



고려대학교
KOREA UNIVERSITY

COMP427 Embedded Systems

Lecture 3. Virtual Platform and ARM Intro.

Prof. Taeweon Suh

Computer Science & Engineering

Korea University

Virtual Platform (Virtual Prototype)

- **Virtual Platform (Virtual Prototype)** is a software model of a hardware system
 - Virtual Platform is very widely used for software development much before hardware is ready
 - Virtual Platform is used for the development of SoCs (System-on-Chips) and future PC systems



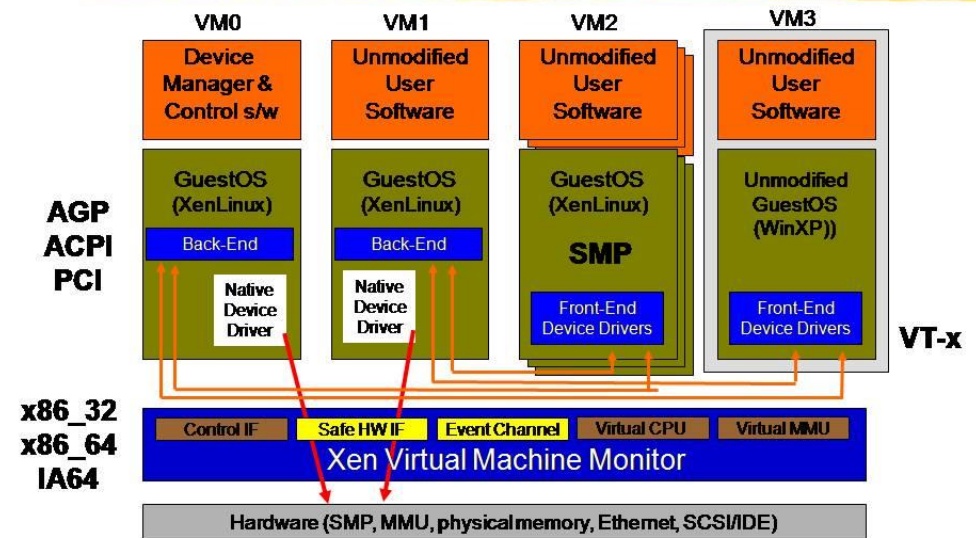
- Don't be confused with **Virtual Machine**!
 - VM allows the sharing of the underlying physical machine resources between different virtual machines, each running its own OS
 - The software layer providing the virtualization is called a **virtual machine monitor (VMM)** or **hypervisor**
 - x86 provides several instructions for virtualization

Virtual Machine Examples

KVM (Kernel-based Virtual Machine)



Xen 3.0 Architecture



Virtual Platform

Software models

SoC or AP model for the year 2016

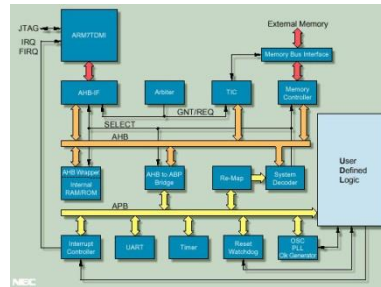
Software running on new products

- Firmware and RTOS porting to SoC
- Applications on SoC

PC system model for the year 2016

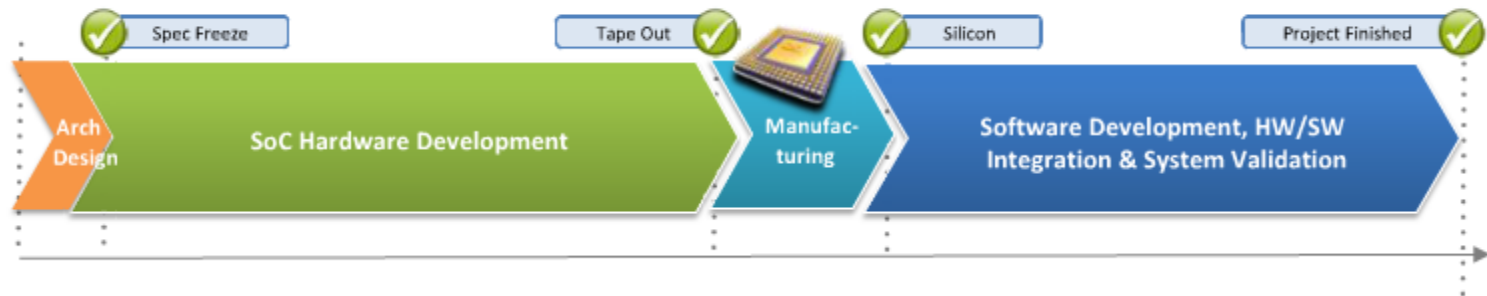
- BIOS, Firmware and OS development
- Validation software development

Your PC

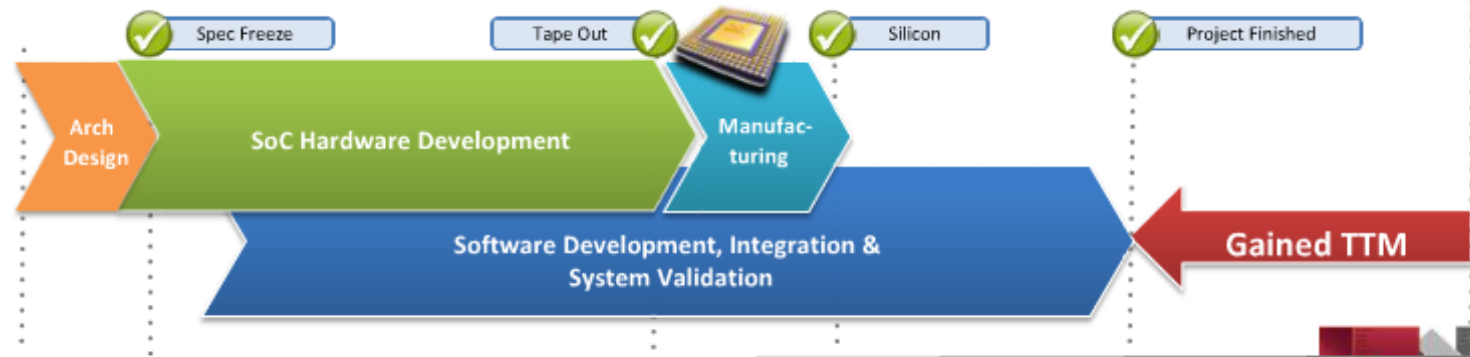


Time-to-Market Benefit

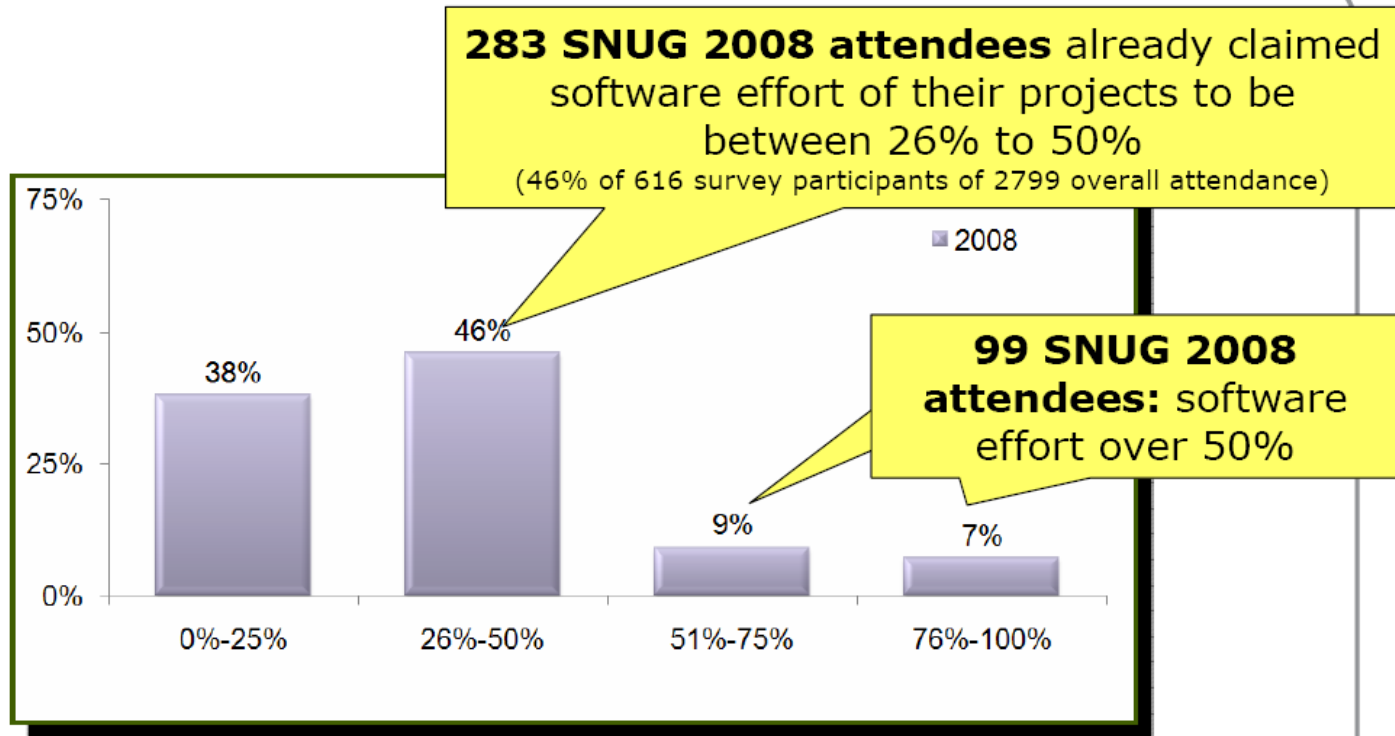
Standard project flow without prototyping



Project flow with prototyping



SoC Market Dynamics



What percentage of your total project effort is spent on software development (vs. hardware development) during design?

2008 N = 404; Margin of error = +/- 5%

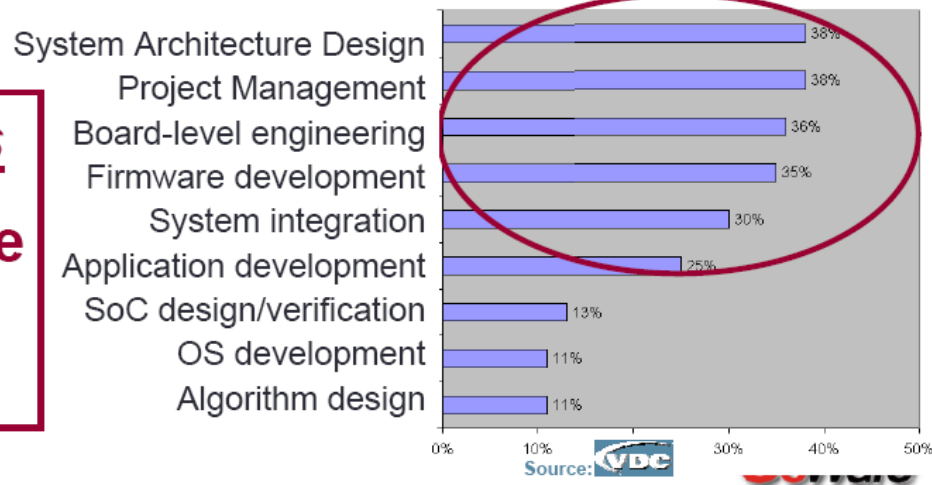
SNUG: Synopsys Users Group

SoC Design Challenges

Design Challenges

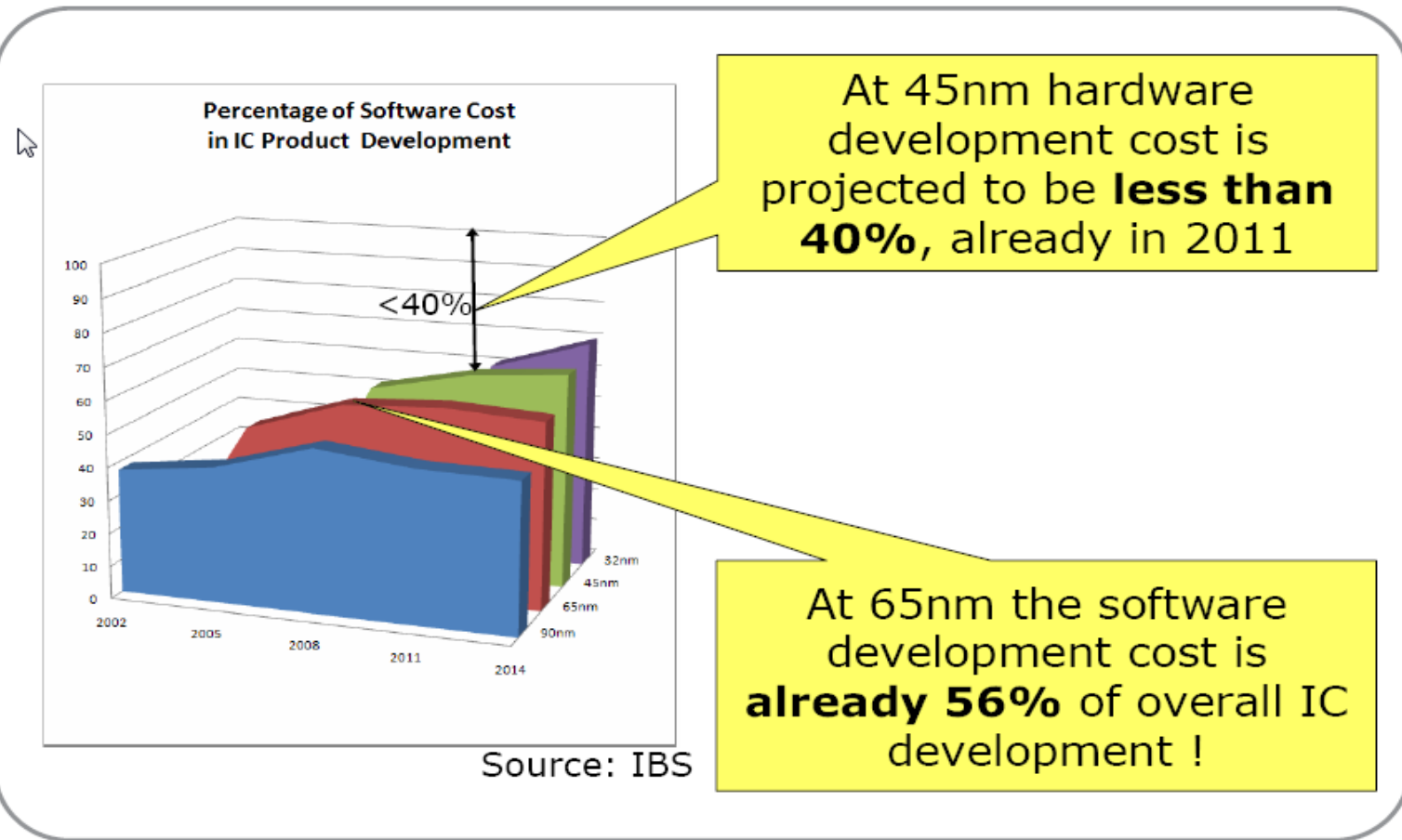
- 24% of projects canceled due to schedule slip
- 54% of SW designs completed **behind schedule**
- 33% of devices **miss functionality/performance**
- 80% of effort to correct **errors discovered late**

Engineering Tasks Believed To Be the Cause of
Schedule Delay on the Current Project



Top issues
Architecture
Software
Integration

Software Determines Project Schedules



Advantages of Virtual Platform



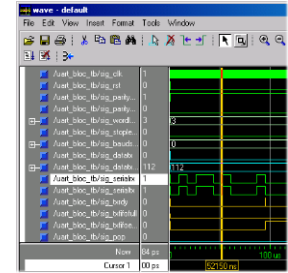
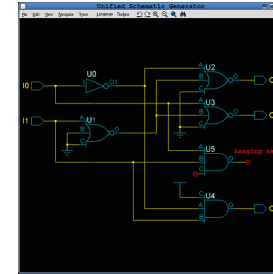
Early Availability	Available before chips come back from the fab and before boards have been built and debugged
Enhanced Debugging	Full visibility and control of multi-core platform with non intrusive access to all components
Easy to Deploy	No physical boards - minimal user ramp up time and logistical efforts to distribute and maintain

How is it different from simulators?

- In a broader sense, all the simulators may be viewed as virtual platform
 - Benchmarks and testvectors are running on virtual models (simulators)
- However, simulators tend to model only specific components rather than a whole system (platform)
 - For example, Simplescalar doesn't model peripheral devices. So, it is not feasible to run BIOS, DOS, OS (Windows)
 - <http://www.simplescalar.com/>

How fast VP should run?

- Performance comparisons of simulation, emulation, and virtual platform
 - **Hardware simulation**
 - Concurrent modeling
 - $\sim$ IPS (Instruction / second)
 - **Hardware emulation**
 - Porting RTLs into reconfigurable fabric - array of FPGAs (Field Programmable Gate Array)
 - KIPS $\sim$ MIPS depending on what you emulate
 - **Virtual platform**
 - $\sim$ MIPS
 - Able to run real-applications on top of OS in reasonable time



How to model VP?

- Depending on the level of accuracy you want to achieve, there are different levels of abstractions
- Level of abstractions
 - Cycle accurate model (CA)
 - Clock cycle-by-cycle accurate model
 - Programmer's view model (**PV, we use PV**)
 - Highly abstracted mode
 - Register accurate model
 - Functionally correct

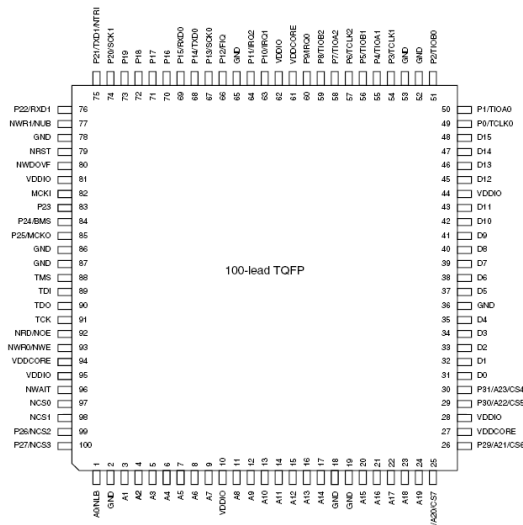
Which Language to Use for Modeling?

- Verilog-HDL and VHDL
 - Used to model cycle-accurate model
 - Too slow ($\sim$ IPS depending on complexity)
- **C, C++**
 - **Used to model PV in general**
 - Also can be used for cycle-accurate modeling

In this class...

- We are **not** going to use any hardware
- Instead, we are going to use a virtual platform (software model) of AT91
 - <http://www.atmel.com/>
- AT91 is an SoC (hardware chip) from Atmel
 - www.atmel.com
 - It includes ARM CPU and various peripherals such as timer and UART
- On top of the software model, we are going to run
 - Assembly programs
 - OS (Embedded Linux)
 - Applications written in C on top of the Embedded Linux

AT91x40

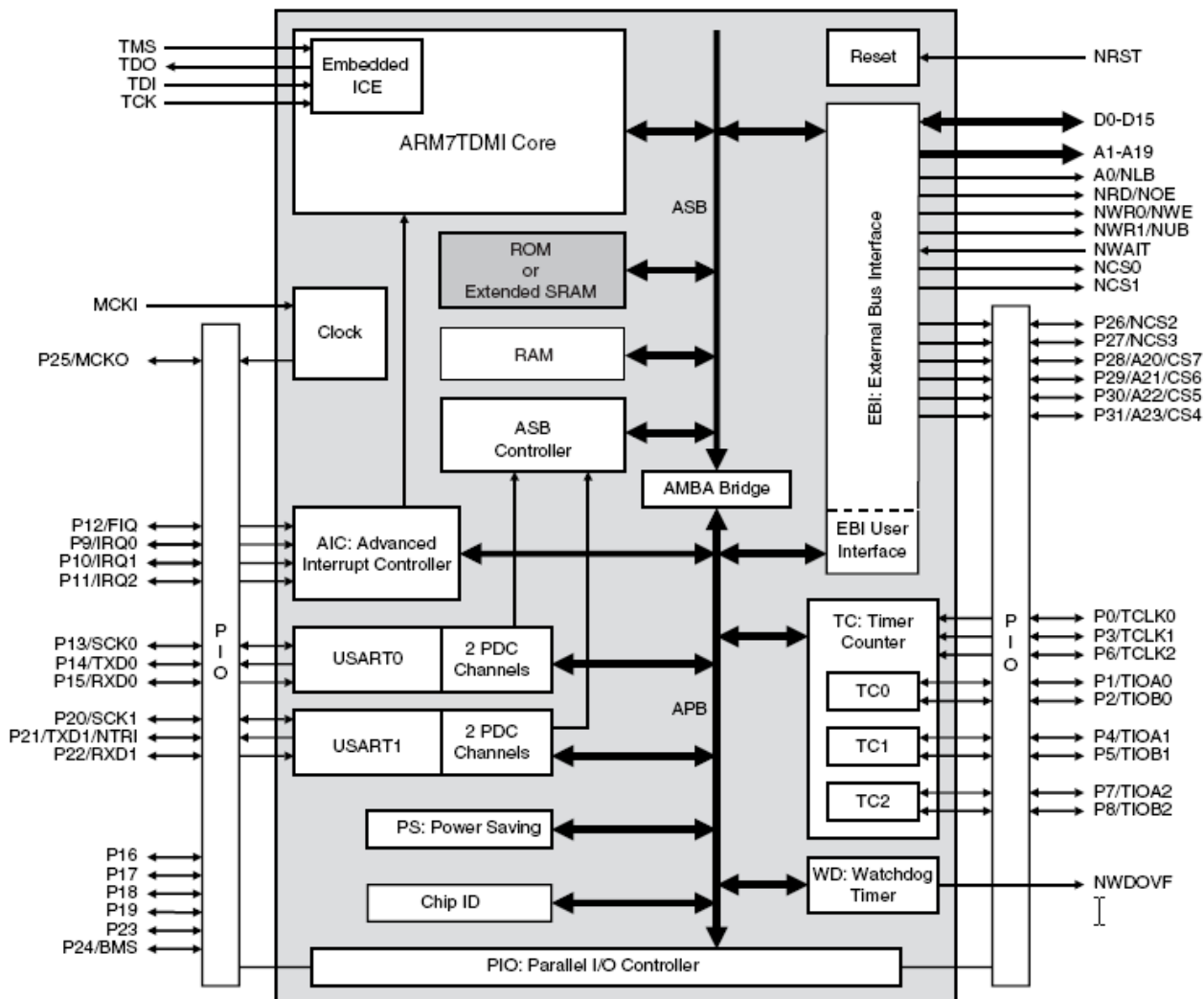


Description

The AT91X40 Series is a subset of the Atmel AT91 16/32-bit microcontroller family, which is based on the ARM7TDMI processor core. This processor has a high-performance 32-bit RISC architecture with a high-density 16-bit instruction set and very low power consumption. In addition, a large number of internally banked registers result in very fast exception handling, making the device ideal for real-time control applications. The AT91X40 Series features a direct connection to off-chip memory, including Flash, through the fully programmable External Bus Interface (EBI). An eight-level priority vectored interrupt controller, in conjunction with the Peripheral Data Controller significantly improve the real-time performance of the device.

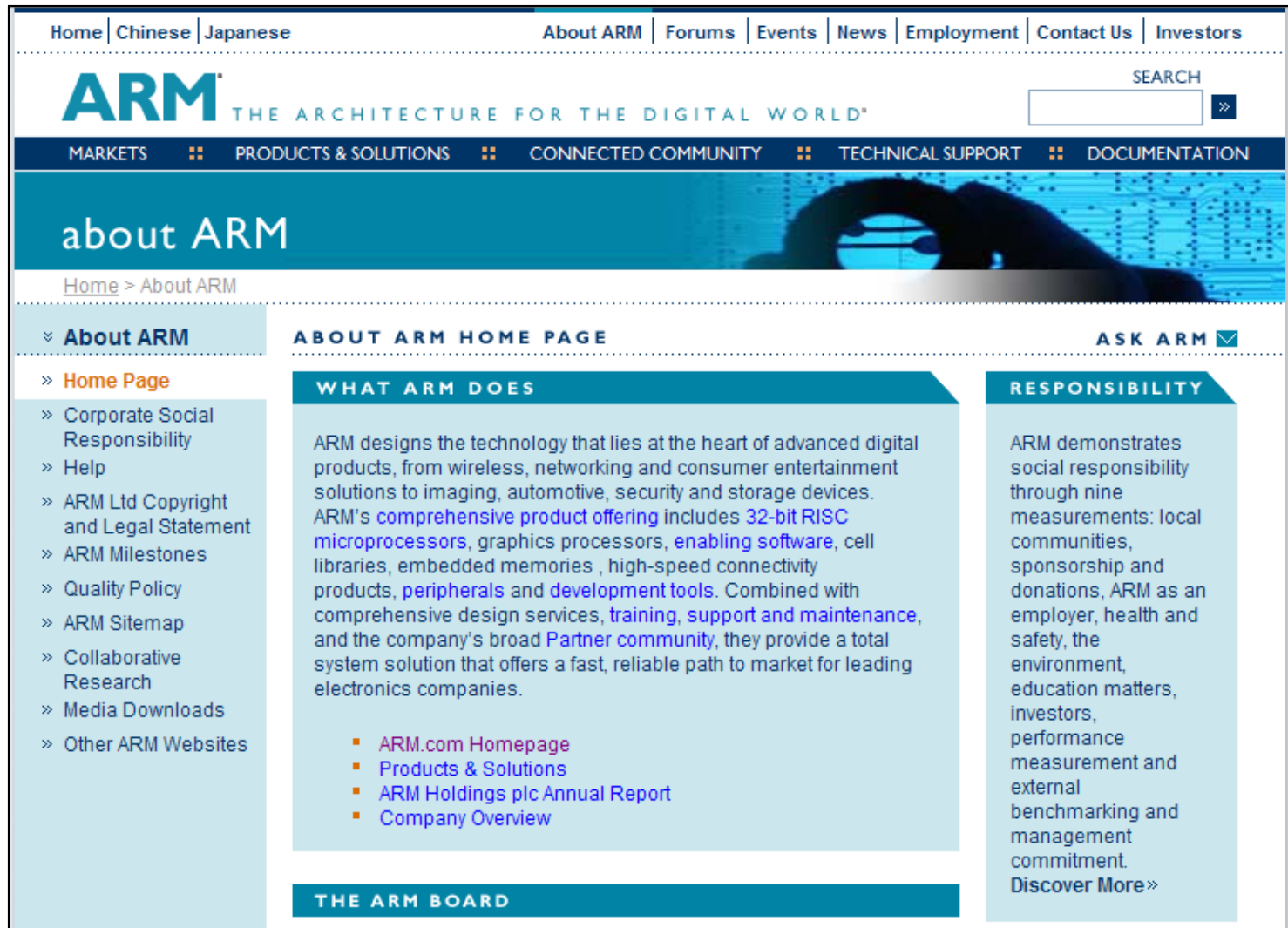
The devices are manufactured using Atmel's high-density CMOS technology. By combining the ARM7TDMI processor core with on-chip high-speed memory and a wide range of peripheral functions on a monolithic chip, the Atmel AT91X40 Series is a family of powerful microcontrollers that offer a flexible, cost-effective solution to many compute-intensive embedded control applications.

Block Diagram of AT91x40



-
- Let's focus on CPU (ARM7TDMI) first and come back later to the system block diagram

ARM (www.arm.com)



The screenshot shows the ARM website homepage. At the top, there is a navigation bar with links for Home, Chinese, Japanese, About ARM, Forums, Events, News, Employment, Contact Us, and Investors. Below this is the ARM logo and the tagline "THE ARCHITECTURE FOR THE DIGITAL WORLD". A search bar is located on the right. A secondary navigation bar contains links for Markets, Products & Solutions, Connected Community, Technical Support, and Documentation. The main content area has a large blue banner with the text "about ARM" and a background image of a hand holding a magnifying glass over a circuit board. Below the banner, there is a sidebar on the left with a list of links under the heading "About ARM". The main content area is divided into three columns. The first column is titled "WHAT ARM DOES" and contains a paragraph about ARM's technology and a list of links. The second column is titled "RESPONSIBILITY" and contains a paragraph about ARM's social responsibility and a link to "Discover More". The third column is titled "ASK ARM" and contains a link to "ASK ARM".

Home | Chinese | Japanese | About ARM | Forums | Events | News | Employment | Contact Us | Investors

ARM THE ARCHITECTURE FOR THE DIGITAL WORLD®

SEARCH >>

MARKETS || PRODUCTS & SOLUTIONS || CONNECTED COMMUNITY || TECHNICAL SUPPORT || DOCUMENTATION

about ARM

[Home](#) > [About ARM](#)

» **About ARM**

- » **Home Page**
- » Corporate Social Responsibility
- » Help
- » ARM Ltd Copyright and Legal Statement
- » ARM Milestones
- » Quality Policy
- » ARM Sitemap
- » Collaborative Research
- » Media Downloads
- » Other ARM Websites

ABOUT ARM HOME PAGE

WHAT ARM DOES

ARM designs the technology that lies at the heart of advanced digital products, from wireless, networking and consumer entertainment solutions to imaging, automotive, security and storage devices. ARM's [comprehensive product offering](#) includes [32-bit RISC microprocessors](#), graphics processors, [enabling software](#), cell libraries, embedded memories, high-speed connectivity products, [peripherals](#) and [development tools](#). Combined with comprehensive design services, [training, support and maintenance](#), and the company's broad [Partner community](#), they provide a total system solution that offers a fast, reliable path to market for leading electronics companies.

- [ARM.com Homepage](#)
- [Products & Solutions](#)
- [ARM Holdings plc Annual Report](#)
- [Company Overview](#)

THE ARM BOARD

ASK ARM

RESPONSIBILITY

ARM demonstrates social responsibility through nine measurements: local communities, sponsorship and donations, ARM as an employer, health and safety, the environment, education matters, investors, performance measurement and external benchmarking and management commitment. [Discover More](#)»

Our Vision

ARM designs technology that lies at the heart of advanced digital products



**EMBEDDED
SOLUTIONS**



**ENTERPRISE
SOLUTIONS**



**HOME
SOLUTIONS**



**MOBILE
SOLUTIONS**

EMERGING APPLICATIONS

ARM Partners



Over 1 Billion Processors in 3rd Quarter



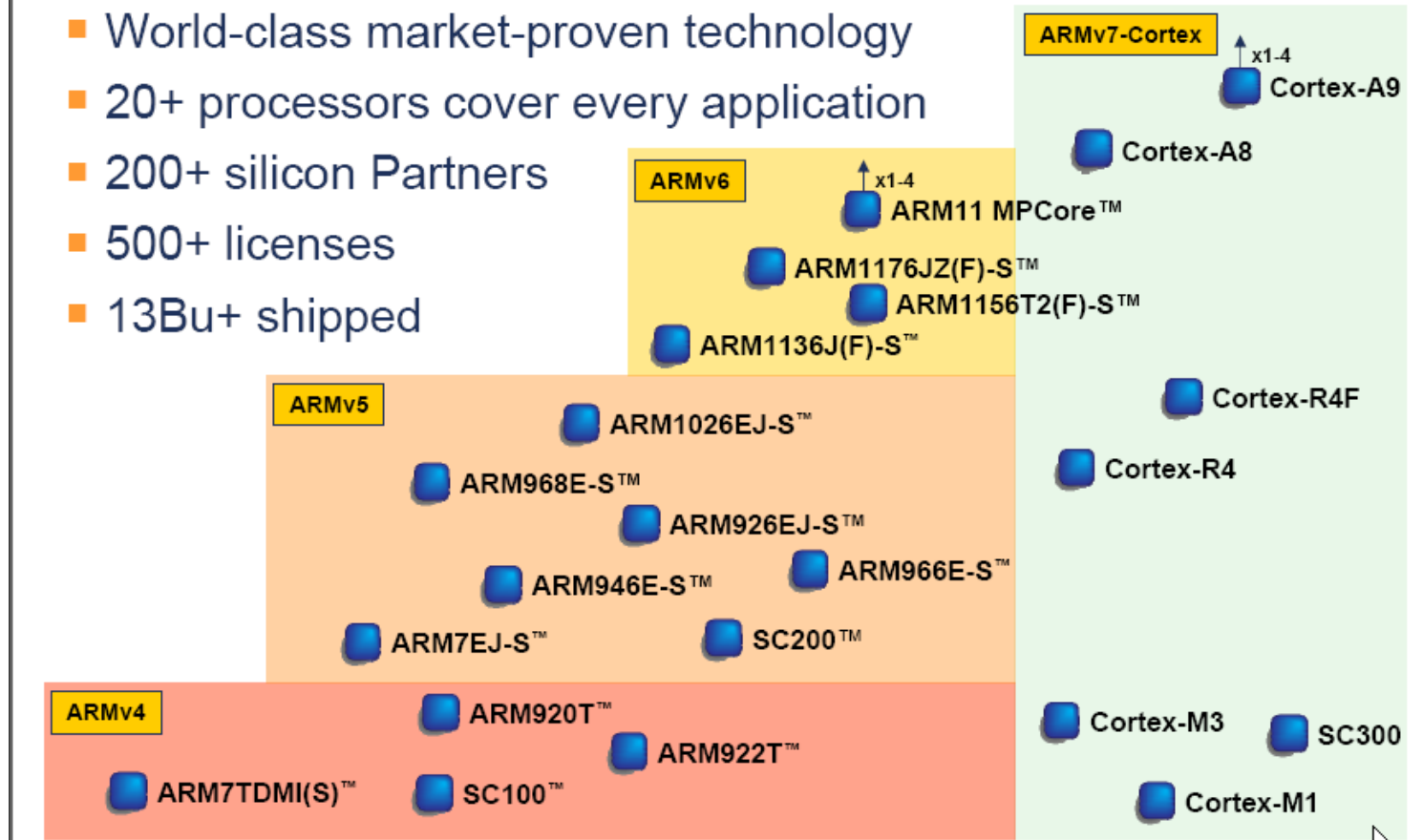
ARM Brief

- ARM architecture was first developed in the 1980s by Acorn
- Spin off from Acron in 1990
- Released ARM6 in early 1992
- ...
- As of 2013, ARM architecture is the most widely used 32-bit ISA in terms of quantity produced
- In 2010 alone, 6.1 billion ARM-based processors shipped, representing
 - 95% of smartphones
 - 35% of digital TV and set-top boxes
 - 10% of mobile computers

ARM Processor Portfolio

ARM Processor Portfolio

- World-class market-proven technology
- 20+ processors cover every application
- 200+ silicon Partners
- 500+ licenses
- 13Bu+ shipped

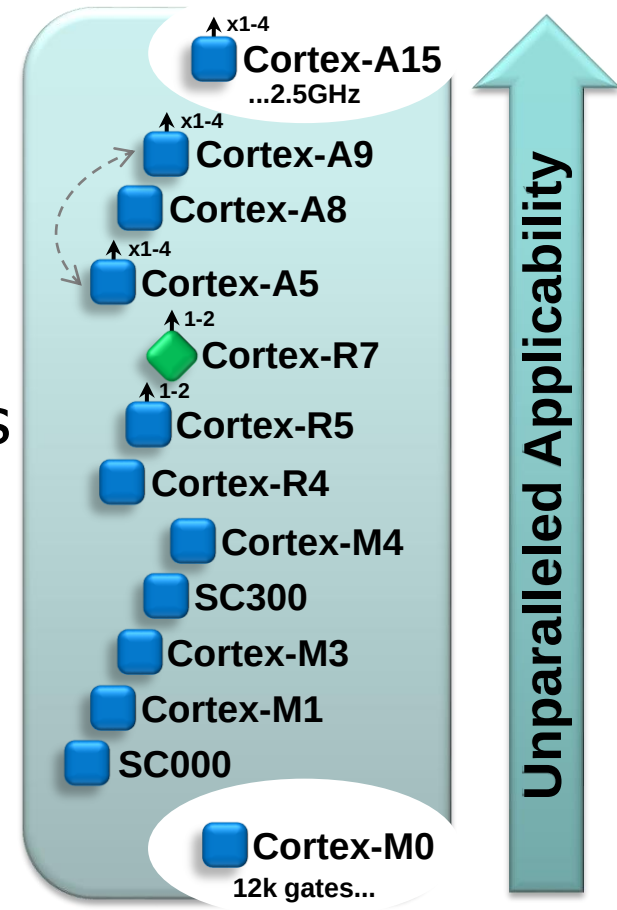


Product Code

- T: Thumb
- T2: Thumb-2 Enhancement
- D: Debug
- M: Multiplier
- I: Embedded ICE (In-Circuit Emulation)
- E: Enhanced DPS Extension
- J: Jazelle
 - Direct execution of 8-bit Java bytecode in hardware
- S: Synthesizable core
- Z: Should be TrustZone?

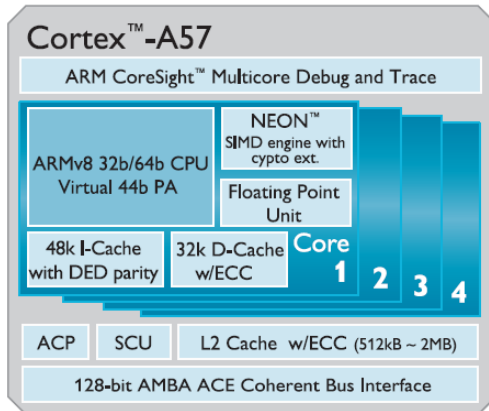
ARM Cortex Series

- ARM Cortex-**A** family:
 - Applications processors for feature-rich OS and 3rd party applications
- ARM Cortex-**R** family:
 - Embedded processors for real-time signal processing, control applications
- ARM Cortex-**M** family:
 - Microcontroller-oriented processors for MCU, ASSP, and SoC applications

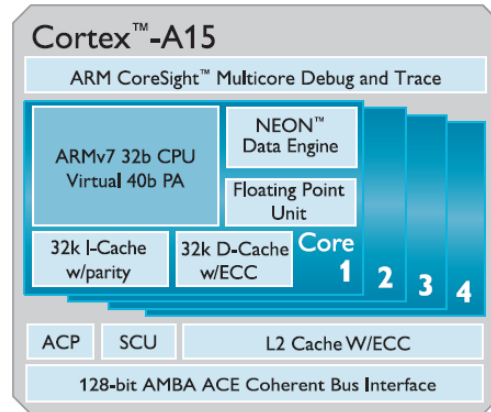


ARMv7-A

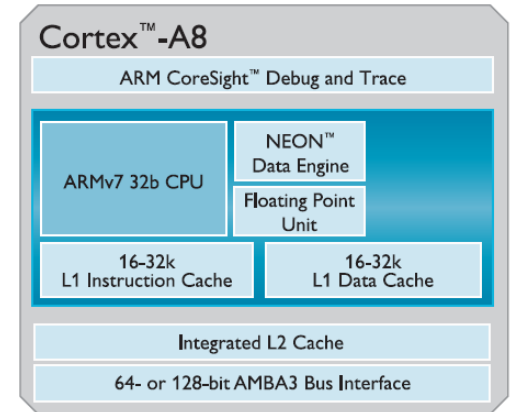
Cortex-A57 Processor



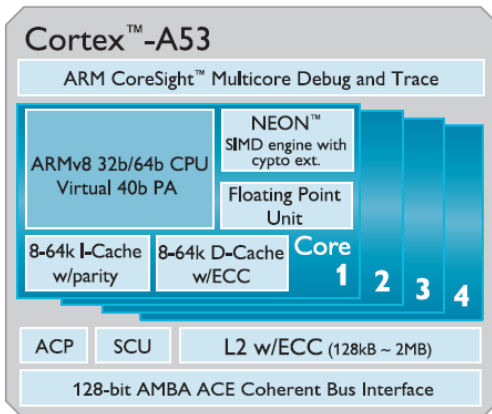
Cortex-A15 Processor



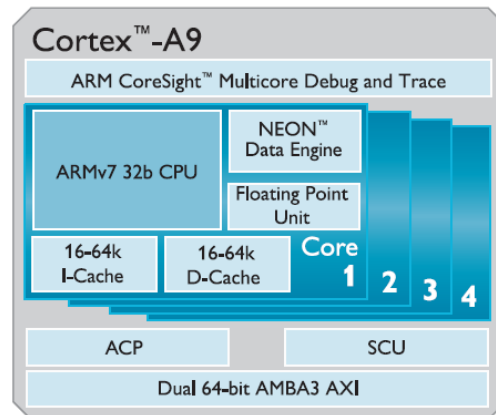
Cortex-A8 Processor



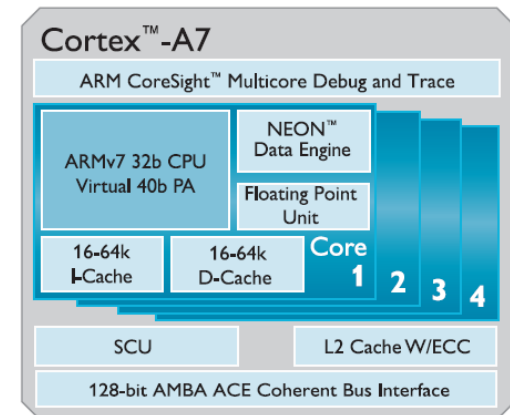
Cortex-A53 Processor



Cortex-A9 Processors



Cortex-A7 Processor



■ ACP: Accelerator Coherency Port

■ SCU: Snoop Control Unit

ARM Processor Brief

	#pipeline stages	Frequency	Architecture	Process
ARM6 (1992)	3	~33MHz	ARMv3	1.2μm
ARM7TDMI	3	~70MHz	ARMv4	0.13nm
ARM920T	5	~400MHz	ARMv4	90nm
ARM1136J	8	~1Ghz	ARMv6	65nm
Cortex-A9	8~11 (OoO)	~2GHz	ARMv7	32nm
Cortex-A15	15~24 (OoO)	~2.5GHz	ARMv7	22nm

■ OOO: Out Of Order

Abstraction

- Abstraction helps us deal with complexity
 - Hide lower-level detail
- Instruction set architecture (ISA)
 - An abstract interface between the hardware and the low-level software interface

Abstraction Analogies

Driver



**Abstraction
layer**



**Machine
Details**



**Combustion
Engine in a car**



**Break system
in a car**

Customer



**Abstraction
layer**



**Machine
Details**



**Hardware board in a
vending machine**

Abstraction in Computer

Users

Application programming using APIs

**Abstraction
layer**

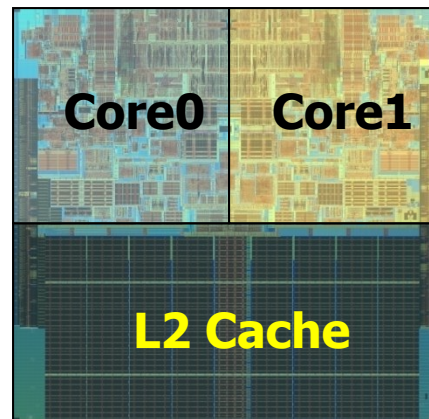
Operating Systems

**Abstraction
layer**

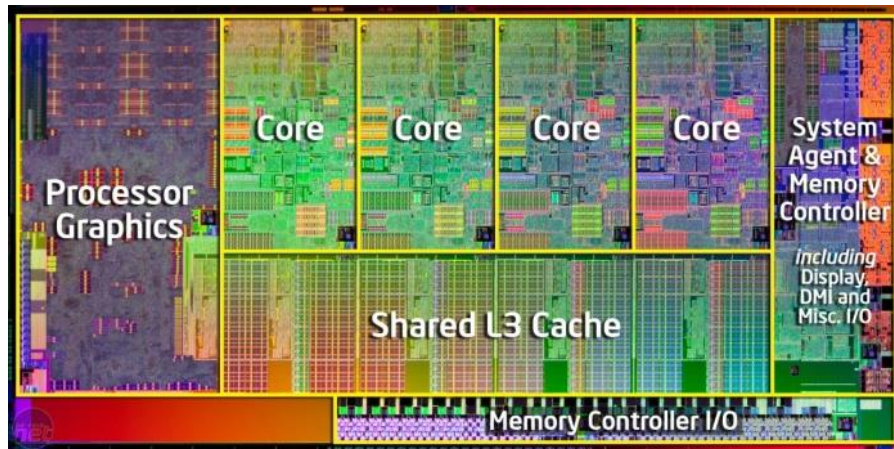
**Instruction Set Architecture
(ISA)**

**Machine language
Assembly language**

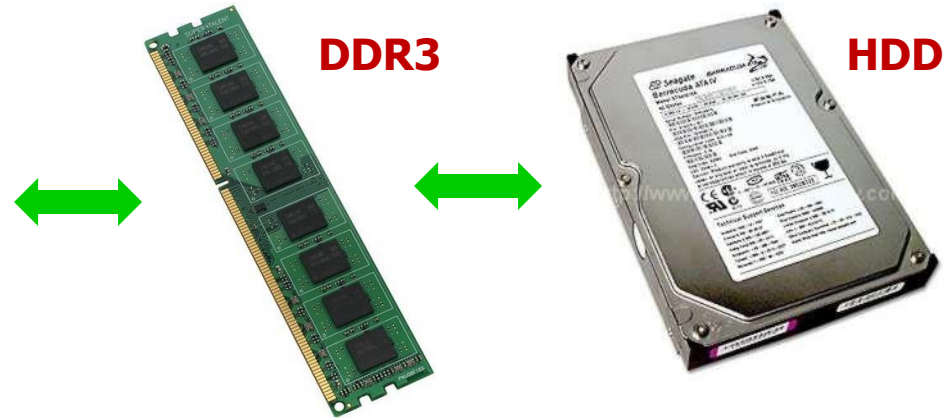
**Hardware
implementation**



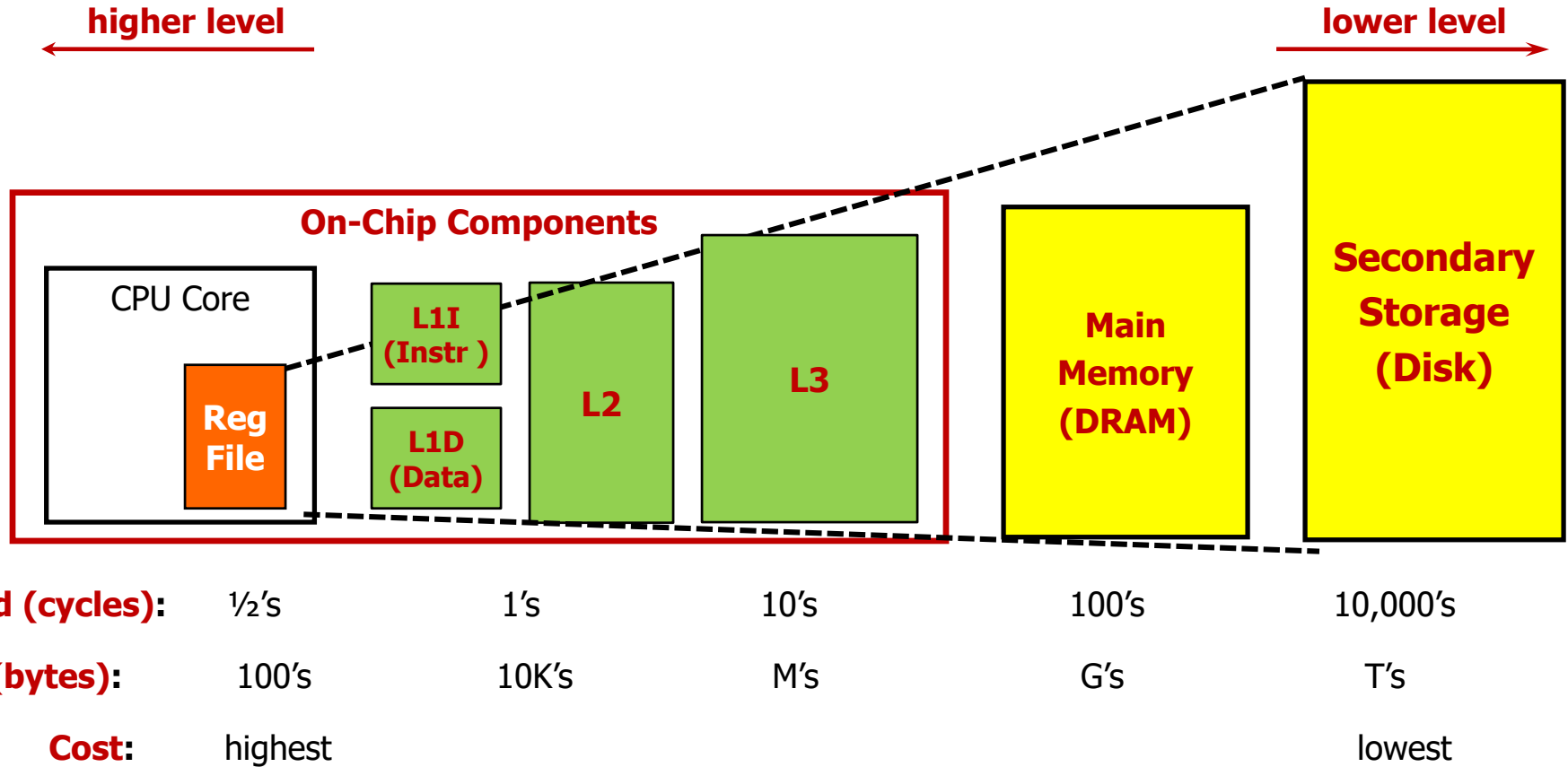
A Memory Hierarchy



**2nd Gen. Core i7
(2011)**



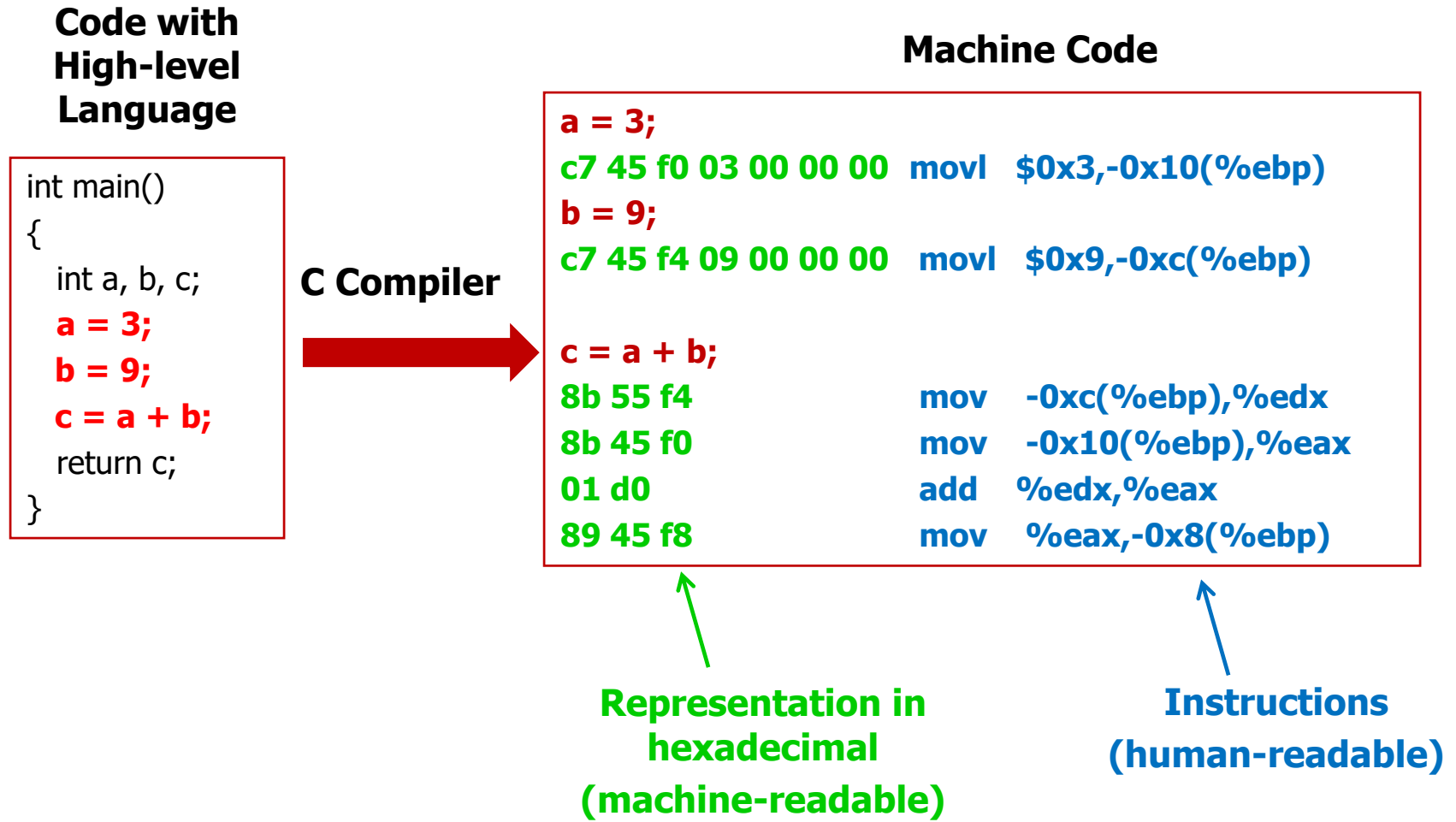
A Memory Hierarchy



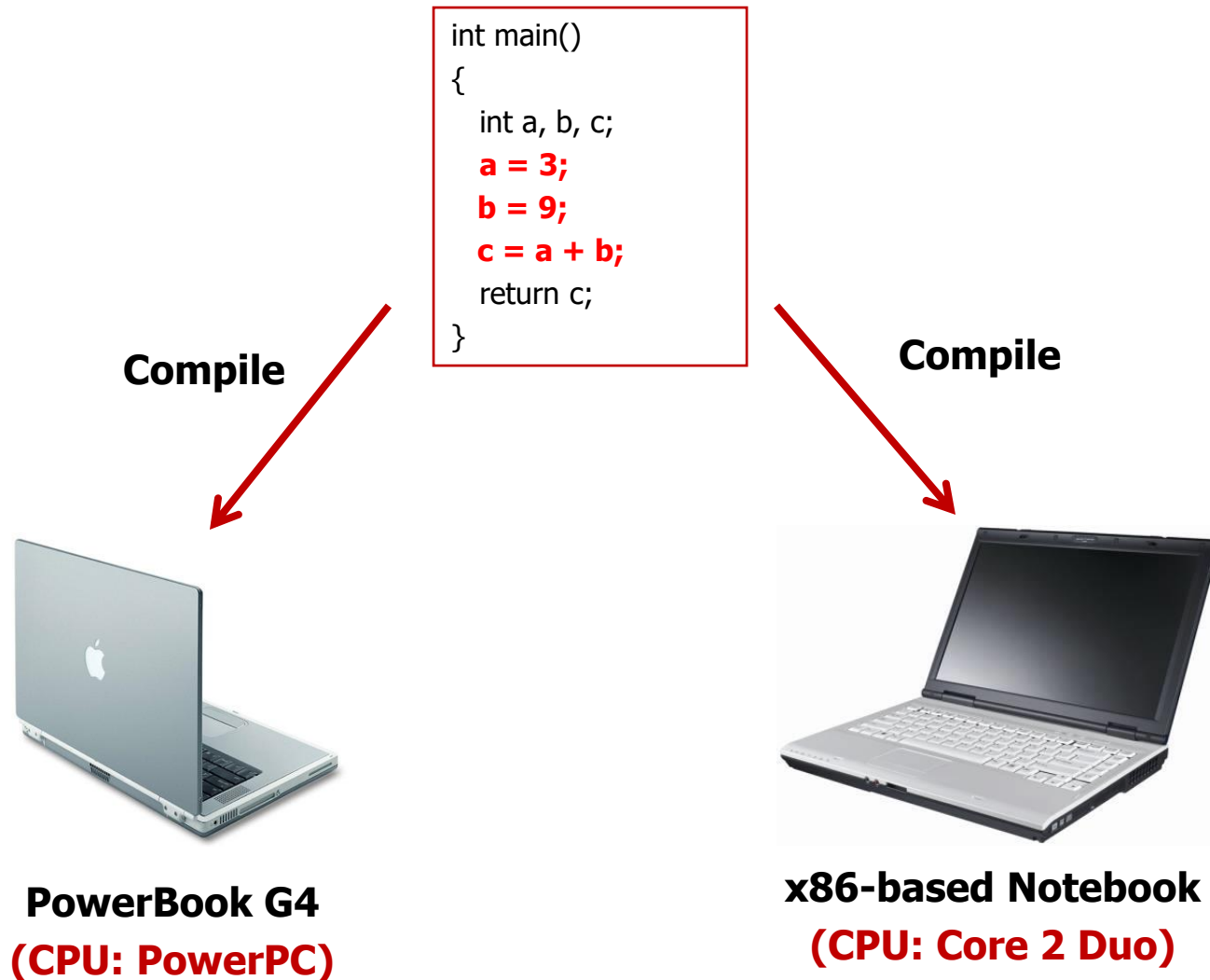
Typical and Essential Instructions

- CPU provides many instructions
 - It would be time-consuming to study all the instructions CPU provides
 - There are essential and common instructions
- Instruction categories
 - Data processing instructions
 - Arithmetic and Logical (Integer)
 - Memory access instructions
 - Load and Store
 - Branch instructions

Levels of Program Code (x86)



High-Level Code is Portable



Levels of Program Code (ARM)

- High-level language program (in C)

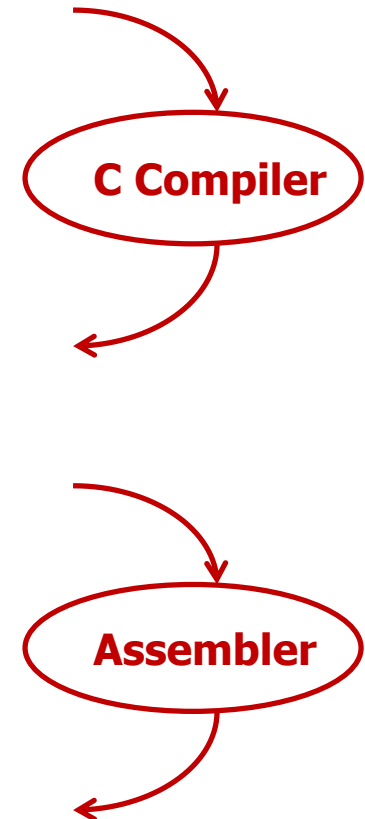
```
swap (int v[], int k)
{
    int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

- **Assembly language** program

```
swap:    sll    R2, R5, #2
         add    R2, R4, R2
         ldr    R12, 0(R2)
         ldr    R10, 4(R2)
         str    R10, 0(R2)
         str    R12, 4(R2)
         b      exit
```

- **Machine (object, binary)** code

```
000000 00000 00101 0001000010000000
000000 00100 00010 0001000000100000
. . .
```



CISC vs RISC

- CISC (Complex Instruction Set Computer)
 - One assembly instruction does **many (complex)** job
 - Example: `movs` in x86
 - Variable length instruction
 - Example: x86 (Intel, AMD), Motorola 68k



- RISC (Reduced Instruction Set Computer)
 - Each assembly instruction does a **small (unit)** job
 - Example: `lw`, `sw`, `add`, `slt` in MIPS
 - Fixed-length instruction
 - Load/Store Architecture
 - Example: MIPS, ARM

MIPS
TECHNOLOGIES

ARM[®]

ARM Architecture

- ARM is RISC (Reduced Instruction Set Computer)
 - x86 ISA is based on CISC (Complex Instruction Set Computer) even though x86 internally implements RISC-like microcode and pipelining
- Suitable for embedded systems
 - Very small die size (low price)
 - Low power consumption (longer battery life)

ARM Registers

- ARM has 31 general purpose registers and 6 status registers (32-bit each)

Modes						
Privileged modes						
Exception modes						
User	System	Supervisor	Abort	Undefined	Interrupt	Fast interrupt
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

 indicates that the normal register used by User or System mode has been replaced by an alternative register specific to the exception mode

ARM Registers

- Unbanked registers: R0 ~ R7
 - Each of them refers to the same 32-bit physical register in all processor modes.
 - They are completely general-purpose registers, with no special uses implied by the architecture
- Banked registers: R8 ~ R14
 - R8 ~ R12 have no dedicated special purposes
 - FIQ mode has dedicated registers for fast interrupt processing
 - R13 and R14 are dedicated for special purposes for each mode

Modes						
Privileged modes						
Exception modes						
User	System	Supervisor	Abort	Undefined	Interrupt	Fast interrupt
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
PC	PC	PC	PC	PC	PC	PC

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

 indicates that the normal register used by User or System mode has been replaced by an alternative register specific to the exception mode

R13, R14, and R15

- Some registers in ARM are used for special purposes
 - R15 == PC (Program Counter)
 - x86 uses a terminology called IP (Instruction Pointer)
 - R14 == LR (Link Register)
 - R13 == SP (Stack Pointer)

CPSR

- Current Program Status Register (CPSR) is accessible in all modes
- Contains all condition flags, interrupt disable bits, the current processor mode

Program status registers

The *Current Program Status Register* (CPSR) is accessible in all processor modes. It contains condition code flags, interrupt disable bits, the current processor mode, and other status and control information. Each exception mode also has a *Saved Program Status Register* (SPSR), that is used to preserve the value of the CPSR when the associated exception occurs.

Note

User mode and System mode do not have an SPSR, because they are not exception modes. All instructions that read or write the SPSR are UNPREDICTABLE when executed in User mode or System mode.

The format of the CPSR and the SPSRs is shown below.

31	30	29	28	27	26	25	24	23	20	19	16	15	10	9	8	7	6	5	4	0
N	Z	C	V	Q	Res	J	RESERVED		GE[3:0]		RESERVED			E	A	I	F	T	M[4:0]	

CPSR in ARM

N	Is set to bit 31 of the result of the instruction. If this result is regarded as a two's complement signed integer, then $N = 1$ if the result is negative and $N = 0$ if it is positive or zero.
Z	Is set to 1 if the result of the instruction is zero (this often indicates an <i>equal</i> result from a comparison), and to 0 otherwise.
C	Is set in one of four ways: • For an addition, including the comparison instruction CMN, C is set to 1 if the addition produced a carry (that is, an unsigned overflow), and to 0 otherwise. • For a subtraction, including the comparison instruction CMP, C is set to 0 if the subtraction produced a borrow (that is, an unsigned underflow), and to 1 otherwise. • For non-addition/subtractions that incorporate a shift operation, C is set to the last bit shifted out of the value by the shifter. • For other non-addition/subtractions, C is normally left unchanged (but see the individual instruction descriptions for any special cases).
V	Is set in one of two ways: • For an addition or subtraction, V is set to 1 if signed overflow occurred, regarding the operands and result as two's complement signed integers. • For non-addition/subtractions, V is normally left unchanged (but see the individual instruction descriptions for any special cases).

CPSR bits

A, I, and F are the interrupt disable bits:

- A bit** Disables imprecise data aborts when it is set. This is available only in ARMv6 and above. In earlier versions, bit[8] of CPSR and SPSRs must be treated as a reserved bit, as described in *Types of PSR bits* on page A2-11.
- I bit** Disables IRQ interrupts when it is set.
- F bit** Disables FIQ interrupts when it is set.

M[4:0]	Mode	Accessible registers
0b10000	User	PC, R14 to R0, CPSR
0b10001	FIQ	PC, R14_fiq to R8_fiq, R7 to R0, CPSR, SPSR_fiq
0b10010	IRQ	PC, R14_irq, R13_irq, R12 to R0, CPSR, SPSR_irq
0b10011	Supervisor	PC, R14_svc, R13_svc, R12 to R0, CPSR, SPSR_svc
0b10111	Abort	PC, R14_abt, R13_abt, R12 to R0, CPSR, SPSR_abt
0b11011	Undefined	PC, R14_und, R13_und, R12 to R0, CPSR, SPSR_und
0b11111	System	PC, R14 to R0, CPSR (ARMv4 and above)

CPSR bits

J	T	Instruction set
0	0	ARM
0	1	Thumb
1	0	Jazelle
1	1	RESERVED

- ARM: 32-bit mode
- Thumb: 16-bit mode
- Jazelle: Special mode for JAVA acceleration

Interrupt

- **Interrupt** is an **asynchronous signal** from hardware indicating the need for attention or a **synchronous event in software** indicating the need for a change in execution.
 - **Hardware interrupt** causes the processor (CPU) to save its state of execution via a context switch, and begin execution of an interrupt handler.
 - **Software interrupt** is usually implemented as an instruction in the instruction set, which cause a context switch to an interrupt handler similar to a hardware interrupt.
- Interrupt is a commonly used technique in computer system for communication between CPU and peripheral devices
- Operating systems also extensively use interrupt (timer interrupt) for task (process, thread) scheduling

Hardware Interrupt in ARM

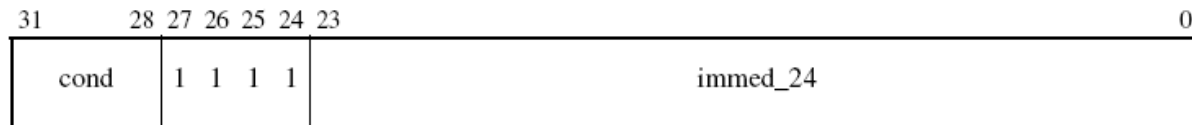
- IRQ (Normal interrupt request)
 - Informed to CPU by asserting IRQ pin
 - Program jumps to 0x0000_0018
- FIQ (Fast interrupt request)
 - Informed to CPU by asserting FIQ pin
 - Has a higher priority than IRQ
 - Program jumps to 0x0000_001C



Software Interrupt in ARM

- There is an software interrupt instruction in ARM
 - SWI instruction
- Software interrupt is commonly used by OS for system calls
 - Example: `open()`, `close()` .. etc

A4.1.107 SWI



SWI (Software Interrupt) causes a SWI exception (see *Exceptions* on page A2-16).

Syntax

SWI{<cond>} <immed_24>

where:

<cond> Is the condition under which the instruction is executed. The conditions are defined in *The condition field* on page A3-3. If <cond> is omitted, the AL (always) condition is used.

<immed_24> Is a 24-bit immediate value that is put into bits[23:0] of the instruction. This value is ignored by the ARM processor, but can be used by an operating system SWI exception handler to determine what operating system service is being requested (see *Usage* on page A4-211 below for more details).

Exception Vectors in ARM

Table A2-4 Exception processing modes

Exception type	Mode	VE ^a	Normal address	High vector address
Reset	Supervisor		0x00000000	0xFFFF0000
Undefined instructions	Undefined		0x00000004	0xFFFF0004
Software interrupt (SWI)	Supervisor		0x00000008	0xFFFF0008
Prefetch Abort (instruction fetch memory abort)	Abort		0x0000000C	0xFFFF000C
Data Abort (data access memory abort)	Abort		0x00000010	0xFFFF0010
IRQ (interrupt)	IRQ	0	0x00000018	0xFFFF0018
		1	IMPLEMENTATION DEFINED	
FIQ (fast interrupt)	FIQ	0	0x0000001C	0xFFFF001C
		1	IMPLEMENTATION DEFINED	

a. VE = vectored interrupt enable (CP15 control); RAZ when not implemented.

N

Exception Priority in ARM

Table A2-1 Exception priorities

Priority		Exception
Highest	1	Reset
	2	Data Abort (including data TLB miss)
	3	FIQ
	4	IRQ
	5	Imprecise Abort (external abort) - ARMv6
	6	Prefetch Abort (including prefetch TLB miss)
Lowest	7	Undefined instruction SWI