Introduction to ARM core architecture

EE5182 Microcontrollers and Embedded Systems

Why ARM?

- Leading provider of 32-bit embedded RISC microprocessors
  - 75% of market
- Common architecture
- High performance
- Low power consumption
- Low system cost
- Solutions for
  - Embedded real-time systems for mass storage, automotive, industrial and networking applications
  - Secure applications – smartcards and SIMs
- Open platforms running complex operating systems

History of ARM

- ARM (Acorn RISC Machine) started as a new, powerful, CPU design for the replacement of the 8-bit 6502 at Acorn Computers (Cambridge, UK) in 1985
- First models had only a 26-bit program counter, limiting the memory space to 64 MB (not too much by today standards, but a lot at that time).
- 1990 spin-off: ARM renamed Advanced RISC Machines

History of ARM

- ARM now focuses on Embedded CPU cores
  - IP licensing: Almost every silicon manufacturer sells some microcontroller with an ARM core. Some even compete with their own designs.
  - Processing power with low current consumption
    - Ideal for portable devices
- New cores with added features
  - Harvard architecture (ARM9, ARM11, Cortex)
  - Floating point arithmetic
  - Vector computing (VFP, NEON)
  - Java language (Jazelle)

ARM processors vs. ARM architectures

- ARM architecture
  - Describes the details of instruction set, programmer’s model, exception model, and memory map
  - Documented in the Architecture Reference Manual
- ARM processor
  - Developed using one of the ARM architectures
  - More implementation details, such as timing information
  - Documented in processor’s Technical Reference Manual

What is RISC?

- Reduced instructions – fixed length
- Use of pipelines to breakdown and speed up processing
- Large number of registers – used as very fast onboard RAM
- Load-store architecture – must load and store from memory to register via special instructions
- Overall faster, simpler processer

ARM architecture features

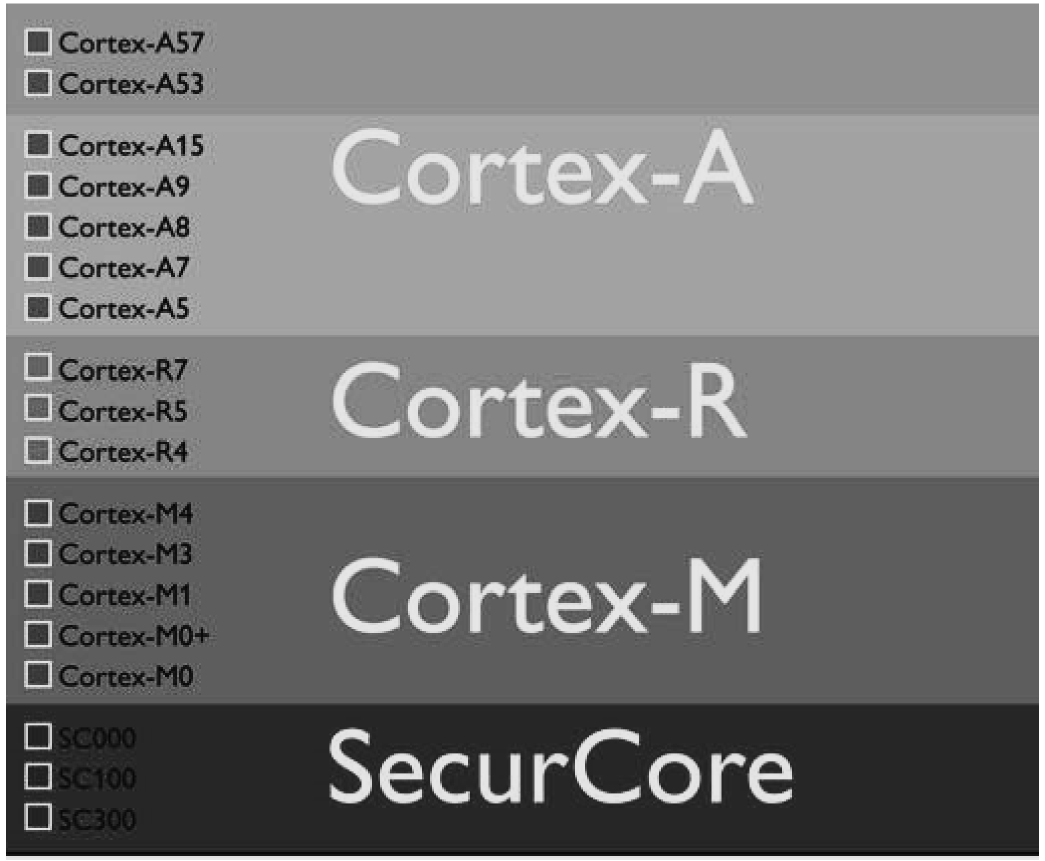
- The typical RISC features:
  - A large uniform register file
  - A load/store architecture, where data-processing operations operate only on register contents, not directly on memory contents
  - Simple addressing modes, with all load/store addresses being determined from register contents and instruction fields only
  - Uniform and fixed-length instructions fields, to simplify instruction decode

ARM architecture features

- Additionally, ARM instruction gives:
  - Control over both the ALU and shifter in every data processing instruction to maximize the use of an ALU and a shifter
  - Auto-increment and auto-decrement addressing modes to optimize program loops
  - Load and Store multiple instructions to maximize data throughput
  - Conditional execution of all instructions to maximize execution throughput.
- These enhancements to a basic RISC architecture allow ARM processor to achieve a good balance of high performance, low code size, low power consumption and low silicon area

ARM processor lines

- ARM architectures and processor families can be profiled into four groups.



ARM processor lines

- The Cortex-M profile
  - Processors of the M profile are optimized for cost sensitive and microcontroller applications, like automotive body electronics, smart sensors.
- The Cortex-A profile
  - It aims at high-end applications running open and complex OSs, like smartphones, tablets, netbooks, eBook readers.
- The Cortex-R profile
  - It marks processors for real time applications, like mass storage or printer controllers.
- The SecureCore profile
  - The ARM SecurCore™ processor family provides processors with security features for applications like smartcards, pay TV, eGovernement.

ARM Cortex-M series

- Cortex-M series: Cortex-M0, M0+, M3, M4, M7, M22, M23
  - Low cost, low power, bit and byte operations, fast interrupt response
- Energy-efficiency
  - Lower energy cost, longer battery life
- Smaller code (Thumb mode instructions)
  - Lower silicon costs
- Ease of use
  - Faster software development and reuse
- Embedded applications
  - Smart metering, human interface devices, automotive and industrial control systems, white goods, consumer products and medical instrumentation

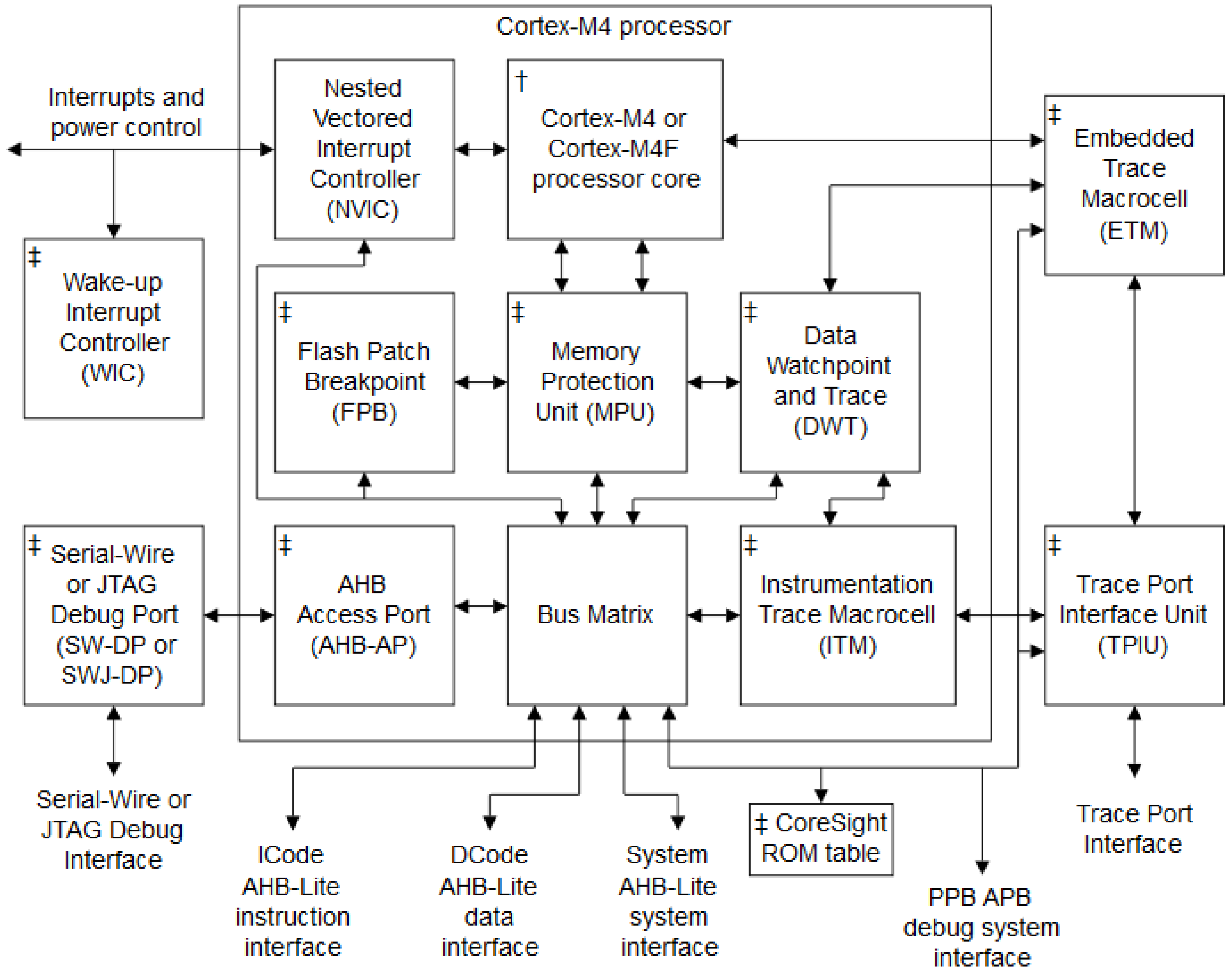
ARM Cortex-M series

- M0: Optimized for size and power (13 µW/MHz dynamic power)
- M0+: Lower power (11 µW/MHz dynamic power), shorter pipeline
- M3: Full Thumb and Thumb-2 instruction sets, single-cycle multiply instruction, hardware divide, saturated math, (32 µW/MHz)
- M4: Adds DSP instructions, optional floating point unit
- M7: designed for embedded applications requiring high performance
- M23, M33: include ARM TrustZone® technology for solutions that require optimized, efficient security

ARM Cortex-M series

ARM Core	Cortex M0	Cortex M0+	Cortex M1	Cortex M3	Cortex M4	Cortex M7	Cortex M23	Cortex M33	Cortex M35P
ARM architecture	ARMv6-M	ARMv6-M	ARMv6-M	ARMv7-M	ARMv7E-M	ARMv7E-M	ARMv8-M Baseline	ARMv8-M Mainline	ARMv8-M Mainline
Computer architecture	Von Neuman	Von Neumann	Von Neumann	Harvard	Harvard	Harvard	Von Neumann	Harvard	Harvard
Instruction pipeline	3 stages	2 stages	3 stages	3 stages	3 stages	6 stages	2 stages	3 stages	3 stages
Thumb-1 instructions	Most	Most	Most	Entire	Entire	Entire	Most	Entire	Entire
Thumb-2 instructions	Some	Some	Some	Entire	Entire	Entire	Some	Entire	Entire
Multiply instructions 32x32 = 32-bit result	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Multiply instructions 32x32 = 64-bit result	No	No	No	Yes	Yes	Yes	No	Yes	Yes
Divide instructions 32/32 = 32-bit quotient	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes

ARM Cortex-M4 internals



Cortex-M4 processor
Interrupts and power control
Nested Vectored Interrupt Controller (NVIC)
† Cortex-M4 or Cortex-M4F processor core
‡ Embedded Trace Macrocell (ETM)
‡ Wake-up Interrupt Controller (WIC)
‡ Flash Patch Breakpoint (FPB)
‡ Memory Protection Unit (MPU)
‡ Data Watchpoint and Trace (DWT)
‡ Serial-Wire or JTAG Debug Port (SW-DP or SWJ-DP)
‡ AHB Access Port (AHB-AP)
Bus Matrix
‡ Instrumentation Trace Macrocell (ITM)
‡ Trace Port Interface Unit (TPIU)
Serial-Wire or JTAG Debug Interface
ICode AHB-Lite instruction interface
DCode AHB-Lite data interface
System AHB-Lite system interface
‡ CoreSight ROM table
PPB APB debug system interface
Trace Port Interface

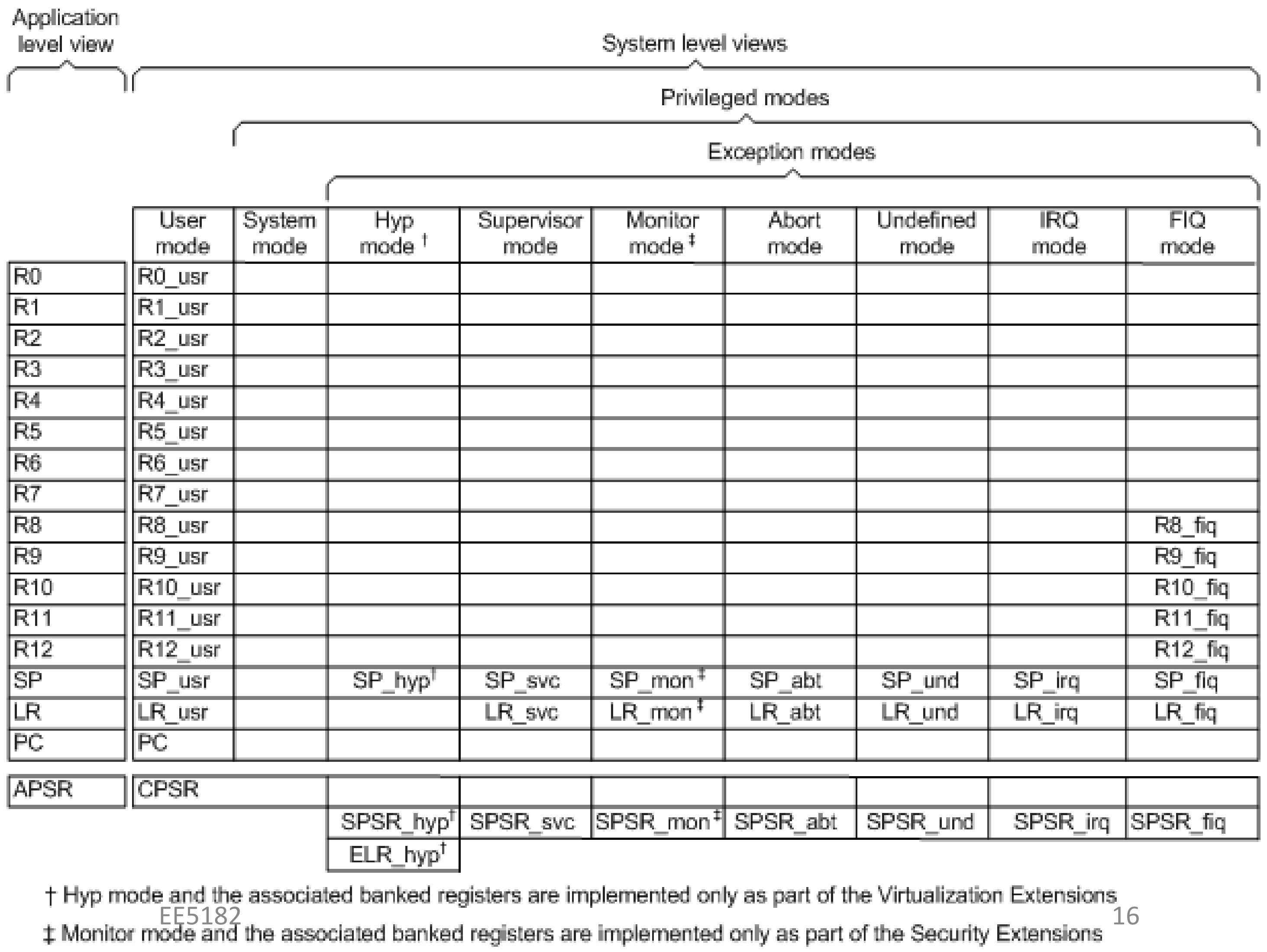
† For the Cortex-M4F processor, the core includes a Floating Point Unit (FPU)
‡ Optional component

ARM processor modes

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

ARM register set

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



Application level view

System level views

Privileged modes

Exception modes

Application level view	User mode	System mode	Hyp mode †	Supervisor mode	Monitor mode ‡	Abort mode	Undefined mode	IRQ mode	FIQ mode
R0	R0_usr								
R1	R1_usr								
R2	R2_usr								
R3	R3_usr								
R4	R4_usr								
R5	R5_usr								
R6	R6_usr								
R7	R7_usr								
R8	R8_usr								R8_fiq
R9	R9_usr								R9_fiq
R10	R10_usr								R10_fiq
R11	R11_usr								R11_fiq
R12	R12_usr								R12_fiq
SP	SP_usr		SP_hyp†	SP_svc	SP_mon‡	SP_abt	SP_und	SP_irq	SP_fiq
LR	LR_usr			LR_svc	LR_mon‡	LR_abt	LR_und	LR_irq	LR_fiq
PC	PC								
APSR	CPSR								
			SPSR_hyp†	SPSR_svc	SPSR_mon‡	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq
			ELR_hyp†						

† Hyp mode and the associated banked registers are implemented only as part of the Virtualization Extensions

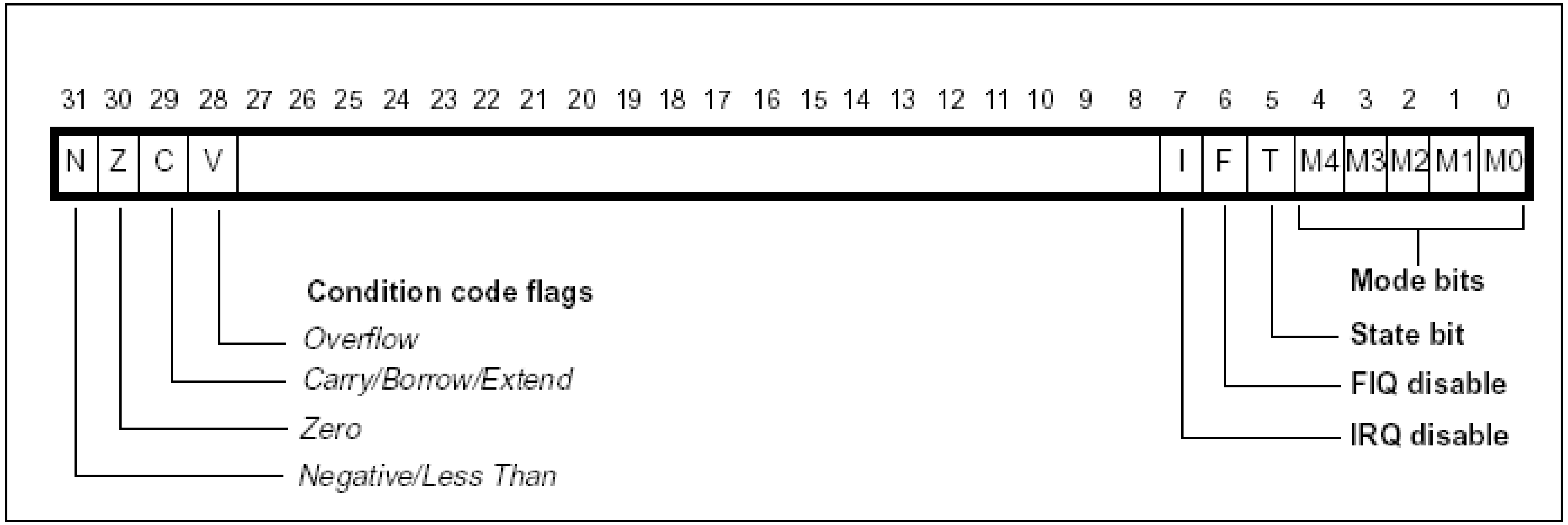
‡ Monitor mode and the associated banked registers are implemented only as part of the Security Extensions

Current Program Status Register (CPSR)

- CPSR is a 32-bit wide register used in the ARM architecture to record various pieces of information regarding the state of the program being executed by the processor and the state of the processor.
- This information is recorded by setting or clearing specific bits in the register.
- The top four bits (bits 31, 30, 29, and 28) are the condition code (cc) bits and are of most interest to us. Condition code bits are sometimes referred to as "flags".
- The lowest 8 bits (bit 7 through to bit 0) store information about the processor's own state.
- The remaining bits (i.e. bit 27 to bit 8) are currently unused in most ARM processors.

Current Program Status Register (CPSR)

- N - the result was negative
- Z - the result was zero
- C - the result produced a carry out
- V - the result generated an arithmetic overflow
- I, F – interrupt enable bits
- T – instruction set (Thumb/ARM)
- In user programs only the top 4 bits of the CPSR are relevant



ARM instruction set architecture (Version 1)

- This version was implemented by ARM 1 and was never used in a commercial product.
- It had only 26-bit address space and is now obsolete.
- It contained:
  - The basic data processing instructions (not including multiplies)
  - Byte, word, and multi-word LOAD / STORE instructions
  - Branch instructions, including a branch-and-link instruction designed for subroutine calls
  - A software interrupt instruction, for use in making Operating System calls

ARM instruction set architecture (Version 2)

- This version extended the Version 1 architecture by adding:
  - Multiply and multiply-accumulate instructions
  - Coprocessor support
  - Two more banked registers in fast interrupt mode
  - Atomic load-and-store instructions called SWP and SWPB (in a slightly variant version called version 2a)
- Version 2 and 2a still only had a 26-bit address space and are now obsolete

ARM instruction set architecture (Version 3)

- Extended the addressing range to 32-bits
  - Program Status information which was stored in R15 previously is now been stored in the Current Program status Register (CPSR) and Saved Program Status Registers (SPSRs) to preserve the CPSR contents when an exception occurs.
- The following changes occurred to the instruction set:
  - two instructions (MRS and MSR) were added to allow the new CPSR and SPSRs to be accessed
  - the functionality of instructions previously used to return from exceptions was modified to allow them to continue to be used for that purpose
- Two new processor modes were added to use Data Abort, Prefetch Abort and undefined Instructions exceptions effectively in Operating System codes

ARM instruction set architecture (Version 4)

- This version added the following to the architecture Version 3:
  - Halfword load/store instructions
  - Instructions to load and sign-extend bytes and halfwords
  - In T variants , an instruction to transfer to Thumb state
  - A new privileged processor mode that uses the User mode registers.
- Version 4 also made it clearer which instructions should cause the undefined Instruction exception to be taken.

ARM instruction set architecture (Version 5)

- This version added some new instructions and modified the definitions of some of the instructions of Version 4 to:
  - Improve the efficiency of ARM/Thumb ineterworking in T variants
  - Allow the same code generation techniques to be used for non-T variants as for T variants
- Version 5 also:
  - Adds a count leading zeros instruction, which (among other things) allows more efficient integer divide and interrupt prioritization routine
  - Adds a software breakpoint instruction
  - Adds more instruction options for coprocessors designers
  - Tightens the definitions of how flags are set by multiply instructions

ARM instruction set architecture (Version 6)

- Key ARMv6 Improvements:
    - Memory Management
    - Multiprocessing
    - Multimedia Support
    - Data Handling
    - Exceptions and Interrupts

The Thumb Instruction Set (T Variants)

- Thumb Instruction Set are:
  - Introduced with architecture version 4
  - Re-encoded subset of ARM instruction set
  - Half the size of ARM instructions (16-bits compared with 32), hence greater code density
- Limitations:
  - Thumb code usually uses more instructions for the same job, so ARM code is usually best for maximizing the performance of time-critical code
  - The Thumb instruction set does not include some instructions that are needed for exception handling, so ARM code needs to be used for at least top-level exception handlers (Due to this reason Thumb Instruction is used in conjunction with a suitable ARM instruction set)

Classification of ARM Instruction set

- Branch instructions
- Data processing instructions
- Status register transfer instructions
- Load and store instructions
- Coprocessor instructions
- Exceptions-generating instructions

Advanced Microcontroller Bus Architecture (AMBA)

- AMBA (Advanced Microcontroller Bus Architecture) protocols are an open standard, on-chip interconnect specification for the connection and management of functional blocks in a System-on-Chip (SoC).
- It facilitates right-first-time development of multi-processor designs with large numbers of controllers and peripherals.

ARM vs. x86

- ARM processors require significantly fewer transistors than typical PC processors because of the fact that it has a RISC-based design.
- Fewer transistors minimizes power use, heat and production cost. All qualities that are preferred for battery-powered devices such as laptops, tablets, and smartphones.
- On the other hand x86 processors usually consumes a lot of energy but the are also a lot faster.
- This makes x86 based processors ideal for desktops, gaming and super computer that require speed more than energy efficiency.

ARM vs. x86

- The main difference between ARM and x86 architecture is that ARM is RISC based while x86 is CISC based.
- CISC design is to execute multiple complex (larger) instructions.
- While the RISC design is perfect for small, simple instructions.
- The ARM has a lot more registers than x86.
- The ARM has a thumb mode to increase code density so programs fit in less memory.
- All these features help ARM save power almost everywhere it can.

Resources

- The ARM University Program, ARM Architecture Fundamentals
  - https://www.youtube.com/watch?list=PLqsfB23JsD0FUtaDmaMskIW1wRtFLjmTu&v=7LqPJGnBPMM
- ARM lectures by Dr. Santanu Chaudhury, EE Department, IIT Delhi
  - http://www.youtube.com/watch?v=4VRtujwa_b8&playnext=1&list=PL95AFA4ABA8B28627&feature=results_main
- The ARM Instruction Set Architecture
  - http://users.ece.utexas.edu/~valvano/EE345M/Arm_EE382N_4.pdf