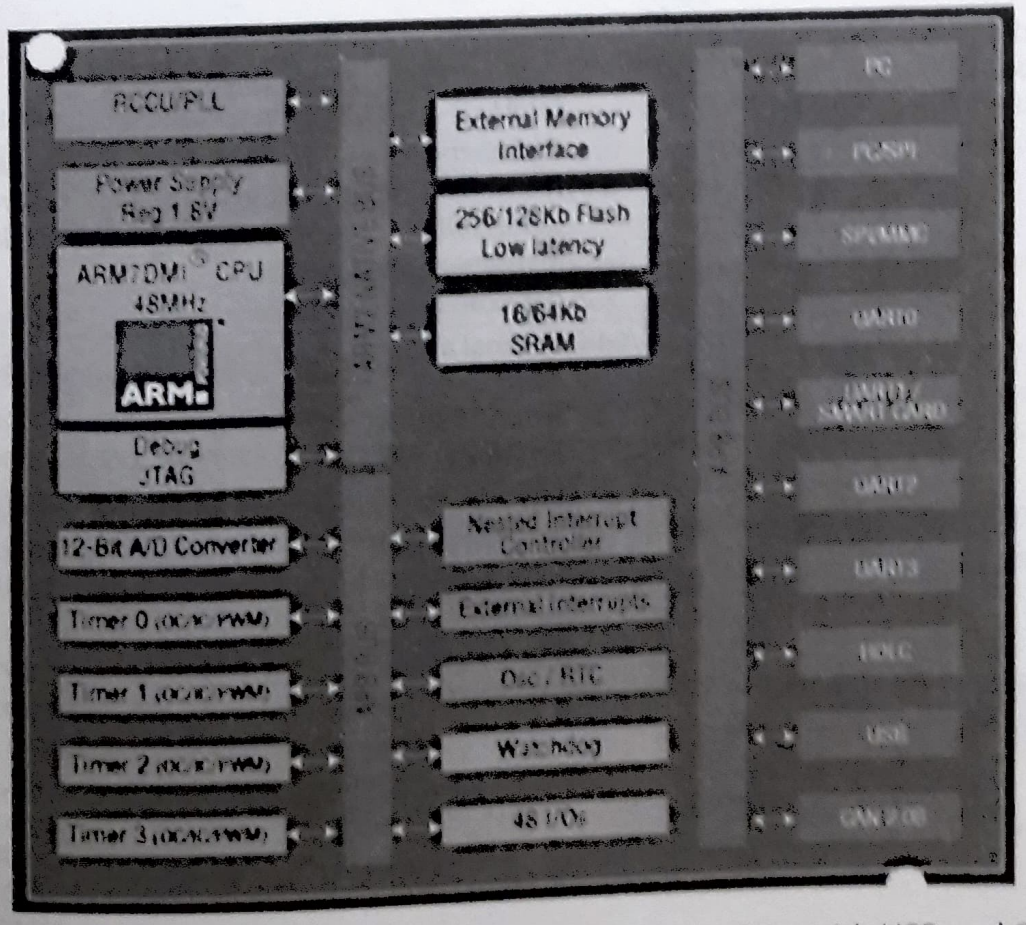

Figure 15.3 STR710 16/32 Bit ARM-based Flash Microcontrollers with USB and CAN (*Source* http:// www.st.com/stonline/products/support/micro/arm/str7/10-htm.htm with Courtesy and Permission of ST Microelectronics)

TABLE 15.2 ARM7TDMI™ Architecture Features along with Additional Features in STR 10, STR 20, LPC21x4 and S3C2410x01

Processor and Devices	Features of Architecture	Additional Features in STR 710[a]	Additional Features in STR 720[a]	Additional Features in LPC2114 and LPC2124	Additional Features in Samsung S3C2410X01
ARM7TDMI™ or ARM9 MCU	Refer ARM features list in Sections 15.3 to 15.5	ARM7TDMI™ MCU [Example STR10F version 48 MHz]	ARM7TDMI™ MCU	0.18-µm CMOS Embedded 60 MHz	ARM9TDMI™ ARM920T[b] core 0.18 µm standard cells
Flash		Internal 128 K/256 K embedded fast flash (a) up to 30 MHz with 0 wait states (without acceleration) (b) Up to 48 MHz with 0 wait states (with acceleration)	Additional external memory interface for flash with internal 128 K/256 K embedded fast flash same as STR 710	Flash process enabling ultra-low 1.8 V voltage operation and 60 MHz operations. Up to 256 K bytes of embedded flash memory[c]	Flash boot loader (OTP area in flash for system initialization programs)
MMU[d]		–	MMU		MMU
Cache		–	8 kB cache [combined write buffer, instruction and data caches[e]]		Separate 16 kB Instruction and 16 kB data cache
SRAM		Internal 16 /64K high-speed	Internal and additional external SRAM interface	16 K of SRAM	
SDRAM		–	SDRAM interface[f] with 128 MB address space	–	System manager (chip select logic, SDRAM controller)
Low power modes (slow, idle, stop and power-off)	Yes	Yes	Yes		Yes
Basic communication interfaces	Five SPI, UART and I2C	Four SPI, UART and I²C	UARTs, SPIs		2-channels SPI (IrDA1.0, 16- byte FIFO), 3-channel UART with handshake[g]

(Continued)

art	Yes	No	No		No
e		Yes	–		–
AN	Yes				
SB	Yes	Yes	–		2-port USB host /1- port USB device (ver 1.1)
HDLC	Yes				
MMC	Yes	–			Yes, protocol version 2.11 compatible
dia rface		–	–		4-channels DMA
n	–				
request					
troller	–		–		STN and TFT
	–	–	10-bit A/D converters	8-channels 10-bit ADC	
		–	–	Yes	4-ch PWM timers
		–	–	Yes	1-ch internal timer[h]
ge options		(a) Small, low pin-count TQFP64 package and (b) TQFP144 package with external memory bus	(a) Small, low pin-count TQFP64 package and (b) TQFP144 package with external memory bus	46 General- purpose input/outputs in a small outline 64-pin package LQFP64 package[i]	117-bit general-purpose I/O ports/24-ch external interrupt source

STR710 and STR 720 derivatives.
arvard Architecture, a memory complier, external I/O microprocessor, AMBA (advanced microcontroller
s architecture) MCU, PLL for clock generation, 1.8 V internal, 1.8 V/2.5 V/3.3 V memory, 3.3 V.
ighest performance embedded flash memory in 128-bit wide zero wait-state flash.
MU (memory management unit) allows running of the RTOS such as Linux, WIN-CE and QNX.
ombined data and instruction caches and write buffer increase the execution speed and reduces memory
andwidth.
ombined SDRAM and Cache for higher optimum performance.
C (1-ch multi-master I²C-BU S/1-ch I²S-BUS controller), Touch screen interface, I²C BUS interface, SD 1.0 Host.
Additionally on-chip generator with PLL, watchdog timer and an RTC with calendar function.
Also the HVQFN64 9 mm x 9 mm packaging.

15.6 ARM CORTEX-M3

ortex-M3 processor has ARMv7-M architecture in addition to the following:

1. It has a small core footprint. [Footprint means number of gates used in the core for similar instruction set of the core.]

2. It gives high code density. It thus requires smaller memory for the code. It has small pin count.
3. It has low power consumption.
4. Harvard memory architecture.
5. Excellent integration of the peripherals/SRAM through memory protection unit.
6. Three-stage pipeline with branch prediction.
7. Single cycle multiply.
8. Two to twelve cycle signed/unsigned divide.
9. Unaligned data storage programmability.
10. Atomic bit manipulation. These instructions are required for wireless networking and serial transmissions of the bits. [Atomic means interrupt only after completing the bit manipulation.]
11. Supports sleep mode.
12. 1.25 Dhrystone MIPS per MHz clock cycle,
13. Thumb®-2 instruction set architecture.
14. Debug port configurable by serial wire or JTAG.
15. DWT [data watch points (break points) and trace].
16. FPB (flash path and break point unit). It has six program breakpoints and two data-fetch break points.
17. ETM (embedded trace macrocell) interface. It is used for tracing the real-time instructions.
18. Exceptional system response mechanism for the interrupts. This is because it deploys a NVIC (nested vectored interrupt controller). The NVIC supports nesting (stacking) of interrupts. Therefore, an interrupt of low priority can be preempted by exerting higher priority. It also supports dynamic re-priority allocations. Interrupts that are being serviced are blocked from further activation until the interrupt service is completed.
19. Interrupt latency is 12 cycles in contrast to ARM 7, which has 24 to 42 cycles of interrupt latency. There is excellent deterministic interrupt behaviour.
20. NVIC also integrates with the timer for the system ticks. It is a 24-bit count-down timer. It generates interrupts at regular time intervals. It thus gives system clock to run RTOS and timer for scheduling the tasks.

Details are in data sheets at http://www.arm.com/pdfs/Cortex_M3_DS.pdf. Applications are use as the cores for application microcontrollers, wireless networking, automobile and industrial control systems.

Configurable NVIC (Nested Vectored Interrupt Controller)

ARM 7 has two interrupts FIRQ/IRQ. Cortex M3 controls one NMI (Non-Maskable Interrupt) and 32 general-purpose physical interrupts using NVIC. There can be 8 levels of priority for preempting the low priority tasks by the higher priority tasks. It can be configurable between 1 and 240 physical interrupts with up to 256 levels of priority. Interrupt Vector table (IVT) locates at 0 address at the processor reset. IVT is re-located using a control register to another address. IVT is used to get the ISR address. When an interrupt takes place, r0–r3, r12, r15 (program counter), r14 (link register) and status register (CPSR) are pushed on to the stack automatically using the data bus. When there is return automatically these are retrieved. This is similar to 9 registers push and pop on the stack in 68HC11. The instruction bus retrieves the address from vector table at the same time slot. Thus, interrupt handling is fast. Inter-interrupt latency is six cycles, which are required for instruction fetch for the ISR. The interrupts can be tail-chained in case of back-to-back interrupts from same priority levels.

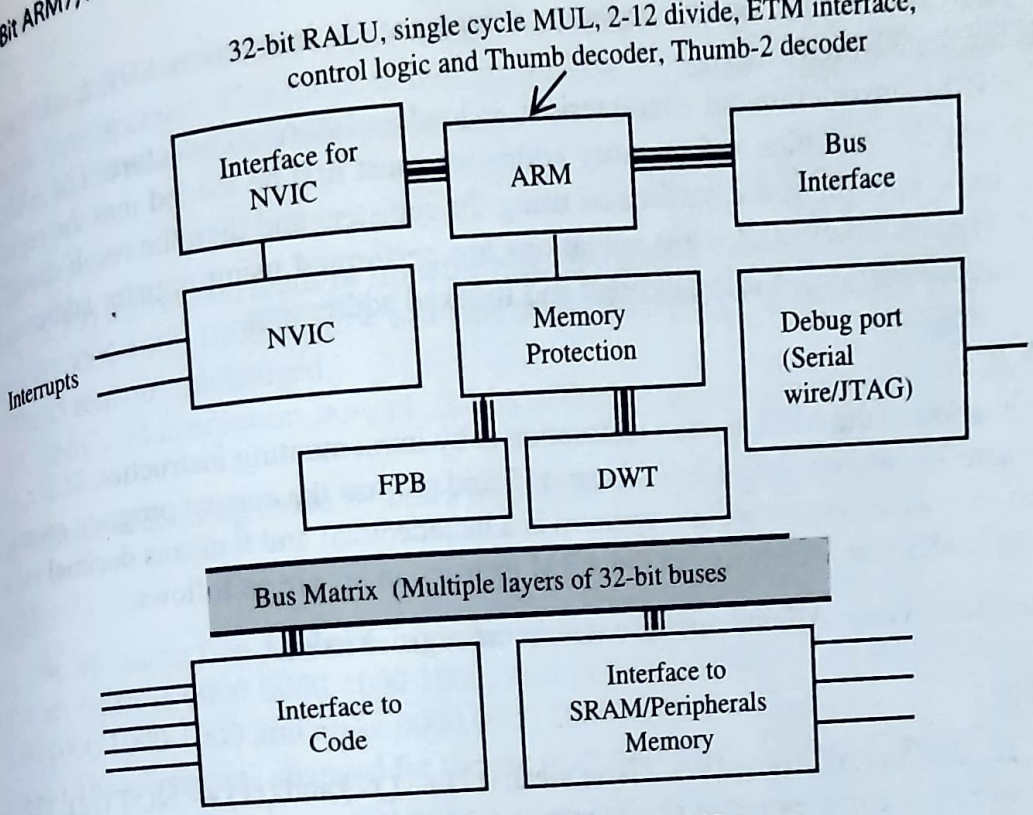

Figure 15.4 ARM Cortex M3

Cortex-A9 gives performance of 2.5 DMIPS/MHz. A9 has the VFP, Jazelle, DBX, Thumb-2 versions. It issues out-of-order instruction in the superscalar pipelines. [Out-of-order issue means that the instructions which don't need result of previous instruction are inserted first in the pipeline stages.] ARM Cortex cores support Jazelle RCT (Runtime Compilation Target). RCT gives the efficient just-in-time (JIT) compilation and ahead-of-time (AOT) compilation with Java and other execution environments.

15.7 ARM INSTRUCTION SET

The ARM architecture is such that the 32-bit ARM instruction set and 16-bit Thumb® instruction subset both can be used. The instruction set extends to provide support for Java acceleration (Jazelle™). The instruction set also extends to provide support for security-coding features (TrustZone™).

15.7.1 32/16-bit ARM Instruction Set

The following are considered in an instruction-set:

a. Instruction types
b. Number of operands in the instruction
c. Whether Instructions are of fixed length or variable lengths and
d. Addressing modes.

The ARM has a large number of registers and instructions of fixed length. Each instruction is of 32-bit in the ARM instruction set. There are 16 registers, r0 to r 15. r15 functions as PC.

An instruction has op-code bits and operand bits. Absolute addressing means use of 32-bit address in the instruction operand. Therefore, absolute-(extended) addressing and direct-addressing instructions are *not present* in the ARM as the length of each instruction is 32-bit. Absolute address can be computed in a register by arithmetic addition or subtraction using the PC address from r15 and *offset* (as immediate operand) between current instruction address and desired absolute address.

Absolute address computation at a register *r* is by a pseudo instruction ADR r, *addr*, where *addr* are lower 12-bit for the 32-bit address.

One of the RISC instruction-set characteristic is load and store architecture. The ARM instructions are also such that the variables and memory addresses must first be loaded into the registers and then arithmetic or logic operations are performed using the registers and then the result should be stored in the memory. The arithmetic and logic operations are performed using register addressing. Load and store operations are performed using indirect and indexed addressing.

Example 15.7

Implementation of the ADR pseudo instruction is by implementing instruction *SUB r, r15, #&byte_offset,* where 12-bit *byte_offset* is *Address*– r15 and r15 has the current program counter at a 32-bit address. (#& means the immediate operand is a hexadecimal and # means decimal value.)

Addressing modes that are present in the ARM instruction set are as follows.

(1) *Register Addressing*: All operands are within the register itself.

Example 15.8

MOV r1, r2. *MVN* r1, r2 instruction. These mean (r1) ← (r2) and (r1) ← NOT (r2). The second one means move r2 or complement of r2 bits into r1 (r2 bits do not change). (Note that logic operation NOT as well as MOV is combined instruction, MVN).

Example 15.9

ADD r1, r2, r3 add the source operands r2 and r3 bits and execution result is at the destination-operand r1.

(2) *Immediate Addressing*: Second source operand can be an immediate operand. Immediate operand is of fewer-bits. For example, 8 bits in case of an ADDGT instruction, which adds if the conditional test is successful, 12-bits in case of an ADD with no condition tests. ADD r6, r7, #8 adds r7 and 8 and the result is in r6.

(3) *Indirect Base and Indexed Addressing*: It is the major mode for load–store architecture as direct absolute addressing is not provided. The following are the types of indirect addressing modes. In all the types the assumption is that all 16 msbs are 0.

a. A byte or half-word or word address is pointed by the *indirect address* in a GPR.

Example 15.10

LDR r1, [r2] instruction loads r1 from the memory address pointed by r2. If r2 is 1000 1000 1000 1000, then r1 will load the 32 bits from memory address 1000 1000 1000 1000. The r2 and r3 remain unchanged.

b. A byte or half-word or word address is pointed by the *indirect address obtained by subtracting two registers* of GPR set.

Example 15.11

LDR r1, (r2, – r3) instruction loads r1 from the memory address, which is as per r2 – r3. (If r2 is 1000 1000 1000 1000 and r3 is 1000 1000 0000 0000, then r1 will load the 32 bits from r2 – r3 = 0000 0000 1000 1000. The r2 and r3 remain unchanged.)

c. A byte or half-word or word address is pointed by the indirect base address by adding GPR with an offset as immediate operand. This mode is indirect addressing by *base plus 12-bit byte_offset* (like base-relative or index relative in 80x86 processor).

Example 15.12

1. LDR r4, [r5, #9] instruction loads r4 from the memory address pointed by r5 + 9. (If base address at r5 is 0000 0000 1000 1000, then r4 will load the 32 bits from memory address 0000 0000 1001 0001. The r5 remains unchanged.)
2. STR (r5, #&0C), r4 instruction store r4 32-bits to the memory address pointed by r5 + 12.

d. There is an indirect addressing mode by *index plus offset with post auto indexing.*

Example 15.13

LDR r4, [r5], #8 instruction loads r4 from the memory address pointed by r5 and then adds the offset 8 to r5. (If r5 is 0000 0000 1000 1000, then r4 will load the 32 bits from memory address from 0000 0000 1000 1000 and make 0000 0000 1001 0000 by adding offset = 8_d at the immediate operand.) The r5 remains changed for the use of r5 next time. This is called auto post indexing.

e. There is an indirect addressing mode by index plus offset with *pre-auto indexing* by using an operator.

Example 15.14

1. *LDR r4, [r5, #8]!* instruction loads r4 from the memory address pointed by r5 after first adding (due to operator ! in the instruction) byte_offset 8 into r5. If r5 is 0000 0000 1000 1000, then r4 will load 32 bits from memory address 0000 0000 1001 0000. This is because of the operator first adding byte_offset = 8 into r5 value before using r5 to point to memory address for loading r4. The r5 changes before the present time use of r5. Byte_offset can be a 12-bit immediate operand. The mode is called auto indexing with base register also updating before fetching.
2. *LDR r15, r1!* instruction loads r15 to the memory address pointed by r13 after first adding (due to operator ! in the instruction) the offset 1 into r13. (Offset 1 means address change of 4 because a 32-bit word = 4 bytes. If r15 is 0000 0000 1000 1000, then r15 will load to the 32 bits from memory address from r13 after making r13 = 1000 0000 0000 1100 if r13 earlier = 1000 0000 0000 1000. This is because of an operator first adds offset = 1 (means 4) into r13 value before using r13 to point to memory address for loading r15. The r13 is changed before the present time use of r13. This is called auto indexing with the base register also updating before fetching).

All instructions have condition-test field in 4 msbs at 32-bit instruction. When cond = 1110, the condition test is ignored and instructions are always executed irrespective of flags. The instruction types are as follows.

15.7.2 Data Transfer, Bit Clearing and Masking Instructions

These are three types of data-transfer instructions: (a) Move (conditional or unconditional and with or without the shift or rotate) from first to second register, (b) Load from a memory address source operand indirect or indirect with offset or (c) Store to destination operand memory address indirect or indirect with offset. MOV and MVN are *move* and *move the complement* to a register and/are register-to-register move instructions. Table 15.3 in Section 15.7.2 gives the instruction format. Second source operand can also

be shifted or rotated before using the source in the comparison operations. Flags can be changed as per the S-bit in the instruction. Move can be conditional or unconditional as per cond 4-bit field in the instruction (Table 15.3).

TABLE 15.3 Instruction Format for 32-bit Move or Compare or Arithmetic Add or Subtract or Logic Operation Instructions

Bit(s)	Field Name	Use
b31–b28	cond	Specifies test condition.
b27–b26	00	Major opcode specifies one of the four group of operations move or compare or add or subtract or logic operations.
b25	X	*X* = 1 means that 12 lowest significance bits specify immediate 8-bit operand, which is used after rotation specified by 4 bits b8–b11; = 0 means that 12 lowest significance bits specify source operand register which is used for shifts in cases 1 and 2
b24–b19	opcode	4-bit Minor opcode specifyies the specific operation in the group
b20	S	S specifies CPSR flags need change or no change from the operation.
b19–b16	Rn	First source operand Rn. = r4 if bits b15-b12 = 0100 and r15 if 1111.
b15–b12	Rd	Destination register Rd. = r9 if bits b15-b12 = 1001 and r15 if 1111.
b11–b0	format as per b25 (X) (b0–b11 are for operand2)	If *X* =1 then b11–b8 specifies rotate right instruction and b7–b0 immediate operand. b11–b8 = 0000 means no rotation, b11–b8 = 0001 means rotate right 2 times, b11–b8 = 1111 means ROR 30 times. Immediate operand is 32-bit with 24 msbs = 0s and lsbs b7–b0.
		Case 1: *X* = 0 and b4 = 1 then b11–b8 specifies shift register, which tells how many times shift, b7 = 0, b6–b5 specifies a field Sh, which tells LSL or LSR, ASL or ASR instruction and b3–b0 specifies register for second source operand.
		Case 2: *X* = 0 and b4 = 0 then b11–b7 specifies tells how many times shift (between 0 to 31), b6–b5 specifies a field Sh, which tells whether LSL or LSR, ASR or ROR operation (refer text) and b3–b0 specifies register for second source operand.

LDRB, LDRH, LDRSH and LDR are the *load* byte, half-word, half-word signed and word-load instructions, respectively. These are indirect memory address-to-register load instructions, in Section 15.7.1(3). STRB, STRH, STRSH and STR are byte, half-word, half-word signed and word store instructions. These are register-to-indirect address based *store* instructions (signed half-word means extend the sign 0 or 1 at *b15* into the bit *b16* up to bit *b31*).

There is a single data (8 or 32-bits) *swap* (conditional or unconditional) between register and memory. Swap takes place by a memory *read* and memory *write*. Instruction specifies source (*Rm*), destination (*Rd0*) and base (*Rn*) registers. *Rn* gives the swap address. Data bits of *Rm* are written at the memory address. Old data bits at memory are written at destination register *Rd)* .

Block Data Load and Store Instructions LDM and STM do the block registers *load* and *store* from or to the memory, respectively. LDM and STM are mainly used for popping and pushing when performing stack operations using these instructions.

Data and Bit Clearing and Masking During Manipulation BIC *r1*, *r2*, *r3* performs find bits after logic AND between *r2* and complements of *r3* bits and moves the result in *r1*.

Example 15.15

Assume that *r1* is to acquire bits of *r2* after clearing the bit specified in *r3*, and then BIC is used. This is assuming that *r3* is 1100 1110 0111 1111. The *r3* functions as a mask register. This means, BIC instruction will clear all bits except *b7*, *b8*, *b12* and *b13* bits from the *r2* bits to put the result in *r1*. Assume *r2* = 1111 1110 1000 1000, then BIC *r1*, *r2*, *r3* will result in *r1*←0011 0000 1000 0000. If *r3* is 1000 0000 0000 0000 only then *b15* will clear in *r1* using *r2* and *r1* will become 0111 1110 1000 1000.

7.3 Arithmetic operation instructions

Basic arithmetic instructions are as follows:

1. There are the destination, first source and second source operands. Second source operand can also be shifted or rotated before using the source in the arithmetic add and subtract operations.
2. There are arithmetic instruction operations of ADD, ADC (add with C flag), SUB, SBC (subtract with C flag used for borrow) and MUL. (8051 does not have SUB subtract without borrow.)

Example 15.16

MULL (32 × 32: 64) is to multiply a long instruction. These arithmetic instruction operations can be defined as conditional or unconditional and defined with or without shift rotate manipulation of second source operand.

3. ARM has instructions for reverse subtraction (first source from the second and with C flag used borrow. This is given because second source can be bit manipulated before being used in an ARM arithmetic instructions.
4. There can be conditional or unconditional MUL and MLA in a 32 × 32 multiply (32 × 32: 32) and multiply with add instruction. MUL can multiply only. MLA multiplies followed by add (bit 0 or 1 in the opcode specifies it). There can be multiplication with or without change of the condition flags (S bit 1 or 0 specifies it). The signed or unsigned multiplication is only 32-bit long.
5. There can be conditional or unconditional in MULL and MLAL instructions. These are 32 × 32 multiply long (32 × 32:64) without or with add instruction, respectively. MULL and MLAL are specified for signed or unsigned operation by U (1 or 0) bit at the opcode for multiply or multiply and add by A (0 or 1) and to change and not to change flags by S bit (1 or 0).
6. These operations are used in control and signal processing systems.

Example 15.17

MLA r1, r2, r3, r4 will multiply r2 and r3 and add the result with r4 and accumulate the result in r1. MLAL r1, r2, r3, r4 will multiply r3 and r4 and add the result with r1(H) r2(L). Sixty-four-bit result accumulates r1(H) r2(L) itself after adding the multiplication of r3 and r4.

7. ARM variant M and ARM7TDMI have instructions MLA and MLAL. These are for *MAC* 32 × 32 multiply and 64-bit accumulate by summing.

Example 15.18

MLAL helps in evaluating a result by summing a multi-term finite series consisting of products of coefficients and variables. Some examples are $y = c0 + i.a + j.b + k.c$ and $y1 = c1 + c1.x + c2.x2 + c3.x3 + \ldots$ MLAL helps in matrix or multi-term series multiplications.