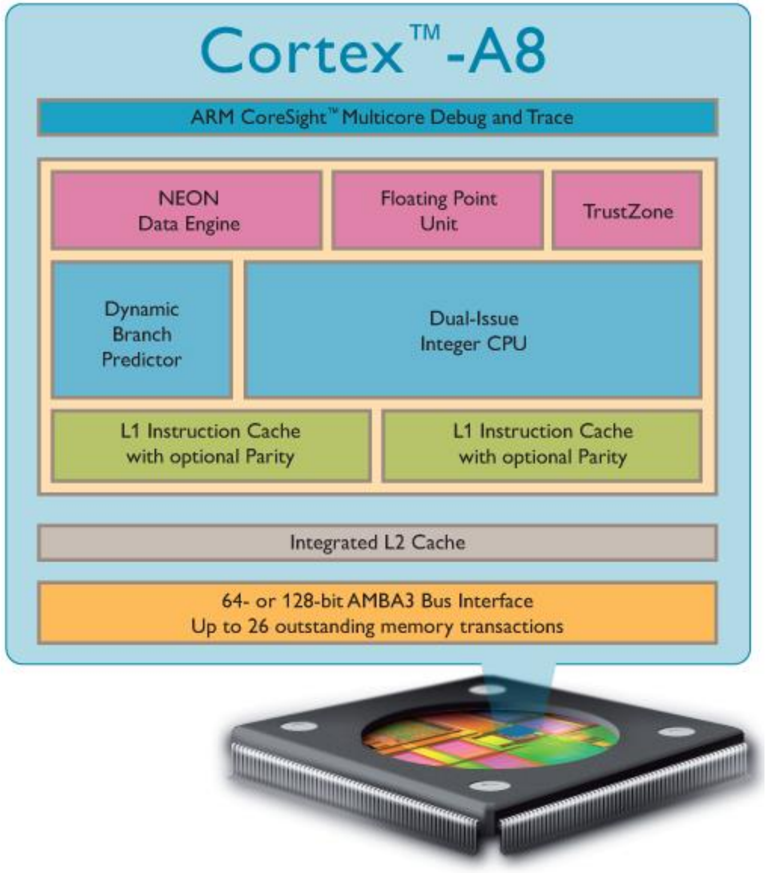
ECEN 5803

Mastering Embedded Systems Architecture

Be Boulder.

University of Colorado Boulder

Cortex-A8



Cortex™-A8
ARM CoreSight™ Multicore Debug and Trace
NEON Data Engine
Floating Point Unit
TrustZone
Dynamic Branch Predictor
Dual-Issue Integer CPU
L1 Instruction Cache with optional Parity
L1 Instruction Cache with optional Parity
Integrated L2 Cache
64- or 128-bit AMBA3 Bus Interface
Up to 26 outstanding memory transactions

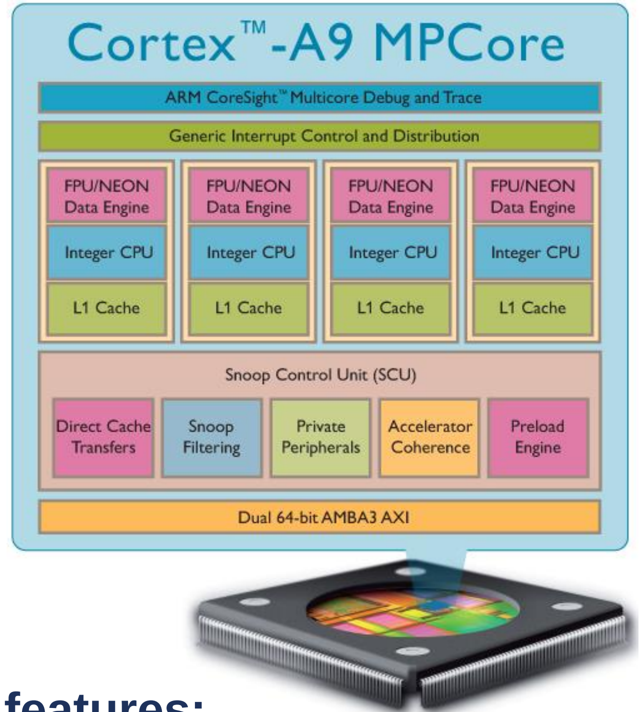
- **ARMv7-A Architecture**
  - Thumb-2
  - Thumb-2EE (Jazelle-RCT)
  - TrustZone extensions
- **Custom or synthesized design**
- **MMU**
- **64-bit or 128-bit AXI Interface**
- **L1 caches**
  - 16 or 32KB each
- **Unified L2 cache**
  - 0-2MB in size
  - 8-way set-associative

- **Optional features**
  - VFPv3 Vector Floating-Point
  - NEON media processing engine

- **Dual-issue, super-scalar 13-stage pipeline**
  - Branch Prediction & Return Stack
  - NEON and VFP implemented at end of pipeline

Cortex-A9

- **ARMv7-A Architecture**
  - Thumb-2, Thumb-2EE
  - TrustZone support
- **Variable-length Multi-issue pipeline**
  - Register renaming
  - Speculative data prefetching
  - Branch Prediction & Return Stack
- **64-bit AXI instruction and data interfaces**
- **TrustZone extensions**
- **L1 Data and Instruction caches**
  - 16-64KB each
  - 4-way set-associative



Cortex™-A9 MPCore

ARM CoreSight™ Multicore Debug and Trace

Generic Interrupt Control and Distribution

FPU/NEON Data Engine | FPU/NEON Data Engine | FPU/NEON Data Engine | FPU/NEON Data Engine

Integer CPU | Integer CPU | Integer CPU | Integer CPU

L1 Cache | L1 Cache | L1 Cache | L1 Cache

Snoop Control Unit (SCU)

Direct Cache Transfers | Snoop Filtering | Private Peripherals | Accelerator Coherence | Preload Engine

Dual 64-bit AMBA3 AXI

- **Optional features:**
  - PTM instruction trace interface
  - IEM power saving support
  - Full Jazelle DBX support
  - VFPv3-D16 Floating-Point Unit (FPU) or NEON™ media processing engine

CortexA9 Microarchitecture

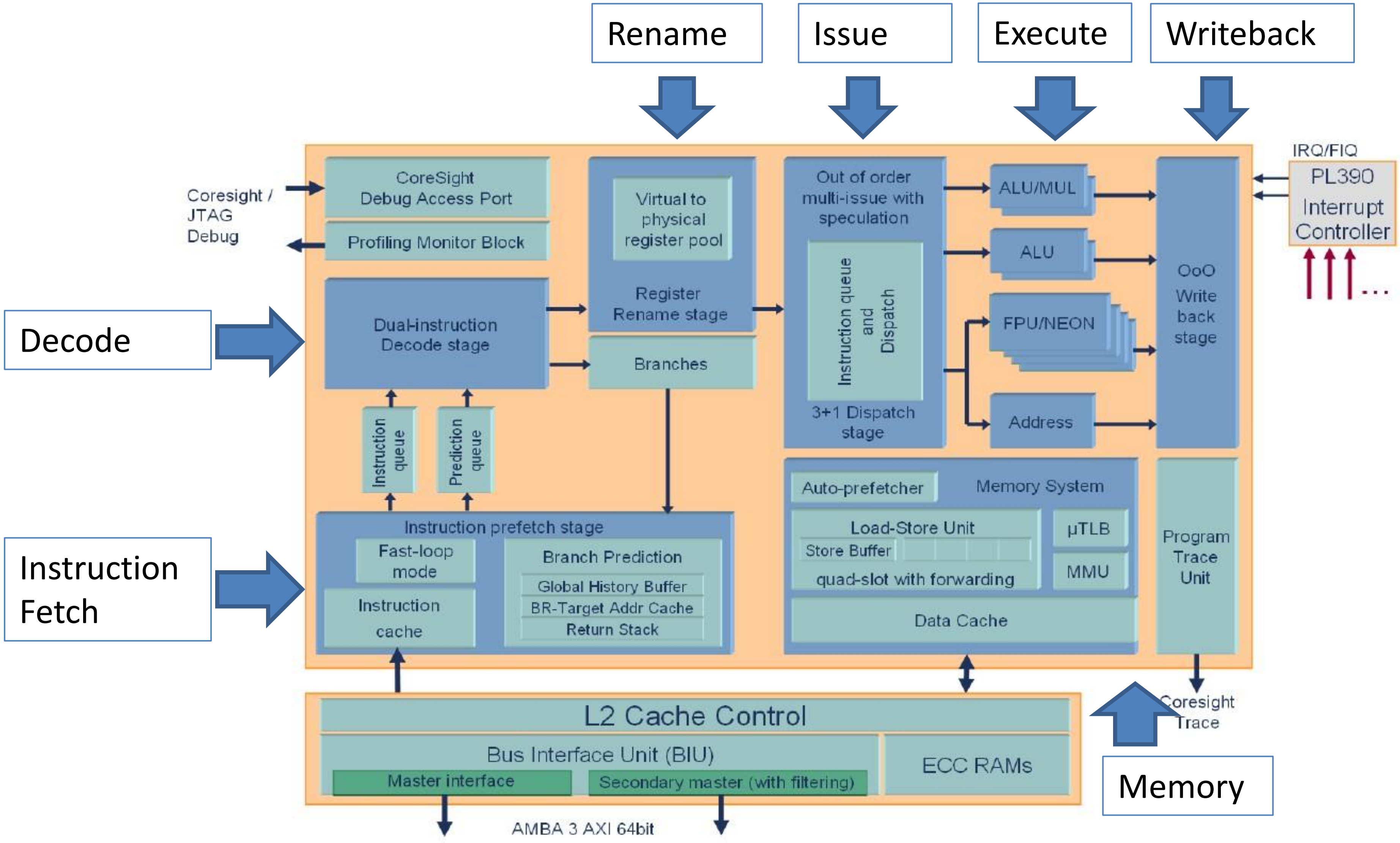
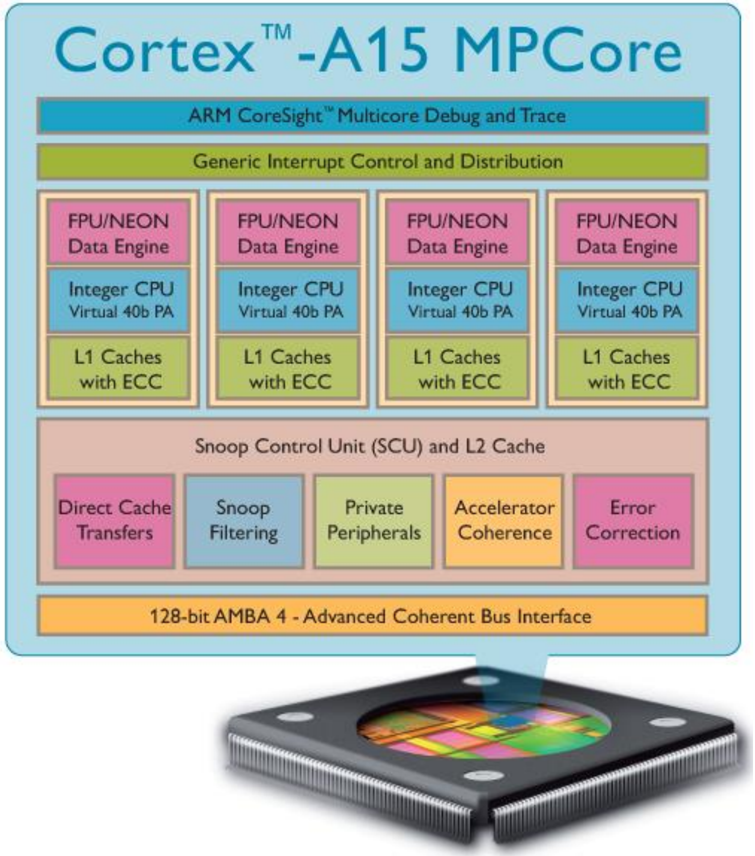


Fig. 1 Cortex-A9 microarchitecture structure and the single core interfaces.

www.arm.com/files/pdf/armcortexa-9processors.pdf

Cortex-A15 MPCore

- **1-4 processors per cluster**
- **Fixed size L1 caches (32KB)**
- **Integrated L2 Cache**
  - 512KB – 4MB
- **System-wide coherency support with AMBA 4 ACE**
- **Backward-compatible with AXI3 interconnect**
- **Integrated Interrupt Controller**
  - 0-224 external interrupts for entire cluster
- **CoreSight debug**
- **Advanced Power Management**
- **Large Physical Address Extensions (LPAE) to ARMv7-A Architecture**
- **Virtualization Extensions to ARMv7-A Architecture**



Cortex™-A15 MPCore
ARM CoreSight™ Multicore Debug and Trace
Generic Interrupt Control and Distribution
FPU/NEON Data Engine
Integer CPU Virtual 40b PA
L1 Caches with ECC
FPU/NEON Data Engine
Integer CPU Virtual 40b PA
L1 Caches with ECC
FPU/NEON Data Engine
Integer CPU Virtual 40b PA
L1 Caches with ECC
FPU/NEON Data Engine
Integer CPU Virtual 40b PA
L1 Caches with ECC
Snoop Control Unit (SCU) and L2 Cache
Direct Cache Transfers
Snoop Filtering
Private Peripherals
Accelerator Coherence
Error Correction
128-bit AMBA 4 - Advanced Coherent Bus Interface

Data Sizes and Instruction Sets

- **ARM is a 32-bit load / store RISC architecture**
  - The only memory accesses allowed are loads and stores
  - Most internal registers are 32 bits wide
  - Most instructions execute in a single cycle

- **When used in relation to ARM cores**
  - **Halfword** means 16 bits (two bytes)
  - **Word** means 32 bits (four bytes)
  - **Doubleword** means 64 bits (eight bytes)

- **ARM cores implement two basic instruction sets**
  - ARM instruction set – instructions are all 32 bits long
  - Thumb instruction set – instructions are a mix of 16 and 32 bits
    - Thumb-2 technology added many extra 32- and 16-bit instructions to the original 16-bit Thumb instruction set

- **Depending on the core, may also implement other instruction sets**
  - VFP instruction set – 32 bit (vector) floating point instructions
  - NEON instruction set – 32 bit SIMD instructions
  - Jazelle-DBX - provides acceleration for Java VMs (with additional software support)
  - Jazelle-RCT - provides support for interpreted languages

Processor Modes

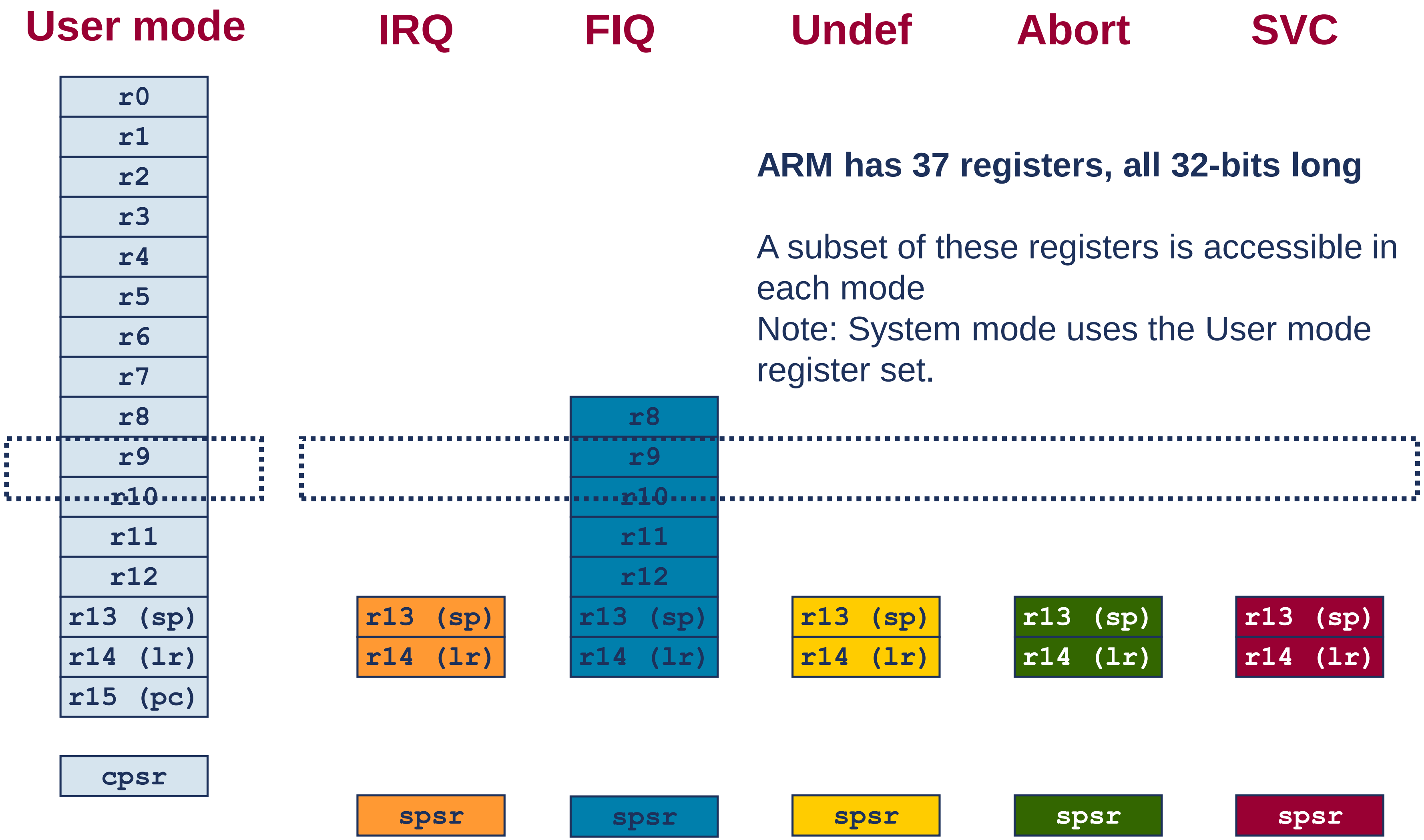
- **ARM has seven basic operating modes**
  - Each mode has access to its own stack space and a different subset of registers
  - Some operations can only be carried out in a privileged mode

	Mode	Description	
Exception modes	**Supervisor (SVC)**	Entered on reset and when a Supervisor call instruction (SVC) is executed	**Privileged modes**
	FIQ	Entered when a high priority (fast) interrupt is raised	
	IRQ	Entered when a normal priority interrupt is raised	
	Abort	Used to handle memory access violations	
	Undef	Used to handle undefined instructions	
	System	Privileged mode using the same registers as User mode	
	User	Mode under which most Applications / OS tasks run	**Unprivileged mode**

The ARM Register Set

ARM has 37 registers, all 32-bits long

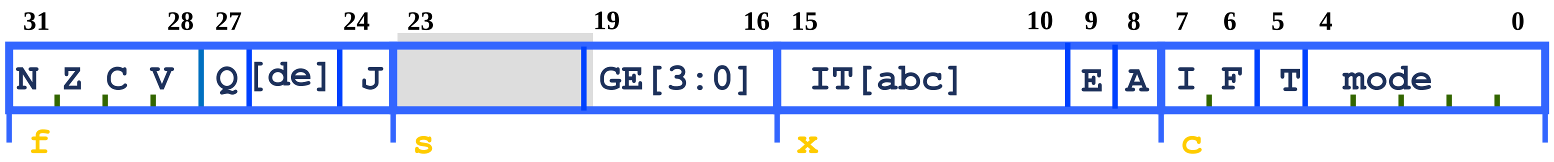
A subset of these registers is accessible in each mode
Note: System mode uses the User mode register set.



User mode	IRQ	FIQ	Undef	Abort	SVC
r0					
r1					
r2					
r3					
r4					
r5					
r6					
r7					
r8		r8			
r9		r9			
r10		r10			
r11		r11			
r12		r12			
r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)
r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)
r15 (pc)					
cpsr	spsr	spsr	spsr	spsr	spsr

Current mode **Banked out registers**

Program Status Registers



- **Condition code flags**
  - N = Negative result from ALU
  - Z = Zero result from ALU
  - C = ALU operation Carried out
  - V = ALU operation oVerflowed
- **Sticky Overflow flag - Q flag**
  - Indicates if saturation has occurred
- **SIMD Condition code bits – GE[3:0]**
  - Used by some SIMD instructions
- **IF THEN status bits – IT[abcde]**
  - Controls conditional execution of Thumb instructions
- **T bit**
  - T = 0: Processor in ARM state
  - T = 1: Processor in Thumb state
- **J bit**
  - J = 1: Processor in Jazelle state
- **Mode bits**
  - Specify the processor mode
- **Interrupt Disable bits**
  - I = 1: Disables IRQ
  - F = 1: Disables FIQ
- **E bit**
  - E = 0: Data load/store is little endian
  - E = 1: Data load/store is bigendian
- **A bit**
  - A = 1: Disable imprecise data aborts

Instruction Set basics

- **The ARM Architecture is a Load/Store architecture**
  - No direct manipulation of memory contents
  - Memory must be loaded into the CPU to be modified, then written back out
- **Cores are either in ARM *state* or Thumb *state***
  - This determines which instruction set is being executed
  - An instruction must be executed to switch between states
- **The architecture allows programmers and compilation tools to reduce branching through the use of conditional execution**
  - Method differs between ARM and Thumb, but the principle is that most (ARM) or all (Thumb) instructions can be executed conditionally.

Data Processing Instructions

- **These instructions operate on the contents of registers**
  - They DO NOT affect memory

	arithmetic		logical		move
manipulation (has destination register)	ADC ADD	SBC SUB RSB RSC	BIC AND	ORR EOR ORN	MVN MOV
comparison (set flags only)	CMN (ADDS)	CMP (SUBS)	TST (ANDS)	TEQ (EORS)	

- **Syntax:**

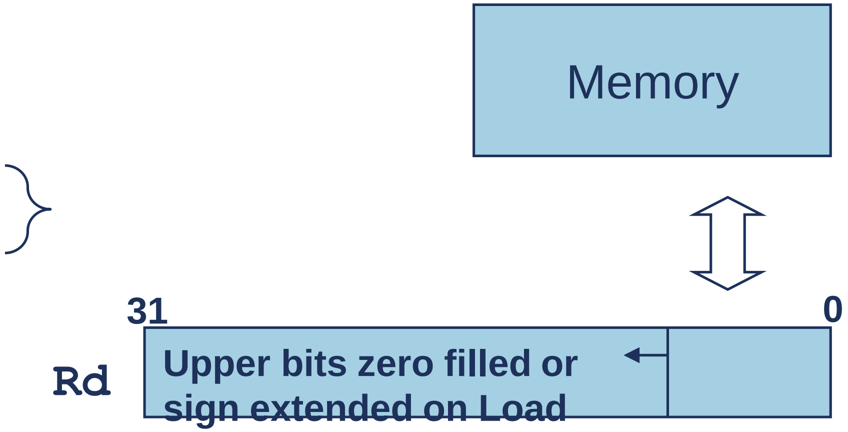
```
<Operation>{<cond>}{S} {Rd,} Rn, Operand2
```

- **Examples:**
  - `ADD r0, r1, r2        ; r0 = r1 + r2`
  - `TEQ r0, r1            ; if r0 = r1, Z flag will be set`
  - `MOV r0, r1            ; copy r1 to r0`

Single Access Data Transfer

- **Use to move data between one or two registers and memory**

LDRD	STRD	Doubleword
LDR	STR	Word
LDRB	STRB	Byte
LDRH	STRH	Halfword
LDRSB		Signed byte load
LDRSH		Signed halfword load



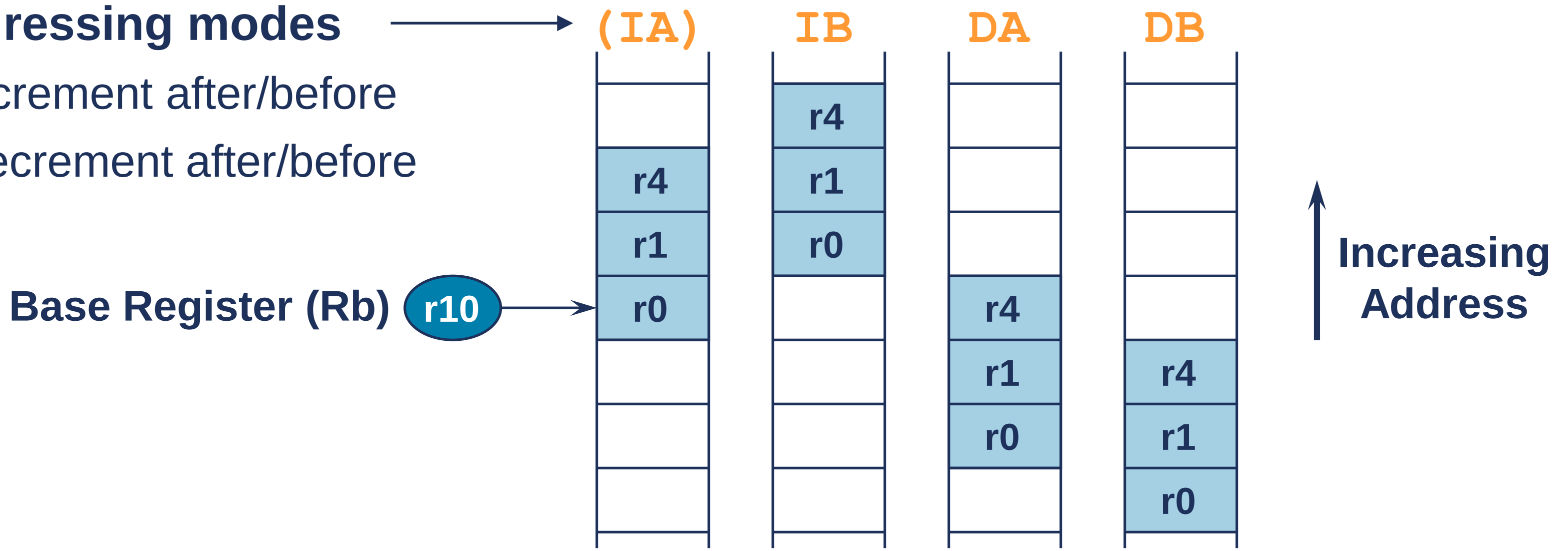
- **Syntax:**
  - `LDR{<size>}{<cond>} Rd, <address>`
  - `STR{<size>}{<cond>} Rd, <address>`

- **Example:**

  -
    ```
    LDRB r0, [r1]       ; load bottom byte of r0 from the
                        ; byte of memory at address in r1
    ```

Multiple Register Data Transfer

- These instructions move data between multiple registers and memory
- Syntax
  - `<LDM|STM>{<addressing_mode>}{<cond>} Rb{!}, <register list>`
- 4 addressing modes
  - Increment after/before
  - Decrement after/before

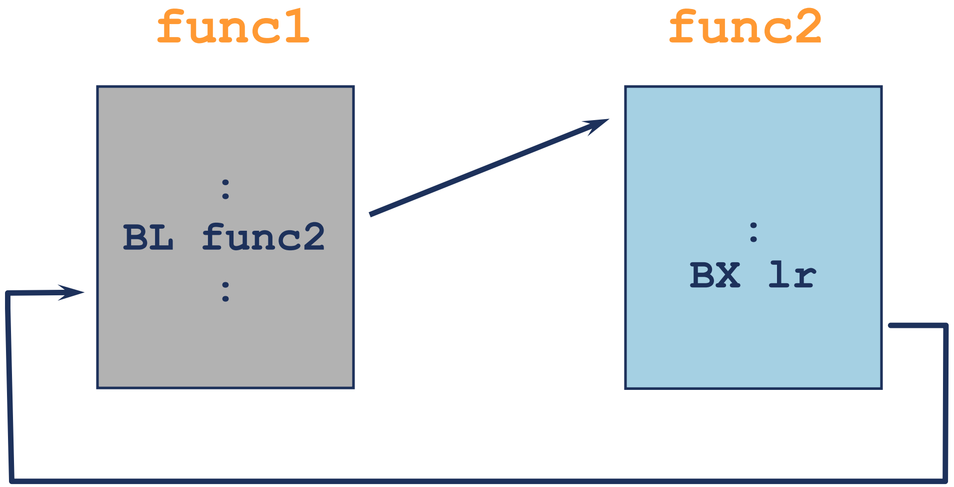


- Also
  - `PUSH/POP`, equivalent to `STMDB/LDMIA` with `SP!` as base register
- Example
  - `LDM    r10, {r0,r1,r4}     ; load registers, using r10 base`
  - `PUSH   {r4-r6,pc}          ; store registers, using SP base`

Subroutines

- **Implementing a conventional subroutine call requires two steps**
  - Store the return address
  - Branch to the address of the required subroutine
- **These steps are carried out in one instruction, `BL`**
  - The return address is stored in the link register (`lr/r14`)
  - Branch to an address (range dependent on instruction set and width)
- **Return is by branching to the address in `lr`**

```
void func1 (void)
{
        :
        func2();
        :
}
```



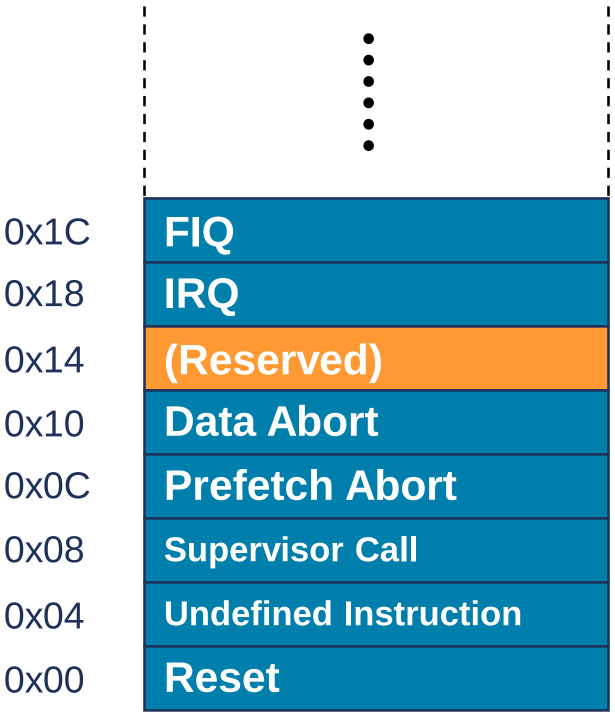
Supervisor Call (SVC)

```
SVC{<cond>} <SVC number>
```

- **Causes an SVC exception**
- **The SVC handler can examine the SVC number to decide what operation has been requested**
  - But the core ignores the SVC number
- **By using the SVC mechanism, an operating system can implement a set of privileged operations (system calls) which applications running in user mode can request**
- **Thumb version is unconditional**

Exception Handling

- **When an exception occurs, the core…**
  - Copies CPSR into SPSR_<mode>
  - Sets appropriate CPSR bits
    - Change to ARM state (if appropriate)
    - Change to exception mode
    - Disable interrupts (if appropriate)
  - Stores the return address in LR_<mode>
  - Sets PC to vector address
- **To return, exception handler needs to…**
  - Restore CPSR from SPSR_<mode>
  - Restore PC from LR_<mode>
- **Cores can enter ARM state or Thumb state when taking an exception**
  - Controlled through settings in CP15
- **Note that v7-M and v6-M exception model is different**

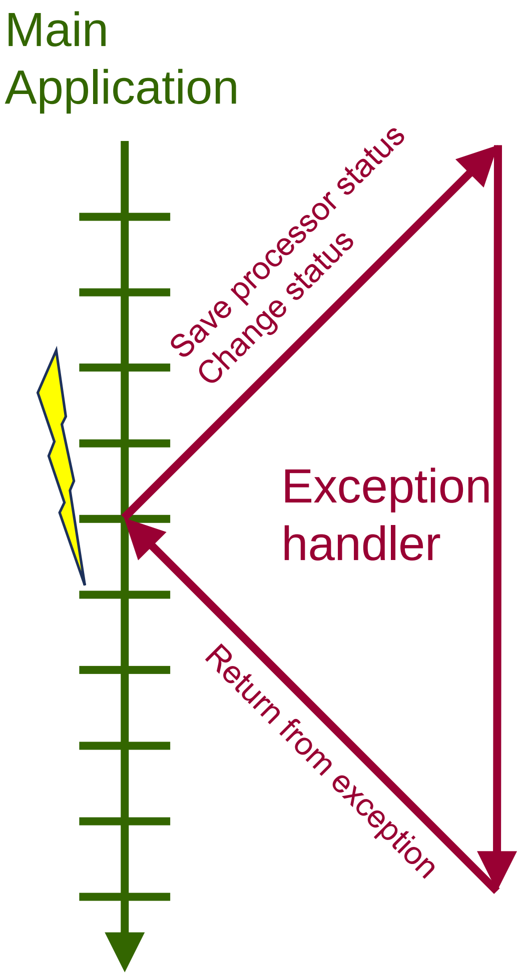


0x1C	FIQ
0x18	IRQ
0x14	(Reserved)
0x10	Data Abort
0x0C	Prefetch Abort
0x08	Supervisor Call
0x04	Undefined Instruction
0x00	Reset

Vector Table

Vector table can also be at `0xFFFF0000` on most cores

Exception handling process

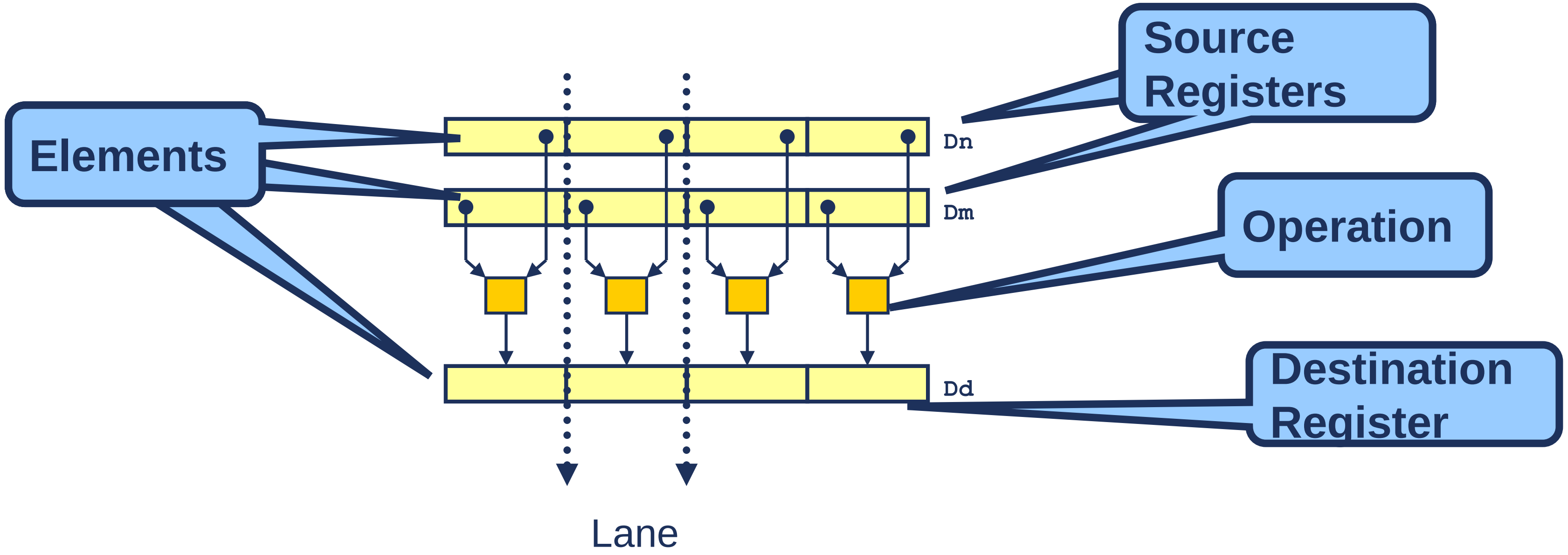


1. **Save processor status**
   - Copies `CPSR` into `SPSR_<mode>`
   - Stores the return address in `LR_<mode>`
   - Adjusts LR based on exception type
2. **Change processor status for exception**
   - Mode field bits
   - ARM or Thumb state
   - Interrupt disable bits (if appropriate)
   - Sets PC to vector address
3. **Execute exception handler**
   - <users code>
4. **Return to main application**
   - Restore `CPSR` from `SPSR_<mode>`
   - Restore PC from `LR_<mode>`

- **1 and 2 performed automatically by the core**
- **3 and 4 responsibility of software**

What is NEON?

- **NEON is a wide SIMD data processing architecture**
  - Extension of the ARM instruction set (v7-A)
  - 32 x 64-bit wide registers (can also be used as 16 x 128-bit wide registers)
- **NEON instructions perform “Packed SIMD” processing**
  - Registers are considered as **vectors** of **elements** of the same data type
  - Data types available: signed/unsigned 8-bit, 16-bit, 32-bit, 64-bit, single prec. float
  - Instructions usually perform the same operation in all **lanes**

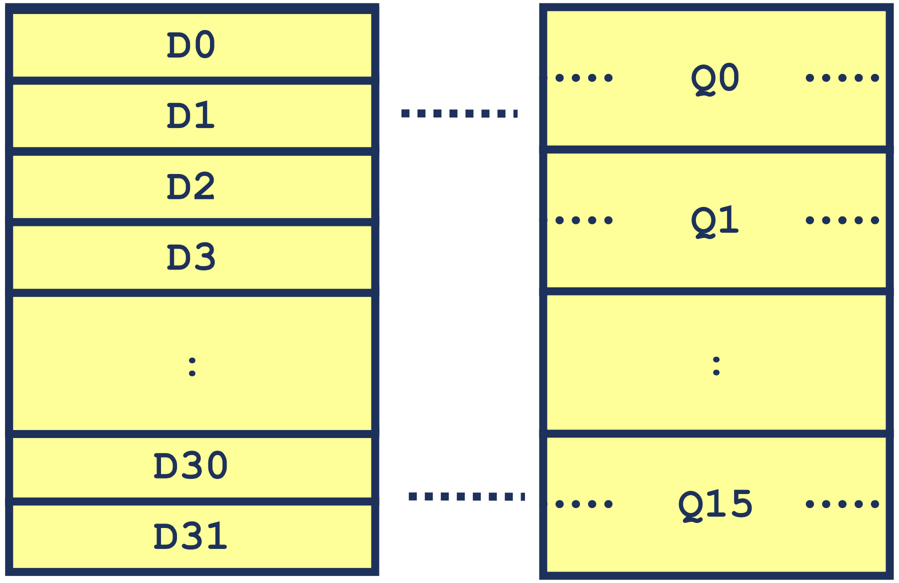


NEON Coprocessor registers

- **NEON has a 256-byte register file**
  - Separate from the core registers (r0-r15)
  - Extension to the VFPv2 register file (VFPv3)

- **Two different views of the NEON registers**
  - 32 x 64-bit registers (D0-D31)
  - 16 x 128-bit registers (Q0-Q15)

- **Enables register trade-offs**
  - Vector length can be variable
  - Different registers available



NEON vectorizing example

- **How does the compiler perform vectorization?**

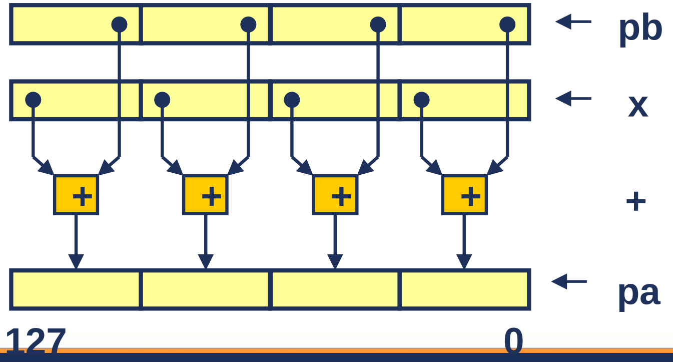
```
void add_int(int * __restrict pa,
             int * __restrict pb,
             unsigned int n, int x)
{
  unsigned int i;
  for(i = 0; i < (n & ~3); i++)
    pa[i] = pb[i] + x;
}
```

1. Analyze each loop:
   - Are pointer accesses safe for vectorization?
   - What data types are being used? How do they map onto NEON vector registers?
   - Number of loop iterations

2. Unroll the loop to the appropriate number of iterations, and perform other transformations like pointerization

```
void add_int(int *pa, int *pb,
             unsigned n, int x)
{
  unsigned int i;
  for (i = ((n & ~3) >> 2); i; i--)
  {
    *(pa + 0) = *(pb + 0) + x;
    *(pa + 1) = *(pb + 1) + x;
    *(pa + 2) = *(pb + 2) + x;
    *(pa + 3) = *(pb + 3) + x;
    pa += 4; pb += 4;
  }
}
```

3. Map each unrolled operation onto a NEON vector lane, and generate corresponding NEON instructions



Memory Types

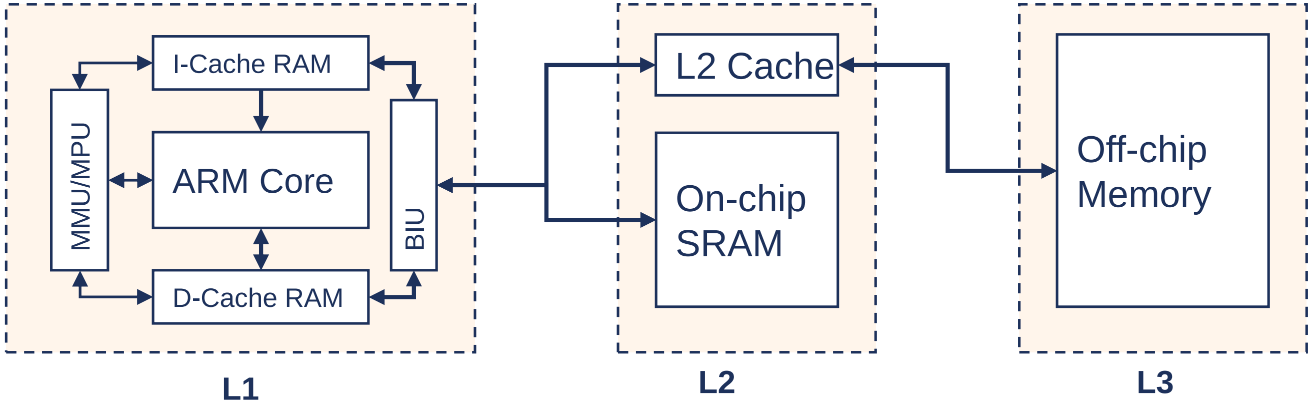
- **Each defined memory region will specify a memory type**

- **The memory type controls the following:**
  - Memory access ordering rules
  - Caching and buffering behaviour

- **There are 3 mutually exclusive memory types:**
  - Normal
  - Device
  - Strongly Ordered

- **Normal and Device memory allow additional attributes for specifying**
  - The cache policy
  - Whether the region is Shared
  - Normal memory allows you to separately configure Inner and Outer cache policies (discussed in the Caches and TCMs module)

L1 and L2 Caches

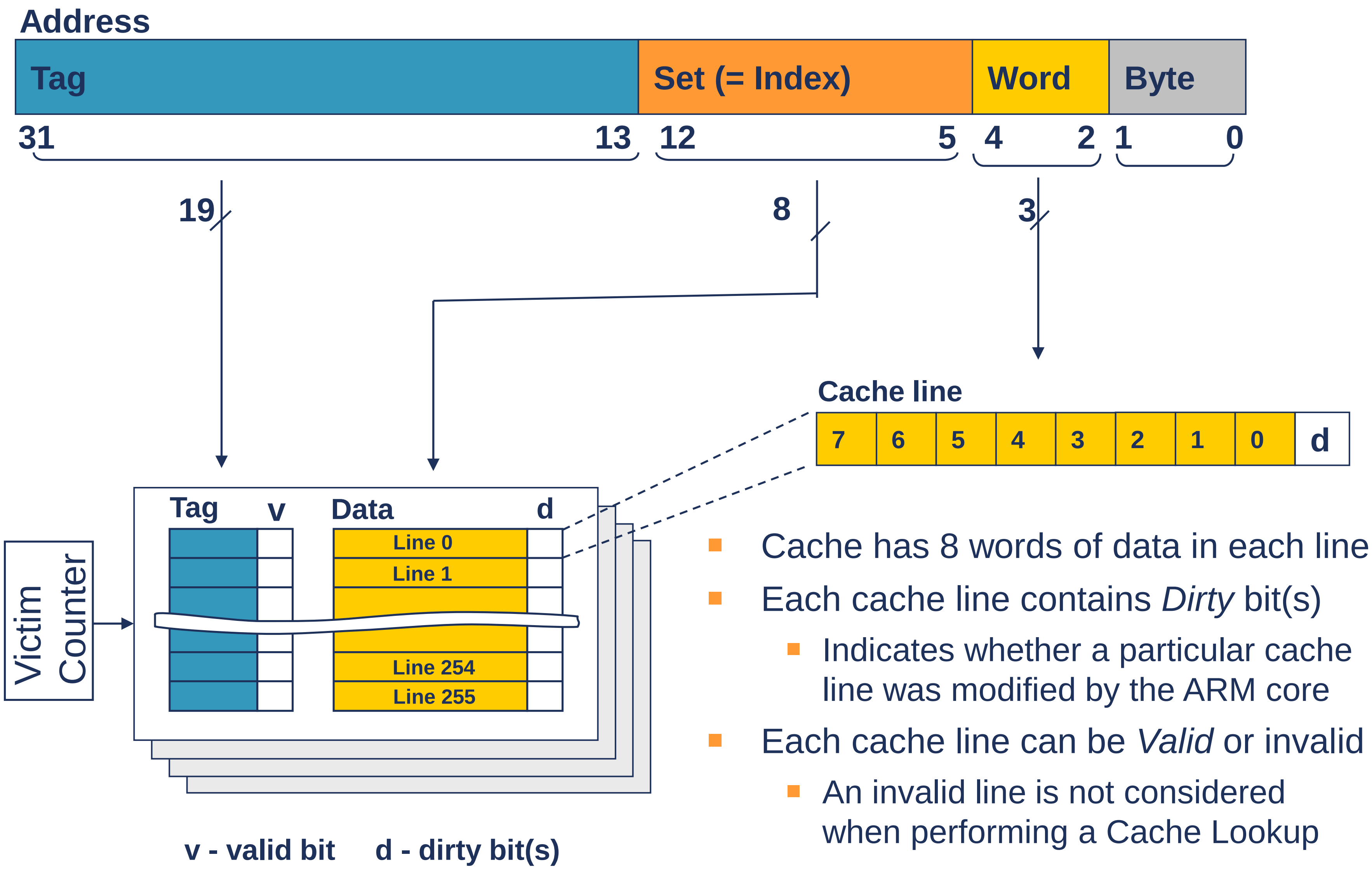


- **Typical memory system can have multiple levels of cache**
  - Level 1 memory system typically consists of L1-caches, MMU/MPU and TCMs
  - Level 2 memory system (and beyond) depends on the system design
- **Memory attributes determine cache behavior at different levels**
  - Controlled by the MMU/MPU (discussed later)
  - Inner Cacheable attributes define memory access behavior in the L1 memory system
  - Outer Cacheable attributes define memory access behavior in the L2 memory system (if external) and beyond (as signals on the bus)
- **Before caches can be used, software setup must be performed**

ARM Cache Features

- **Harvard Implementation for L1 caches**
  - Separate Instruction and Data caches
- **Cache Lockdown**
  - Prevents line Eviction from a specified Cache Way (discussed later)
- **Pseudo-random and Round-robin replacement strategies**
  - Unused lines can be allocated before considering replacement
- **Non-blocking data cache**
  - Cache Lookup can hit before a Linefill is complete (also checks Linefill buffer)
- **Streaming, Critical-Word-First**
  - Cache data is forwarded to the core as soon as the requested word is received in the Linefill buffer
  - Any word in the cache line can be requested first using a ‘WRAP’ burst on the bus
- **ECC or parity checking**

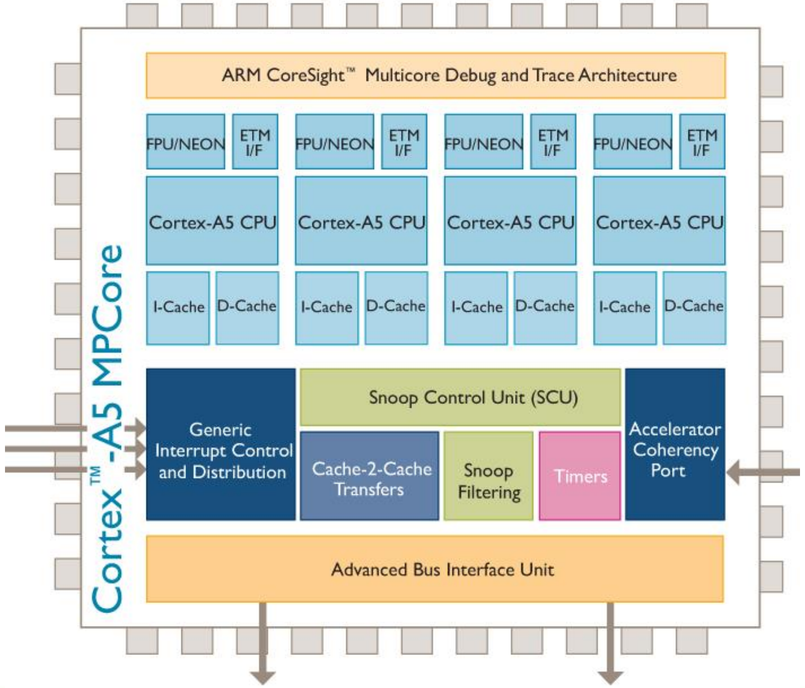
Example 32KB ARM cache

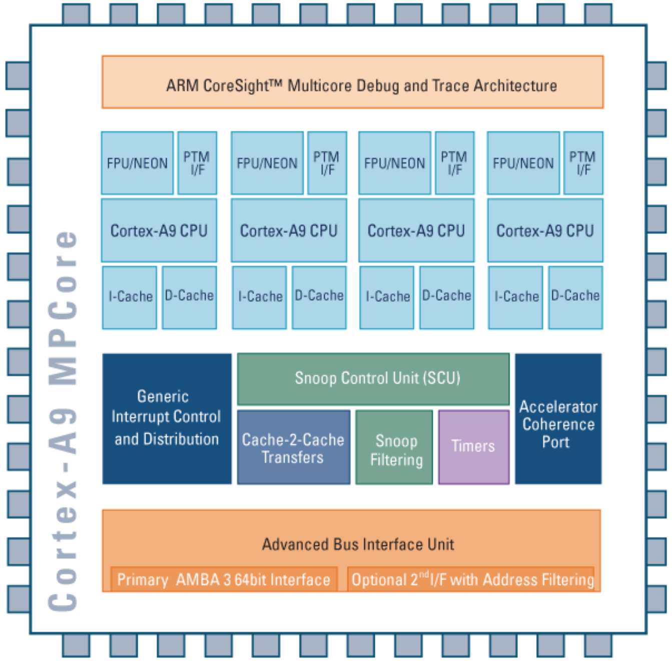


- Cache has 8 words of data in each line
- Each cache line contains *Dirty* bit(s)
  - Indicates whether a particular cache line was modified by the ARM core
- Each cache line can be *Valid* or invalid
  - An invalid line is not considered when performing a Cache Lookup

Cortex MPCore Processors

- **Standard Cortex cores, with additional logic to support MPCore**
  - Available as 1-4 CPU variants
- **Include integrated**
  - Interrupt controller
  - Snoop Control Unit (SCU)
  - Timers and Watchdogs

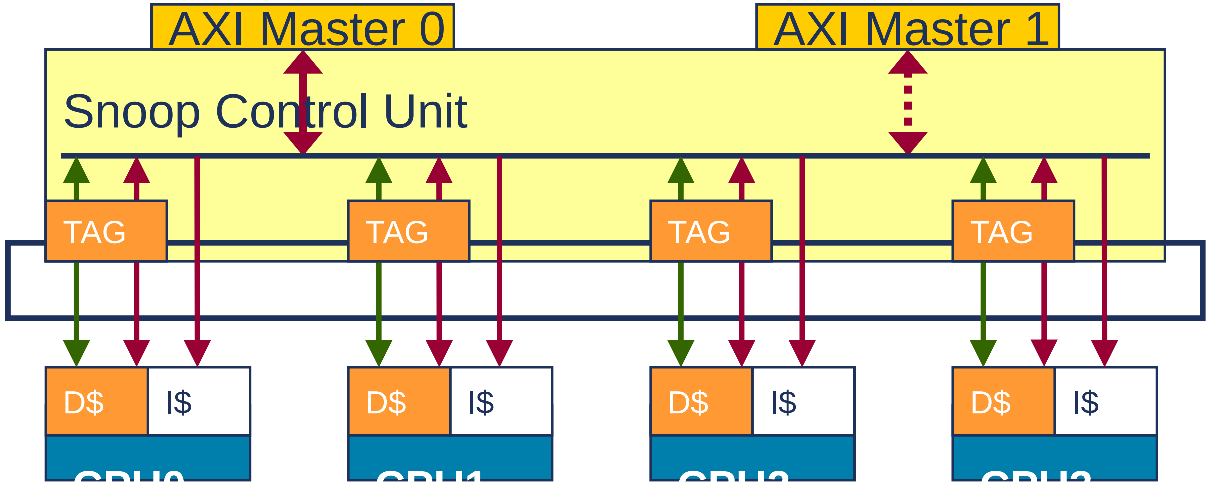




Snoop Control Unit

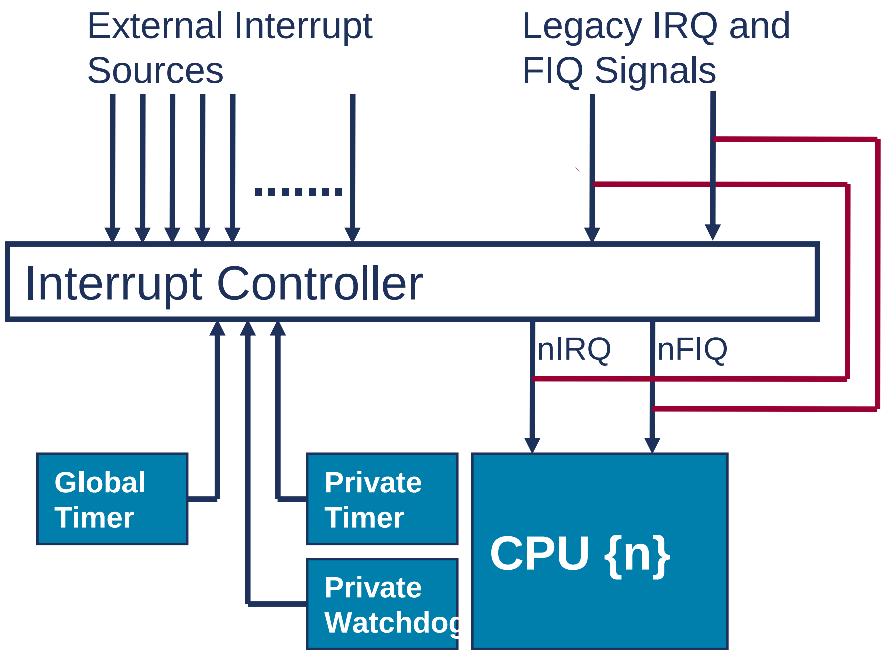
- **The Snoop Control Unit (SCU) maintains coherency between L1 data caches**
  - Duplicated Tag RAMs keep track of what data is allocated in each CPU’s cache
    - Separate interfaces into L1 data caches for coherency maintenance
  - Arbitrates accesses to L2 AXI master interface(s), for both instructions and data

- **Optionally, can use address filtering**
  - Directing accesses to configured memory range to AXI Master port 1

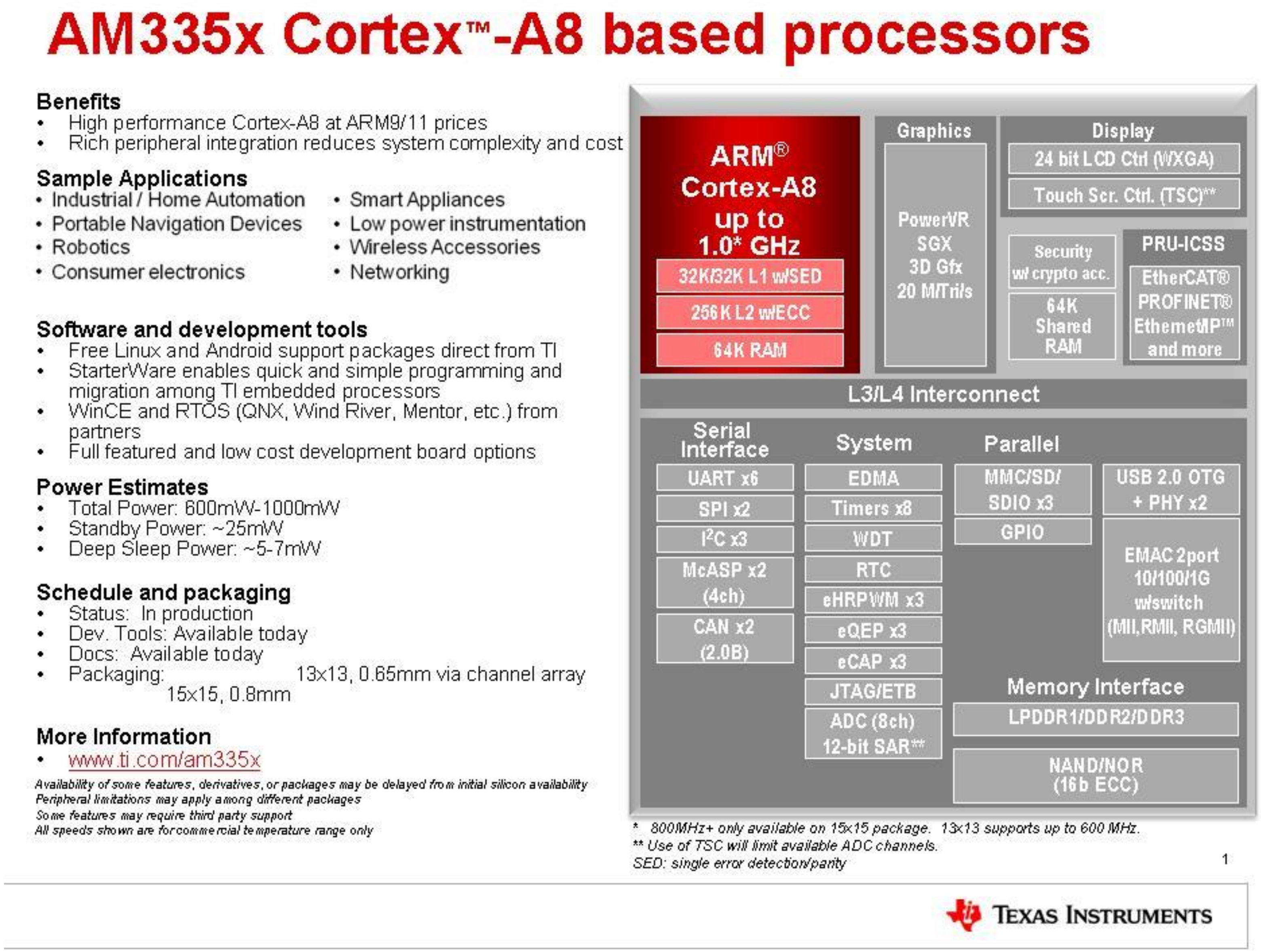


Interrupt Controller

- **MPCore processors include an integrated Interrupt Controller (IC)**
  - Implementation of the Generic Interrupt Controller (GIC) architecture
- **The IC provides:**
  - Configurable number of external interrupts (max 224)
  - Interrupt prioritization and pre-emption
  - Interrupt routing to different cores
- **Enabled per CPU**
  - When not enabled, that CPU will use legacy nIRQ[n] and nFIQ[n] signals



AM 335x



AM335x Cortex™-A8 based processors

Benefits

- High performance Cortex-A8 at ARM9/11 prices
- Rich peripheral integration reduces system complexity and cost

Sample Applications

- Industrial / Home Automation
- Portable Navigation Devices
- Robotics
- Consumer electronics
- Smart Appliances
- Low power instrumentation
- Wireless Accessories
- Networking

Software and development tools

- Free Linux and Android support packages direct from TI
- StarterWare enables quick and simple programming and migration among TI embedded processors
- WinCE and RTOS (QNX, Wind River, Mentor, etc.) from partners
- Full featured and low cost development board options

Power Estimates

- Total Power: 600mW-1000mW
- Standby Power: ~25mW
- Deep Sleep Power: ~5-7mW

Schedule and packaging

- Status: In production
- Dev. Tools: Available today
- Docs: Available today
- Packaging: 13x13, 0.65mm via channel array
  15x15, 0.8mm

More Information

- www.ti.com/am335x

Availability of some features, derivatives, or packages may be delayed from initial silicon availability
Peripheral limitations may apply among different packages
Some features may require third party support
All speeds shown are for commercial temperature range only

ARM® Cortex-A8 up to 1.0* GHz
32K/32K L1 w/SED
256K L2 w/ECC
64K RAM

Graphics
PowerVR SGX 3D Gfx 20 M/Tri/s

Display
24 bit LCD Ctrl (WXGA)
Touch Scr. Ctrl. (TSC)**

Security w/ crypto acc.
64K Shared RAM

PRU-ICSS
EtherCAT® PROFINET® EtherNet/IP™ and more

L3/L4 Interconnect

Serial Interface
UART x6
SPI x2
I^2C x3
McASP x2 (4ch)
CAN x2 (2.0B)

System
EDMA
Timers x8
WDT
RTC
eHRPWM x3
eQEP x3
eCAP x3
JTAG/ETB
ADC (8ch) 12-bit SAR**

Parallel
MMC/SD/ SDIO x3
GPIO
USB 2.0 OTG + PHY x2
EMAC 2port 10/100/1G w/switch (MII,RMII, RGMII)

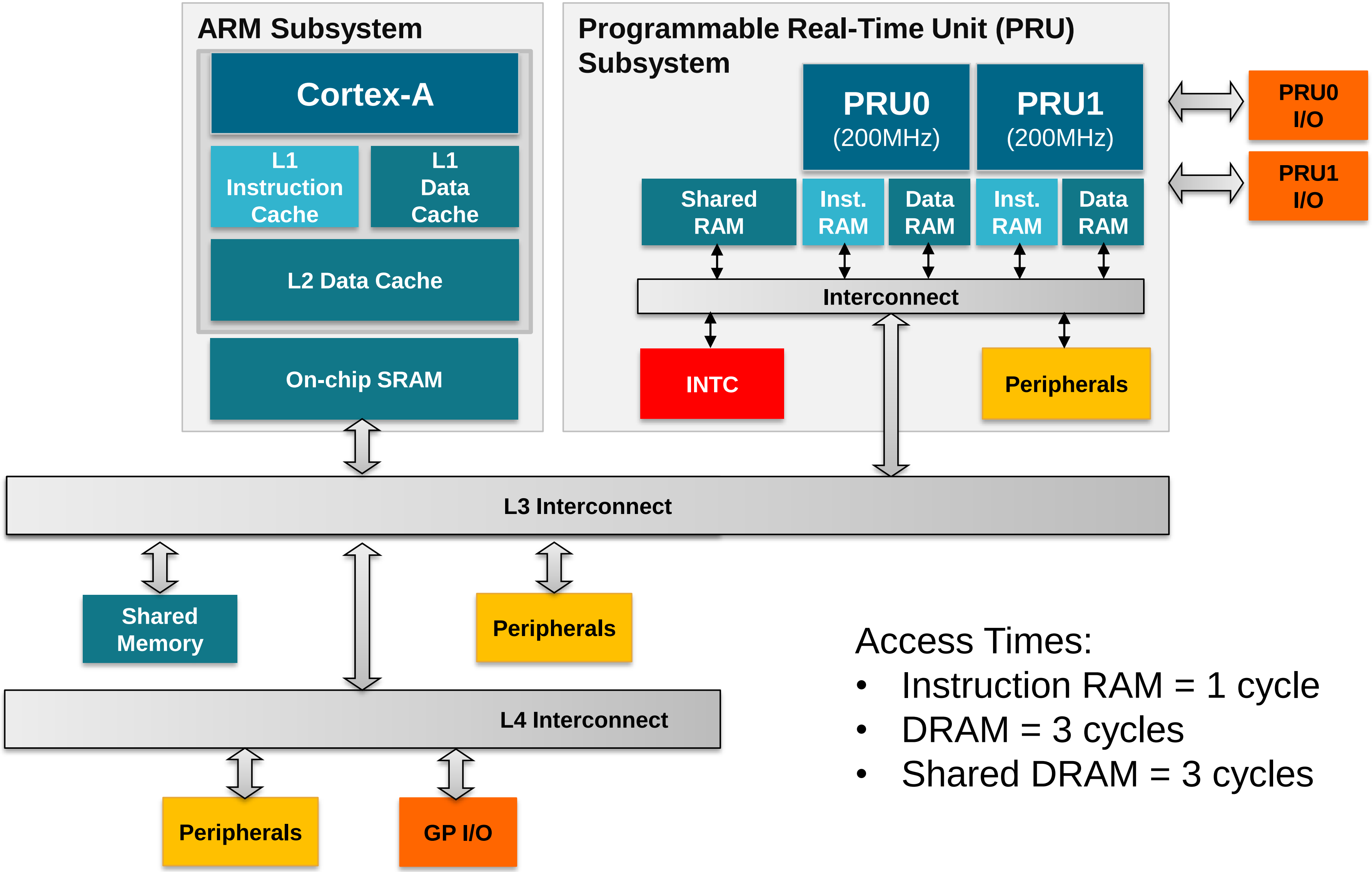
Memory Interface
LPDDR1/DDR2/DDR3
NAND/NOR (16b ECC)

* 800MHz+ only available on 15x15 package. 13x13 supports up to 600 MHz.
** Use of TSC will limit available ADC channels.
SED: single error detection/parity

1

TEXAS INSTRUMENTS

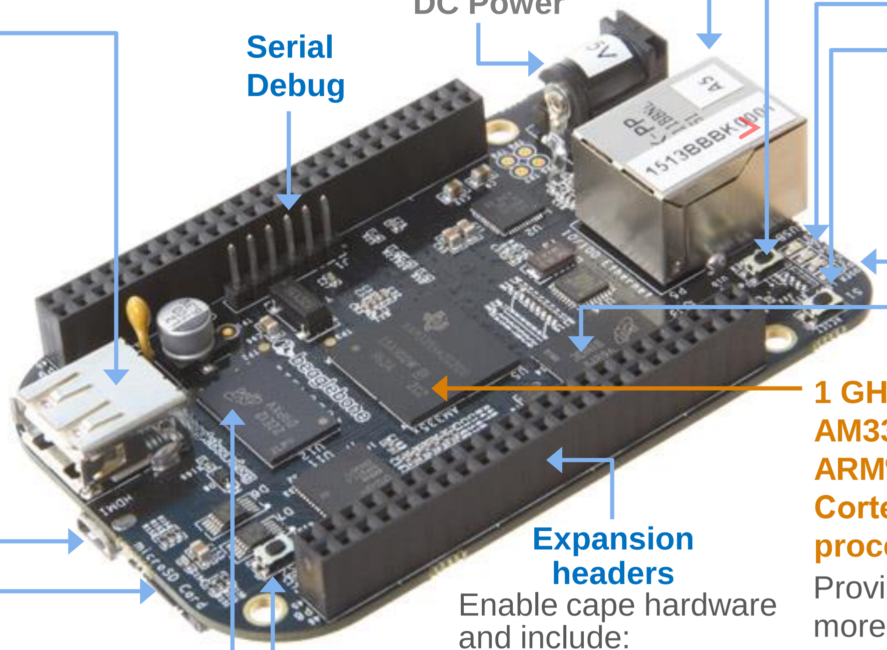
ARM + PRU SoC Architecture



Access Times:

- Instruction RAM = 1 cycle
- DRAM = 3 cycles
- Shared DRAM = 3 cycles

BeagleBone Black
1 GHz performance ready to use for $45



10/100 Ethernet

USB Host
Easily connects to almost any everyday device such as mouse or keyboard

microHDMI
Connect directly to monitors and TVs

microSD
Expansion slot for additional storage

512MB DDR3
Faster, lower power RAM for enhanced user-friendly experience

Serial Debug

DC Power

Boot Button

Expansion headers
Enable cape hardware and include:
- 65 digital I/O
- 7 analog
- 4 serial
- 2 SPI
- 2 I2C
- 8 PWMs
- 4 timers
- And much much more!

1 GHz Sitara AM335x ARM® Cortex™-A8 processor
Provides a more advanced user interface and up to 150% better performance than ARM11

Power Button

LEDS

Reset Button

USB Client
Development interface and directly powers board from PC

2GB on-board storage using eMMC
- Pre-loaded with Ångström Linux Distribution
- 8-bit bus accelerates performance
- Frees the microSD slot to be used for additional storage for a less expensive solution than SD cards

Included in price:
- **Power supply ~ $10**
- **USB network cable ~ $3**
- **2GB on-board storage $5-$10**
- **PRU for real-time tasks typically on FPGA ~ $20**

ARM Cortex–A53

Supports
A32 Instruction Set (ARM)
T32 Instruction Set (Thumb)
A64 Instruction Set

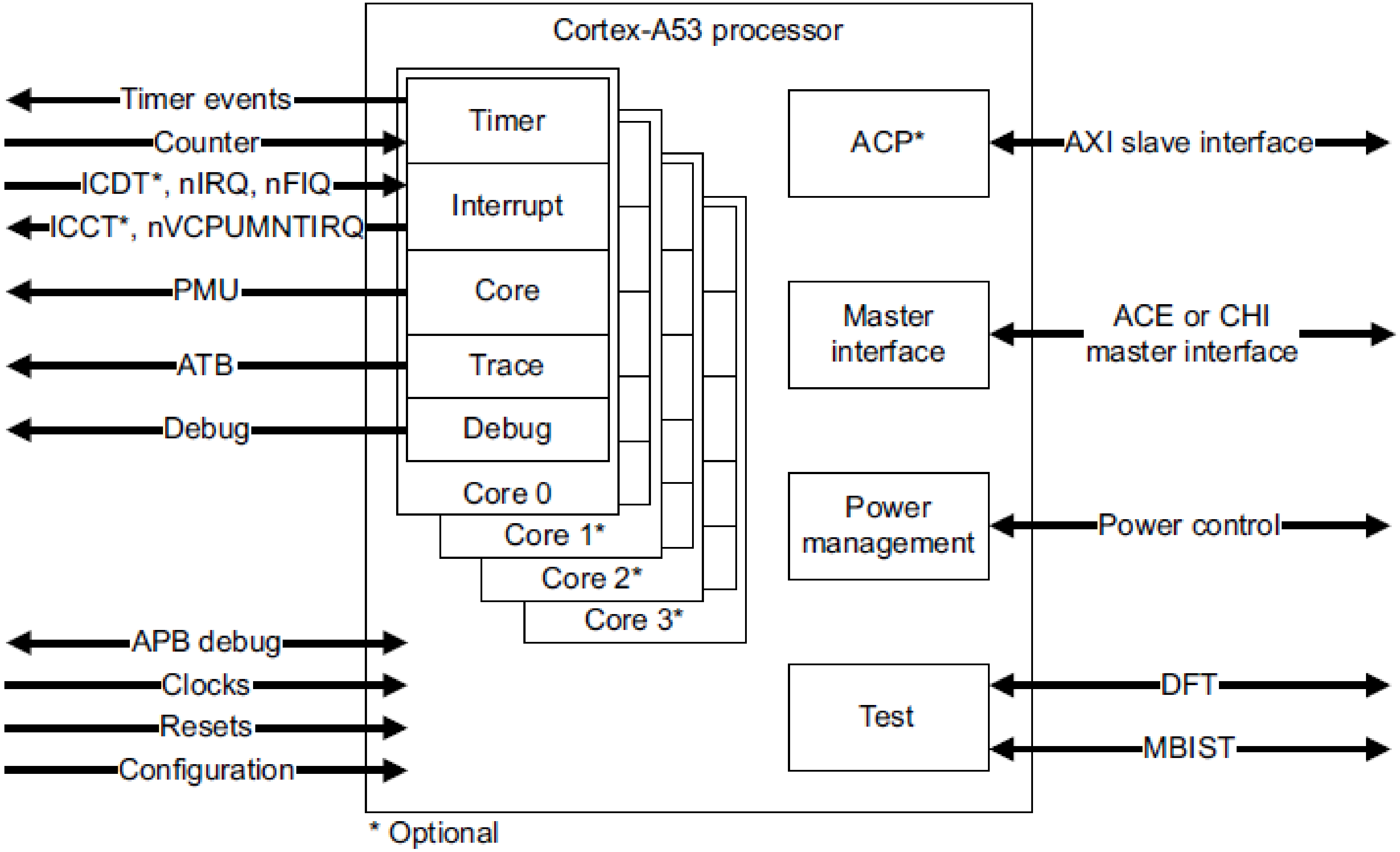
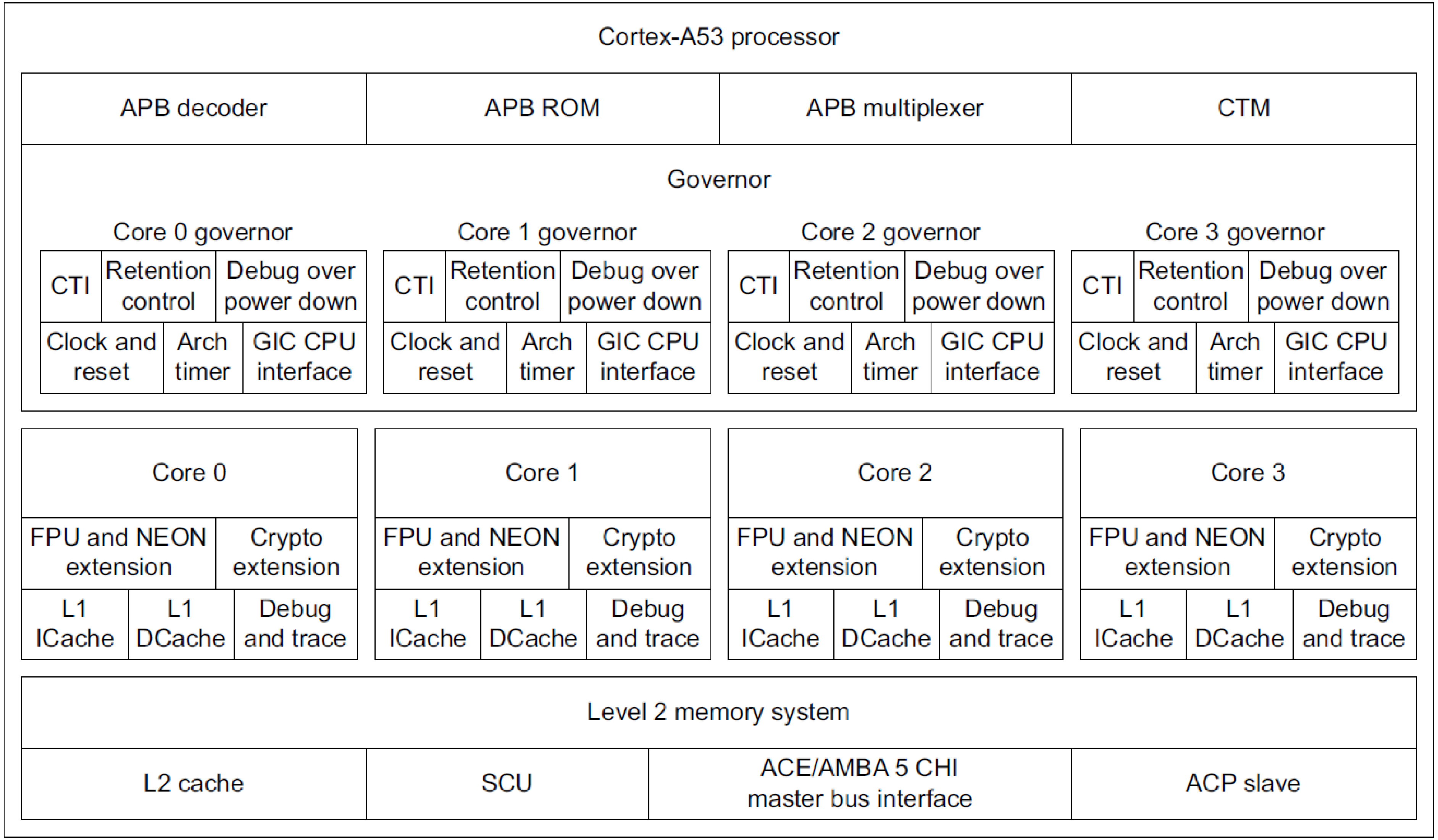


Figure 1-1 Example Cortex-A53 processor configuration

ARM Cortex–A53



Cortex-A53 processor

APB decoder | APB ROM | APB multiplexer | CTM

Governor

Core 0 governor: CTI, Retention control, Debug over power down, Clock and reset, Arch timer, GIC CPU interface

Core 1 governor: CTI, Retention control, Debug over power down, Clock and reset, Arch timer, GIC CPU interface

Core 2 governor: CTI, Retention control, Debug over power down, Clock and reset, Arch timer, GIC CPU interface

Core 3 governor: CTI, Retention control, Debug over power down, Clock and reset, Arch timer, GIC CPU interface

Core 0: FPU and NEON extension, Crypto extension, L1 ICache, L1 DCache, Debug and trace

Core 1: FPU and NEON extension, Crypto extension, L1 ICache, L1 DCache, Debug and trace

Core 2: FPU and NEON extension, Crypto extension, L1 ICache, L1 DCache, Debug and trace

Core 3: FPU and NEON extension, Crypto extension, L1 ICache, L1 DCache, Debug and trace

Level 2 memory system

L2 cache | SCU | ACE/AMBA 5 CHI master bus interface | ACP slave

Raspberry Pi 3B – BCM2837 (A53x4)

Raspberry Pi® is an **ARM** based credit card sized **SBC**(Single Board Raspberry Pi runs Debian based **GNU/Linux o**perating system Raspbian.

Quadcore A53 64-bit processor

VideoCore IV GPU

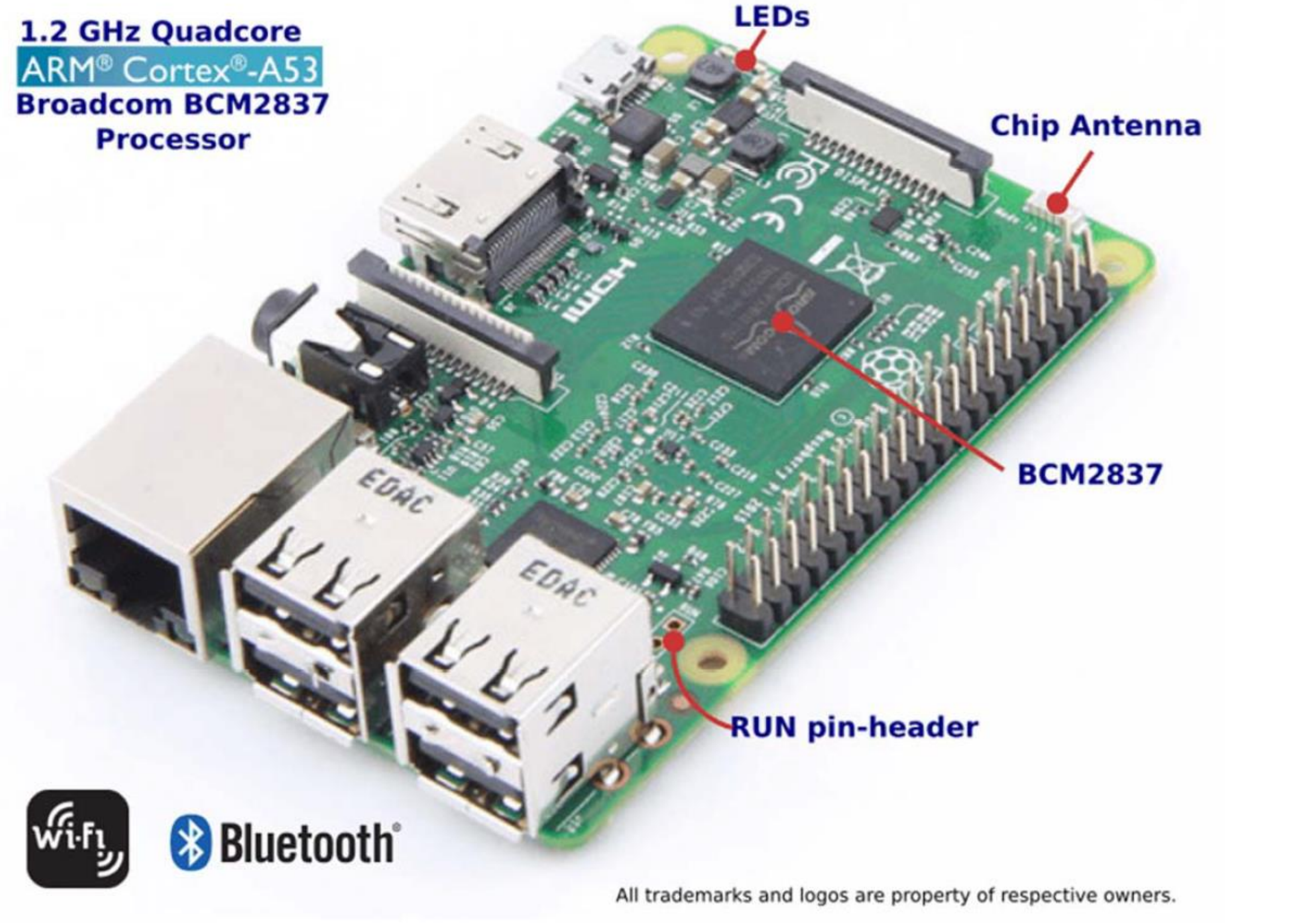
1 10/100 Ethernet

802.11n WiFi and Bluetooth 4.1

4 x USB 2.0

15 pin MIPI

HDMI out/Composite RCA



All trademarks and logos are property of respective owners.