

ARM Microcontroller

October 28, 2020

Introduction

(Basic features of ARM (Advanced RISC Machine))

- 32-bit microcontroller
- 32-bit data bus
- 32-bit registers
- 32-bit instruction set
- 32-bit address bus
- based on von Neumann architecture
- 37 registers (16 usable at a time)
- Load-Store model
- supports 3 data formats (8/16/32 bit i.e. byte/half-word/word)

RISC Philosophy in ARM (Advanced RISC Machine)

Some of the RISC features incorporated in ARM micro-controller are:

- A large array of uniform registers
- A uniform fixed length instructions (32-bit)
- A load-store model of data-processing where operations can only operate on registers and not directly on memory. This requires that all data be loaded into registers before an operation can be performed, the result can then be used for further processing or stored back into memory
- A small number of addressing modes with all load/store addresses being determined from registers and instructions field only

In addition to these traditional features of a RISC system the ARM provides a number of additional features:

- Separate Arithmetic Logic Unit (ALU), multiplier and shifter gives additional control over data processing to maximize execution speed
- Less complicated addressing modes compared to x86 architecture (Register, Immediate, Register Indirect and Indexed)
- Conditional execution of instructions to reduce the number of branch instructions

ARM family history

- ARM Corp. receives its entire revenue from licensing the ARM to other companies since it does not own state of the art chip fabrication facility. This business model is making money from selling IP (intellectual property). ARM has defined the details of architecture, registers, instruction set, memory map, and timing of the ARM CPU and holds the copyright to it. The various design houses and semiconductor manufacturers license the IP (intellectual property) for the CPU and can add their own peripherals.
- Some of the well-known licensees of ARM are:
VLSI Technology, GEC Plessey, Sharp, Texas Instruments, Hyundai, Lucent, Philips, Rockwell, Sony, HP, IBM, Matsushita, Seiko Epson, Qualcomm, LSI Logic, ST Microelectronics, Fujitsu, Agilent, Altera, Micronas, Mitsubishi, Motorola, Sanyo, Triscend, ZTEIC, Global UniChip, Samsung, Zeevo, Seagate, Broadcom, Micrel, eSilicon, Chip Express, ITRI, Flextronics, Microsoft, Cadence, TSMC etc.

ARM family history contd.

- 1985: Acorn computer group developed world's first commercial RISC processor ARMv1. It had 2500 transistors and worked with 4 MHz frequency
- 1989: Acorn introduced ARMv3 with frequency of 25 MHz with 4 kB cache
- 1990: Advanced RISC Machines (ARM) spins out of Acorn and Apple Computers collaboration efforts with a charter to create a new microprocessor standard. VLSI Technology becomes an investor and the first licensee.
- 1991: ARM introduced its first embedded RISC core, the ARM6 solution using ARMv3 architecture.
- 1995: ARM announced the Thumb architecture extension, which gives 32-bit RISC performance at 16-bit system cost
- 1997: ARM710T launched with Thumb architecture
- 1999: ARM announced synthesizable ARM9E processor with enhanced signal processing

ARM family history contd.

- 2000: ARM launched SecurCore family for smartcards
- 2004 The ARM Cortex family of preprocessors based on the ARMv7 architecture was announced. The first processor of this family was ARM Cortex-M3
- 2007: ARM unveils Cortex-A9 processor for scalable performance and low-power designs
- 2008: ARM Mali-200 GPU (Graphics Processing Unit) launched
- 2009: ARM announces 2 GHz capable Cortex-A9 dual core processor implementation
- 2009: ARM launches its smallese, lowest power, most energy efficient processor Cortex-M0
- 2010: ARM launches Cortex-M4 processor for high performance digital signal control
- 2010: ARM Mali-T604 GPU introduced

ARM family history contd.

- 2011: ARM Mali-T658 GPU launched
- 2012: First Windows RT (Windows on ARM) device revealed
- 2015: Raspberry Pi 2 featuring a 900 MHz quad-core ARM Cortex-A7 processor launched
- 2018: Cortex M-35P introduced for hardware based security solutions for protection of IoT devices
- 2019: ARM launches Neoverse family processors E1 and N1 for cloud based high performance computing and data processing
- 2019: ARM launches Ethos neural processor family for machine learning interface
- 2020: ARM introduces Cortex M-55 processor for enhancing AI performance in IoT devices

ARM family of processors

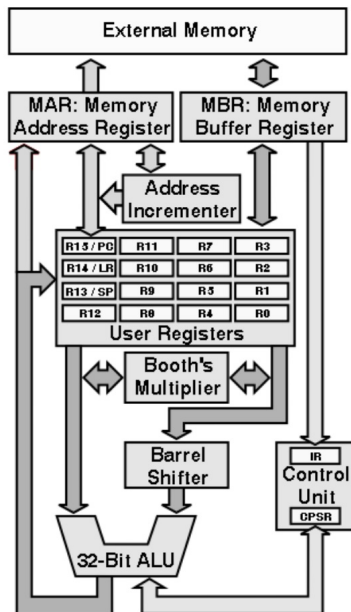
- Classic Core: Conventional RISC processors which can be used for:
 - ▶ faster execution of instructions
 - ▶ operating systems that makes use of MMU
 - ▶ interrupt latency has minimum effect on the overall system performance
- Cortex Core: Designed for embedded System-on-chip (SoC) based simple applications without any cache memory and co-processors. These can be used when the criteria/priority is: low cost, lower interrupt latency, less power consumption and limited code size (Only thumb instructions are used). Different series of Cortex Core are:
 - ▶ Cortex A series: Offers supreme performance at optimum power for industrial, automotive systems, consumer devices etc.
 - ▶ Cortex R series: These are high performance real time processors.
 - ▶ Cortex M series: High performance processors for machine learning, AI interface

ARM family of processors contd.

- SecurCore: Designed for smartcard based physical security applications
- Neoverse Core: Designed specifically for high performance computing and data handling for server based applications
- Ethos Core: Designed for high performance machine learning interface

ARM processor fundamentals

ARM Block Diagram

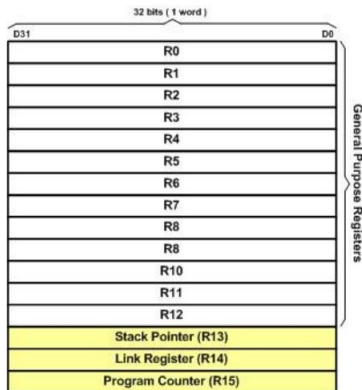


ARM Block Diagram

- Arithmetic and Logic Unit (ALU): The ALU has two 32-bits inputs. The first comes from the register file while the other comes from the shifter. ALU outputs modify the status register flags.
- Booth Multiplier: This unit performs 32-bit multiplication operations using Booth's algorithm. Booth's algorithm is a multiplication algorithm that multiplies two signed binary numbers in two's complement notation.
- Barrel Shifter: This unit performs the logical and arithmetic shifting operations. The barrel shifter has a 32-bit input to be shifted. This input is coming from the register file or it could be immediate data.
- Control Unit: Control unit holds the current status of the processor. It contains the flag, status and instruction registers. It also decodes the instruction stored at the Instruction Register.
- MAR holds the address of the external memory to/from which data is to be transferred and MBR holds the data to be transferred to/from the address stored in MAR.

General Purpose Registers (GPR)

- All ARM registers are 32 bit wide
- ARM has 13 general purpose registers, R0-R12
- They are same as the accumulator in other microprocessors
- They can be used by all arithmetic and logical instructions



Sample instructions related to registers

- MOV instruction:
 - ▶ MOV R2,#0x5 ;load R2 with 5 ($R2 = 0x05$)
 - ▶ MOV R5,R7 ;copy contents of R7 into R5 ($R5 = R7$)
 - ▶ MOV R7,#12 ;R7 = 00001100 or 0C in hex (i.e. 12 in decimal)
- Load and Store instructions:

Data Size	Bits	Decimal	Hexadecimal	Load instruction used
Byte	8	0 – 255	0 - 0xFF	LDRB
Half-word	16	0 – 65535	0 - 0xFFFF	LDRH
Word	32	0 – $2^{32}-1$	0 - 0xFFFFFFFF	LDR

Unsigned Data Range in ARM and associated Load Instructions

Data Size	Bits	Decimal	Hexadecimal	Load instruction used
Byte	8	0 – 255	0 - 0xFF	STRB
Half-word	16	0 – 65535	0 - 0xFFFF	STRH
Word	32	0 – $2^{32}-1$	0 - 0xFFFFFFFF	STR

Unsigned Data Range in ARM and associated Store Instructions

Sample instructions contd.

- Load: LDR instruction copies the contents of four consecutive memory locations into a 32-bit GPR
 - ▶ LDR R0,=0xF62562FA ;R0 = 0xF62562FA
 - ▶ ;assume R5 = 0x40000200
LDR R7,[R5] ;load R7 with the contents of locations
;0x40000200-0x40000203
LDRH R7,[R5] ; load R7 with the contents of locations
;0x40000200-0x40000201
LDRB R7,[R5] ; load R7 with contents of location 0x40000200
- Store: STR instruction copies the contents of 32-bit GPR into four consecutive memory locations.
 - ▶ ;assume R5 = 0x40000100, R1=0x55 and R2 = 0x45260243
STRB R1,[R5] ;copy R1 location pointed to by R5
STRB R2,[R5] ;copy 8 bits of R2 (0x43) location pointed to by R5
STRH R2,[R5] ;copy 16 bits of R2 (0x43 and 0x02) to locations
;0x40000100 and 0x40000101
STR R2,[R5] ;copy 32 bit R2 to locations 0x40000100 - 0x40000103

Sample programs

;assembly language program in ARM to add 2 numbers

MOV R2,#0x5 ;load R2 with 5 ($R2 = 0x05$)

MOV R1,#0x2 ;load R1 with 2 ($R1 = 0x02$)

ADD R2, R1,R2 ; $R2 = R1 + R2$

MOV R5,#0x20 ; $R5 = 0x20$

STRB R2,[R5] ;store R2 into location pointed to by R5

;assembly language program in ARM to subtract 2 numbers

MOV R2,#4 ; $R2 = 4$

MOV R3,#2 ; $R3 = 2$

MOV R4,#4 ; $R4 = 4$

SUB R5,R2,R3 ; $R5 = R2 - R3$ ($R5 = 4 - 2 = 2$)

SUB R5,R2,R4 ; $R5 = R2 - R4$ ($R5 = 4 - 4 = 0$)

Q. State the contents of R2, R1, and memory location 0x20 after the following program:

```
MOV R2,#0x5      ;load R2 with 5 (R2 = 0x05)
MOV R1,#0x2      ;load R1 with 2 (R1 = 0x02)
ADD R2, R1,R2     ;R2 = R1 + R2
ADD R2,R1,R2      ;R2 = R1 + R2
MOV R5,#0x20      ;R5 = 0x20
STRB R2,[R5]      ;store R2 into location pointed to by R5
```

Solution:

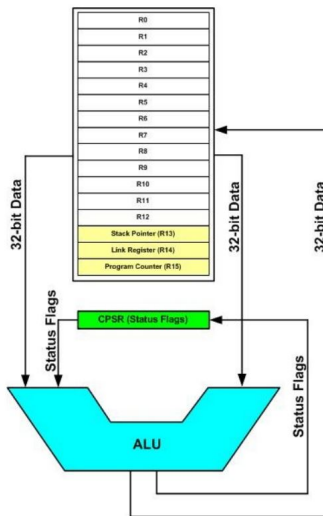
R1: 0x2

R2: 0x9

0x20: 0x9

Special Function Registers (SFR)

- R13, R14, R15 and CPSR (current program status register) are called special function registers

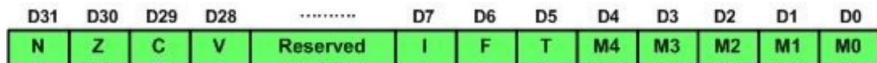


Special Function Registers (SFR)

Each SFR is dedicated to a specific function:

- R13 is the stack pointer
- R14 is designated as link register which holds the return address when the CPU calls a subroutine/ISR
- R15 is the program counter (PC) and it points to the address of the next instruction to be executed. A 32-bit PC can access a maximum of 4G ($2^{32} = 4G$) bytes of program memory locations
- CPSR (Current Program Status Register) is used to store the current status of the processor. This includes various condition code flags, interrupt status, processor mode etc.
- In the exception mode, some of the general purpose registers are used as Saved Processor Status Register (SPSR) to store the processor status (contents of CPSR) during the execution of the exception/interrupt.

CPSR (Current Program Status Register)



CPSR is a 32-bit register. The processor status is divided into two distinct parts: The User flags and the System Control flags. The upper halfword is accessible in User mode and contains a set of flags which are effected by execution of a program. The lower halfword contains System Control information.

CPSR contd.

- User Flags: The upper four bits of the status register contains a set of four flags, known as the conditional flags.
 - ▶ C (Carry flag): This flag is set whenever there is a carry out from the D31 bit. This flag bit is affected after a 32-bit addition or subtraction.
 - ▶ Z (Zero flag): The zero flag reflects the result of an arithmetic or logic operation. If the result is zero then $Z = 1$ and if the result is not zero then $Z = 0$.
 - ▶ N (Negative flag): Binary representation of signed numbers uses D31 as the sign bit. The negative flag reflects the result of an arithmetic operation. If the D31 bit of the result is zero, then $N = 0$ and the result is positive. If the D31 bit is one, then $N = 1$ and the result is negative.
 - ▶ V (Overflow flag): This flag is set whenever the result of a signed number operation is too large, causing the high-order bit to overflow into the sign bit. In general, the carry flag is used to detect errors in unsigned arithmetic operations while the overflow flag is used to detect errors in signed arithmetic operations.

The V and N flag bits are used for signed arithmetic operations.

CPSR contd.

- System Control flags: The System flags can only be altered when the processor is in protected mode. User mode programs cannot alter these.
 - ▶ The T flag bit is used to indicate the ARM is in Thumb state.
 - ▶ The I flag indicates whether Interrupt (IRQ) is enabled or not.
 - ▶ The F flag indicates whether Fast Interrupt (FIQ) is enabled or not.
 - ▶ The mode bits ($M_0 - M_4$) indicate which operating mode the processor is in.

M[4:0]	Mode
10000	User
10001	FIQ
10010	IRQ
10011	SVC
10111	Abort
11011	Undefined
11111	System

Sample programs

;assembly language program in ARM for addition with carry

LDR R0,#0xF62562FA ;R0 = 0xF62562FA

LDR R1,#0xF412963B ;R1 = 0xF412963B

LDR R2,#0x00000000 ;R2 = 0x00000000, it will store the sum

LDR R3,#0x00000000 ;it will store the sum with carry

MOV R4,#0x00 ;R4 = 0x00, it will store the carry

ADDS R2,R1,R0 ;R2 = 0xF62562FA + 0xF412963B ;now C = 1

ADC R3,R1,R0 ;R3 = R1 + R0 + C sum with carry

ADC R4,R4,#0 ;R4 = R4 + 0 + C = 0x00 + 0 + 1 = 0x01

;assembly language program in ARM for subtraction with borrow

LDR R0,#0xF62562FA ;R0 = 0xF62562FA

LDR R1,#0xF412963B ;R1 = 0xF412963B

LDR R2,#0x00000000 ;R2 = 0x00000000, it will store the difference

LDR R3,#0x00000000 ; it will store the difference with borrow

SUBS R2,R1,R0 ;R2 = R1-R0 = 0xF412963B - 0xF62562FA, and C = 0

SBC R3,R1,R0 ;R3 = R1 - R0 - 1 + C ; complement the carry

;assembly language program in ARM for Bit shifting

MOV R0,#0x9A ;R0 = 0x9A

MOVS R1,R0,LSR #3 ;shift R0 to right 3 times

;and then move (copy) the result to R1

;assembly language program in ARM to find one's complement of a number

MOV R1,#0x7 ;R1=0x7

LDR R0,=0x00001000 ;R0=location where result to be stored

MVN R1,R1 ;bitwise logical NOT operation and store result in R1

STR R1,[R0] ; store the result at location pointed by R0

Memory Map

ARM has 4 GB of directly accessible memory space. This memory space has addresses 0 to 0xFFFFFFFF. The 4 GB of memory space can be divided into five sections. They are as follows:

- ➊ On-chip peripheral and I/O registers: This area is dedicated to general purpose I/O (GPIO) and special function registers (SFRs) of peripherals such as timers, serial communication, ADC, and so on.
- ➋ On-chip data SRAM: The data RAM space is read/write memory used by the CPU for storage of data variables, scratch pad, and stack.
- ➌ On-chip EEPROM: EEPROM is used for program code storage. Not all ARM chips have on-chip EEPROM.
- ➍ On-chip Flash ROM: This is a flash type program space used for the program code and storage of look-up tables.
- ➎ Off-chip DRAM space: External memory used to store both code and data.

Memory Map

There is no standard for exact locations and types of memory and peripherals. Therefore the licensees can implement the memory and peripherals as they choose. For this reason the amount and the address locations of memory used by Flash ROM, SRAM, and I/O peripherals varies among the family members and chip manufacturers. ARM uses this 4 GB for both memory and I/O space. This mapping of the I/O ports to memory space is called memory mapped I/O.

Q. Find the address space range of each of the following memory of an ARM chip and draw the memory map:

- a) 4KB EEPROM starting at address 0x00100000
- b) 32KB SRAM starting at address 0x40000000
- c) 512KB Flash starting at address 0x00000000
- d) 256KB memory for peripherals (SFR) starting at 0xFFFC0000

Solution:

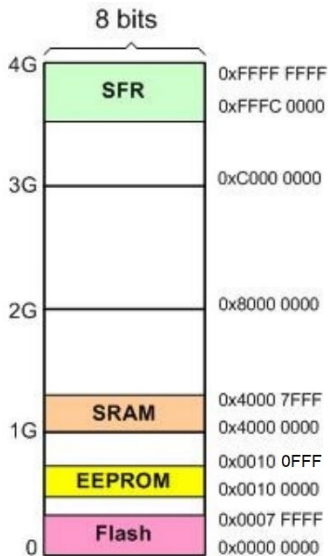
a) 4KB = 4096 bytes i.e. 4096_{10} locations = 1000_{16} locations in Hex starting from 0x00100000. Therefore the address space range of EEPROM is 0x00100000 to 0x00100FFF.

b) 32KB = 32768 bytes i.e. 32768_{10} locations = 8000_{16} locations in Hex starting from 0x40000000. Therefore the address space range of SRAM is 0x40000000 to 0x40007FFF.

c) 512KB = 524288 bytes i.e. 524288_{10} locations = 80000_{16} locations in Hex starting from 0x00000000. Therefore the address space range of Flash memory is 0x00000000 to 0x0007FFFF.

d) 256KB = 262144 bytes i.e. 262144_{10} locations = 40000_{16} locations in Hex starting from 0xFFFC0000. Therefore the address space range of SFR is 0xFFFC0000 to 0xFFFFFFFF.

Memory map:



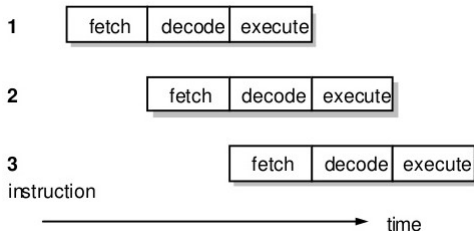
Memory Map in ARM Cortex

Address range	Name	Description
0x00000000-0x1FFFFFFF	Code	ROM or Flash memory
0x20000000-0x3FFFFFFF	SRAM	SRAM region used for on-chip RAM
0x40000000-0x5FFFFFFF	Peripheral	On-chip peripheral address space
0x60000000-0x9FFFFFFF	RAM	Memory, cache support
0xA0000000-0xDFFFFFFF	Device	Shared and non-shared device space
0xE0000000-0xFFFFFFFF	System	PPB and vendor system peripherals

Pipelines

- The Process of fetching the next instruction while the current instruction is being executed is called as pipelining
- Pipelining is supported by the processor to increase the speed of program execution
- A 3-stage pipeline process has the following stages:
 - ① Fetch : In this stage the ARM processor fetches the instruction from memory
 - ② Decode : This stage recognizes the instruction that is to be executed
 - ③ Execute : In this stage the processor processes the instruction and writes the result back to desired register
- 3-stage pipelining is present in version 7 of ARM processor

ARM single cycle instruction 3-stage pipeline operation



- First cycle: Processor fetches instruction 1 from memory
- Second cycle: Fetches instruction 2 from memory and decodes instruction 1
- Third cycle: Fetches instruction 3 from memory, decodes instruction 2 and executes instruction 1
- Fourth cycle: Fetches instruction 4, decodes instruction 3 and executes instruction 2

The pipeline thus executes an instruction in three cycles i.e. it delivers a throughput equal to one instruction per cycle.

5-stage pipelining in ARM 9

- ❶ Fetch : Instructions are fetched from memory and placed in the queue and wait to be decoded. In this stage the Program Counter (PC) is also incremented by 4 since the ARM instructions are 4-bytes.
- ❷ Decode : Instruction is decoded and the register file is also accessed to get everything ready for the Execute stage.
- ❸ Execute : In this stage, instruction is executed and all the effective address calculations are done.
- ❹ Memory : For instructions such as Load and Store in which external memory accesses are needed, the memory stage fetches the data from the external memory and has the data inside the CPU ready for the next stage of the write-back.
- ❺ Write : Also called write-back, is the stage in which the instruction is completed by writing the result to the register file and retiring the instruction.

If an instruction does not need to access memory, write-back is the stage right after the execute stage, meaning the process becomes a 4-stage pipeline.

ARM processor modes

Processor	mode	Description
Supervisor	svc	A protected mode for the operating system
User	usr	Normal program execution mode
System	sys	Runs privileged operating system tasks
FIQ	fiq	Fast Interrupt for high-speed data transfer
IRQ	irq	Used for general-purpose interrupt handling
Abort	abt	Implements virtual memory and/or memory protection
Undefined	und	Supports software emulation of hardware co-processors

Mode changes can be made under software control, or can be caused by external interrupts or exception processing.

ARM processor modes

- User mode: In this mode, the program being executed is unable to access some protected system resources or to change mode. This allows a suitably written operating system to control the use of system resources. Most applications/programs execute in User mode.
- Privilege mode: The modes other than User mode are known as privileged modes. They have full access to system resources and can change mode freely.
 - ▶ Five of them are known as exception modes: FIQ, IRQ, Supervisor, Abort and Undefined. These are entered when specific exceptions occur. Each of them has some additional registers to avoid corrupting User mode state when the exception occurs.
 - ▶ System mode: This mode is not entered by any exception and has exactly the same registers available as User mode. However, it is privileged mode and is therefore not subject to the User mode restrictions. It is intended for use by operating system tasks which need access to system resources.

Exceptions

ARM exception types:

- Reset
- Undefined instruction
- Software interrupt (SWI)
- Prefetch abort
- Data abort
- IRQ
- FIQ

ARM exception types

① Reset

- ▶ Occurs when the processor reset pin is asserted
- ▶ Software reset: can be done by branching to the reset vector (0x0000)

② Undefined instruction

- ▶ Occurs when the processor or coprocessors cannot recognize the currently executed instruction

③ Software interrupt

- ▶ User-defined interrupt instruction
- ▶ Allow a program running in User mode to request privileged operations that are in Supervisor mode

④ Prefetch abort

- ▶ Fetch an instruction from an illegal address, the instruction is flagged as invalid
- ▶ However, instructions already in the pipeline continue to execute until the invalid instruction is reached and then a Prefetch Abort is generated.

ARM exception types

5 Data abort

- ▶ A data transfer instruction attempts to load or store data at an illegal address

6 IRQ

- ▶ The processor external interrupt request pin is asserted (LOW) and the I bit in the CPSR is clear (enable)

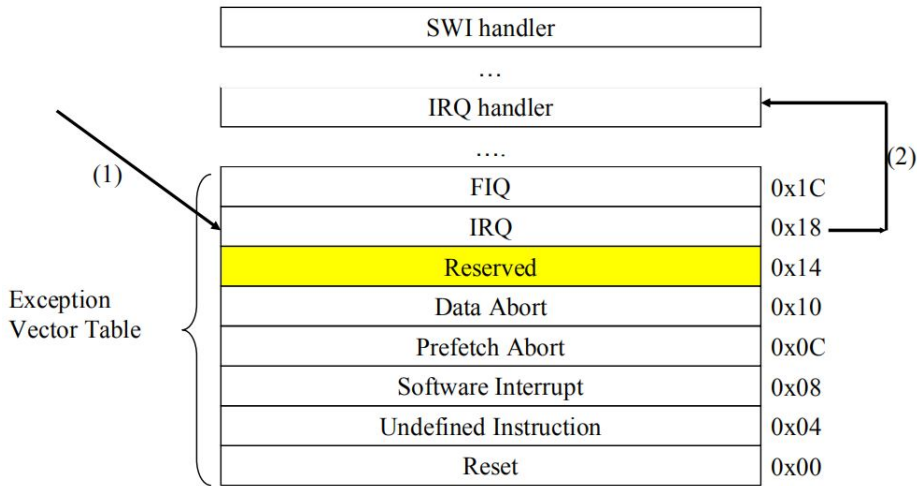
7 FIQ

- ▶ The processor external fast interrupt request pin is asserted (LOW) and the F bit in the CPSR is clear (enable)

Interrupt Vector Table

Exception type	Mode	Normal address	High vector address
Reset	Supervisor	0x00000000	0xFFFF0000
Undefined instructions	Undefined	0x00000004	0xFFFF0004
Software interrupt (SWI)	Supervisor	0x00000008	0xFFFF0008
Prefetch abort	abort	0x0000000C	0xFFFF000C
Data abort	abort	0x00000010	0xFFFF0010
IRQ (interrupt)	IRQ	0x00000018	0xFFFF0018
FIQ (fast interrupt)	FIQ	0x0000001C	0xFFFF001C

Interrupt Vector Table



Interrupt Vector Table

The following events take place on occurrence of an exception:

- Exception changes the mode of the processor.
- The current status of the processor stored in CPSR are copied into SPSR. The program counter value is saved in link register.
- The new values are loaded into CPSR. The program counter is set to the vector address of that exception. [(1) in the previous diagram]
- The interrupt vector table transfers the execution to the corresponding exception handler routine. [(2) in the previous diagram]
- After execution of the exception handler, processor returns from the handler by updating the CPSR with previously stored values in SPSR. PC is loaded with the return address from the link register.

ARM exception priorities

Exception type	Mode	Vector address	Priority (1=high,6=low)
Reset	Supervisor	0x0	1
Undefined instructions	Undefined	0x4	6
Software interrupt (SWI)	Supervisor	0x8	6
Prefetch abort	abort	0xC	5
Data abort	abort	0x10	2
IRQ (interrupt)	IRQ	0x18	4
FIQ (fast interrupt)	FIQ	0x1C	3

Programmer's model of ARM

ARM has 37 registers (32-bit each) in total-

- 30 general purpose registers
- 1 dedicated program counter
- 1 current program status register (CPSR)
- 5 saved program status register (SPSR)

In any mode only 16 registers out of 37 are accessible.

- The hidden registers are called *banked registers*
- Current processor mode governs which registers are accessible

Programmer's model of ARM

Each mode can access:

- A particular set of general purpose registers (R0 to R12)
- A particular set of special function registers:
 - ▶ Stack pointer (R13)
 - ▶ Link register (R14)
- The program counter (R15)
- The current program status register (CPSR)
- Apart from these the privileged modes (except System mode) can access a particular saved program status register (SPSR)

Programmer's model of ARM

- Accessible and banked registers in User mode:

Current Visible Registers

User Mode

r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)
r15 (pc)
cpsr





















Banked out Registers

FIQ	IRQ	SVC	Undef	Abort
r8				
r9				
r10				
r11				
r12				
r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)
r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)
spsr	spsr	spsr	spsr	spsr

- The values stored in banked registers are preserved across mode changes

Programmer's model of ARM

- Register organization:

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	 R8-fiq	R8	R8	R8	R8
R9	 R9-fiq	R9	R9	R9	R9
R10	 R10-fiq	R10	R10	R10	R10
R11	 R11-fiq	R11	R11	R11	R11
R12	 R12-fiq	R12	R12	R12	R12
R13	 R13-fiq	 R13-svc	 R13-abt	 R13-irq	 R13-und
R14	 R14-fiq	 R14-svc	 R14-abt	 R14-irq	 R14-und
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	 SPSR-fiq	 SPSR-svc	 SPSR-abt	 SPSR-irq	 SPSR-und

 = banked register

SPSR = State Program Status Register

Sample programs

- Instructions to know status of / set the CPSR:
 - ▶ MRS R0,CPSR ; Move CPSR to general purpose register
 - ▶ MSR CPSR,R0 ; Move to CPSR

;Assembly language program to clear N,Z,C,V flags of ARM processor

```
MRS R0,CPSR ; move CPSR to R0
```

```
BIC R0,R0,#0xF0000000 ;
```

```
MSR CPSR_f,R0 ;move R0 to CPSR
```

;assembly language program to set Interrupt Request bit of CPSR (IRQ enabling)

```
MRS R0,CPSR ; move CPSR to R0
```

```
ORR R0,R0,#0x080 ;
```

```
MSR CPSR_f,R0 ;move R0 to CPSR
```

Sample programs

;assembly language program to switch ARM to supervisor mode from another privileged mode

MRS R0,CPSR ; move CPSR to R0

BIC R0,R0,#0x1F;modify by removing current mode

ORR R0,R0,#0x13; substitute with supervisor mode

MSR CPSR_c,R0 ;move R0 to CPSR

Instruction set

Thumb instructions

- Thumb is a compressed, 16bit representation of a subset of the ARM instruction set.
- ARM and Thumb are two different instruction sets supported by ARM cores with a “T” in their name. For eg. ARM7 TDMI.
- ARM instructions are 32-bit wide and Thumb instructions are 16-bit wide.
- Thumb mode allows the code to be smaller, and faster in some cases.
- The ‘T’ bit in the CPSR controls the interpretation of the instruction stream.
- Switching between ARM and Thumb can be achieved by executing BX instruction.
- Exceptions can also cause a switch between ARM and Thumb.

Thumb instruction set

Mnemonic	Instruction	Mnemonic	Instruction	Mnemonic	Instruction
ADC	Add with Carry	LDMIA	Load multiple	NEG	Negate
ADD	Add	LDR	Load word	ORR	OR
AND	AND	LDRB	Load byte	POP	Pop registers
ASR	Arithmetic Shift Right	LDRH	Load halfword	PUSH	Push registers
B	Unconditional branch	LSL	Logical Shift Left	ROR	Rotate Right
Bxx	Conditional branch	LDSB	Load sign-extended byte	SBC	Subtract with Carry
BIC	Bit Clear	LDSH	Load sign-extended halfword	STMIA	Store Multiple
BL	Branch and Link	LSR	Logical Shift Right	STR	Store word
BX	Branch and Exchange	MOV	Move register	STRB	Store byte
CMN	Compare Negative	MUL	Multiply	STRH	Store halfword
CMP	Compare	MVN	Move Negative register	SWI	Software Interrupt
EOR	EOR			SUB	Subtract
				TST	Test bits

Extra programs

;assembly language program in ARM for multiplication of two numbers

```
MOV R1,#0x25    ;R1=0x25
MOV R2,#0x65    ;R2=0x65
MUL R3,R1,R2    ;R3 = R1 R2 = 0x65 0x25
```

;assembly language program in ARM to compare two numbers

```
MOV R1,#0x25    ;R1=0x25
MOV R2,#0x65    ;R2=0x65
CMP R1,R2       ;compare R1 and R2
BHI done        ; branch to 'done' if R1 contains the highest
MOV R1,R2       ;otherwise overwrite R1
done: LDR R3,=0x00001000 ;R3=location where result to be stored
STR R1,[R3]     ;store the result in memory location
```

Extra programs

;assemblylanguage program in ARM to divide two numbers

MOV R0,#0x65 ;load dividend in R0

MOV R1,#0x25 ;load divisor number in R1

MOV R3,#0 ;clear register for quotient

loop: CMP R1,#0 ;test for divide by 0

BEQ error

CMP R0,R1 ;is the divisor less than the dividend

BLT done

ADD R3,R3,#1 ;add one to quotient

SUB R0,R0,R1 ;subtract divisor from dividend

B loop ;loop until complete

error: MOV R3,#0xFFFFFFFF ;error flag

done: LDR R5,=0x00001000 ;R5=location where remainder to be stored

LDR R6,=0x00002000 ;R6=location where quotient to be stored

STR R0,[R5] ;store the remainder

STR R3,[R6] ;store the quotient