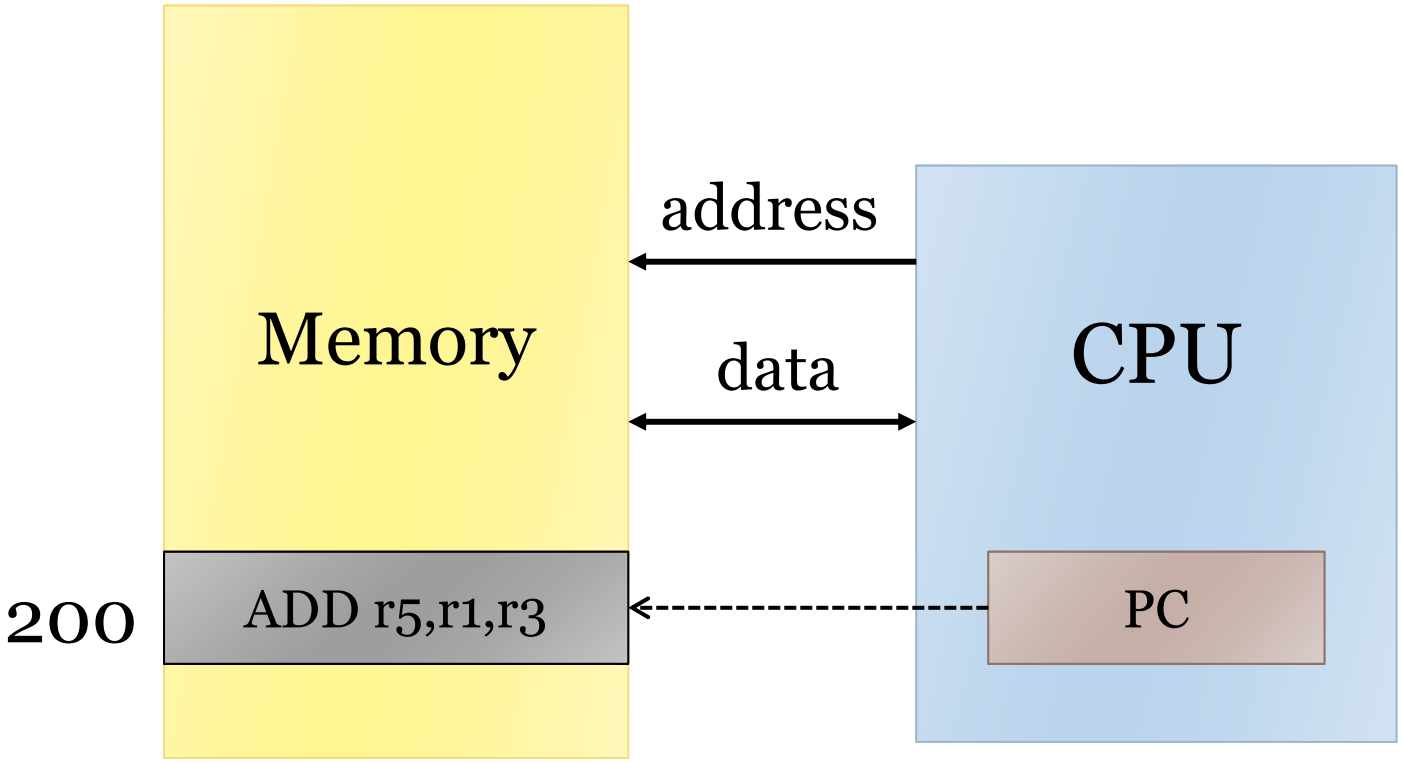
ARM Processor

Jin-Soo Kim (*jinsookim@skku.edu*)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

CPU Architecture

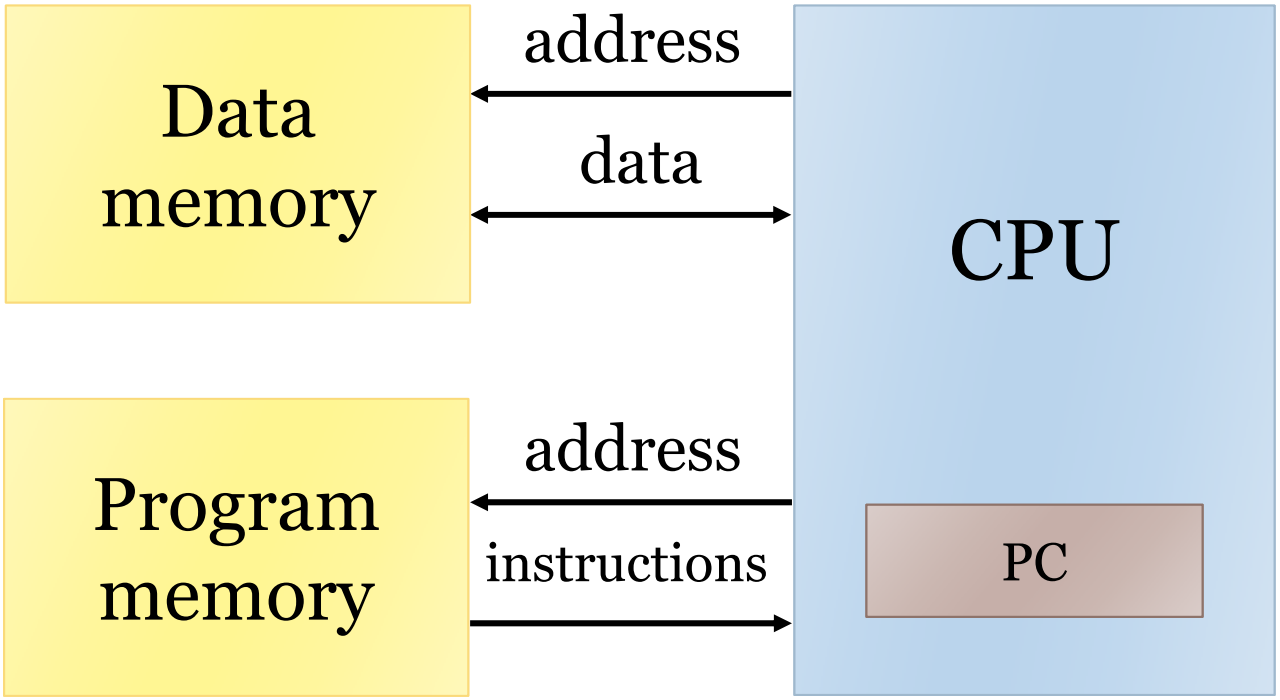
CPU & Memory



von Neumann Architecture

- Separate CPU and memory
  - Memory holds data and instructions.
  - CPU fetches instructions from memory.
  - CPU manipulates data by performing arithmetic and logical operations
- CPU registers help out
  - Program Counter (PC)
  - Instruction Register (IR)
  - General-Purpose Registers (GPRs), etc.

Harvard Architecture



von Neumann vs. Harvard

- Harvard can't use self-modifying code.
- Harvard allows two simultaneous memory fetches.
- Most DSPs use Harvard architecture for streaming data:
  - Greater memory bandwidth
  - More predictable bandwidth

RISC vs. CISC

- Complex Instruction Set Computer (CISC)
  - Many addressing modes
  - Many complex operations
  - Different instruction formats of varying lengths
- Reduced Instruction Set Computer (RISC)
  - Load/store
  - Pipelinable instructions

Instruction Set Architecture

- Supported operations
- Number of operands
- Types of operands
- Addressing modes
- Fixed vs. variable length
- Registers visible to the programmer
- …

Multiple Implementations

- Successful architectures have several implementations:
  - Varying clock speeds
  - Different bus widths
  - Different cache sizes
  - …

Assembly Language

- Textual description of instructions
- One instruction per line

```
        LDR   r0, [r8]
label1  ADD   r4, r0, r1
        CMP   r4, r5        ; Compare r4 < r5
        BGE   label1
```

Labels provide names for addresses

Comment

- Pseudo-ops
  - Define current address, reserve storage, constants, …

Introduction to the ARM Architecture

Overview (1)

- Advanced RISC Machine
- The most widely used 32-bit ISA
  - 98% of the more than 1 billion mobile phones sold used at least one ARM processor (2005)
  - 4 billion shipped in 2008
  - 90% of all embedded 32-bit RISC processors (2009)
- ARM architecture has been extended over several versions.

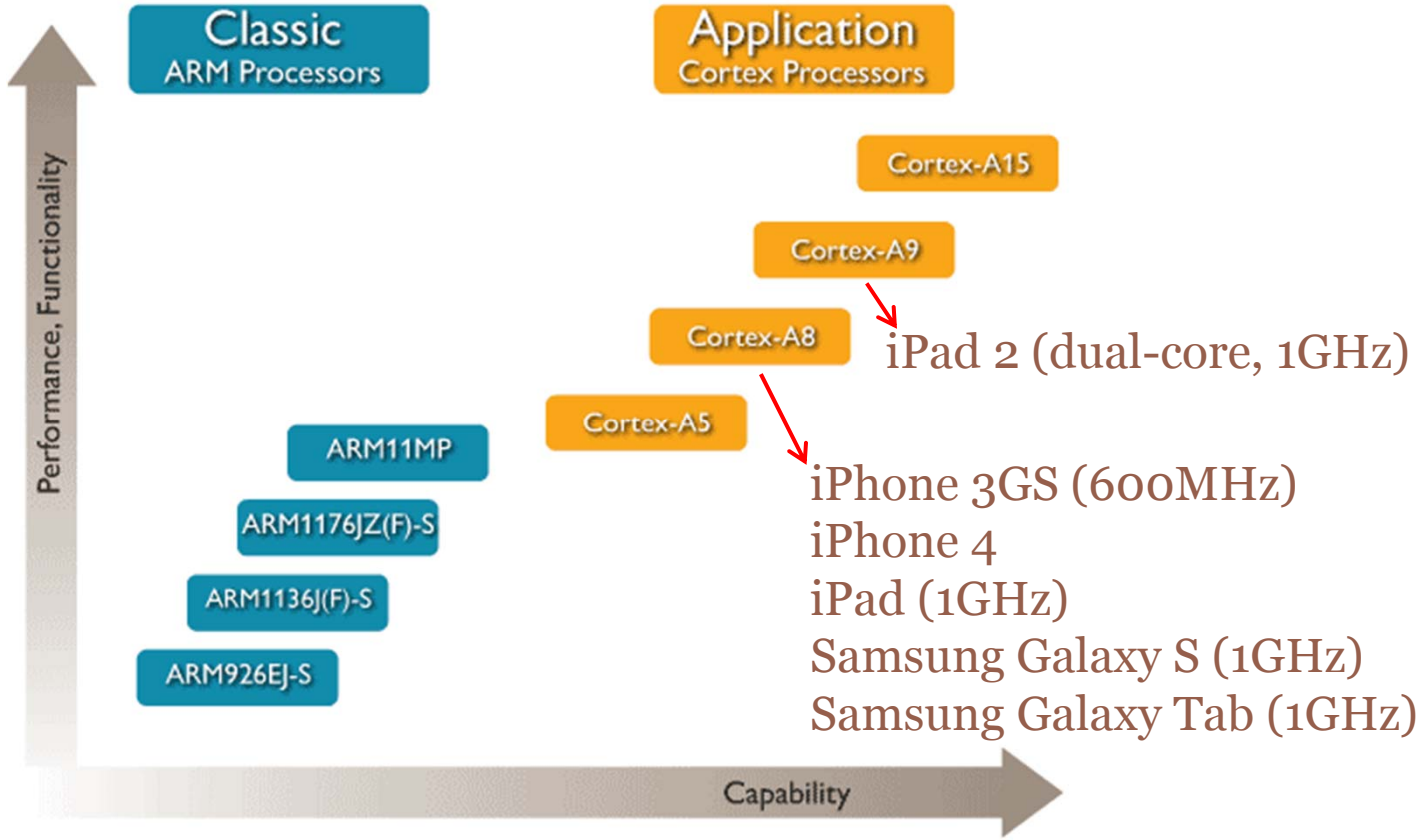
Overview (2)

- ARM processors developed by licensees
  - DEC StrongARM
  - Marvell (formerly Intel) Xscale
  - Nintendo
  - Nvidia Tegra
  - Qualcomm Snapdragon
  - TI OMAP
  - Samsung Hummingbird
  - Apple A4/A5

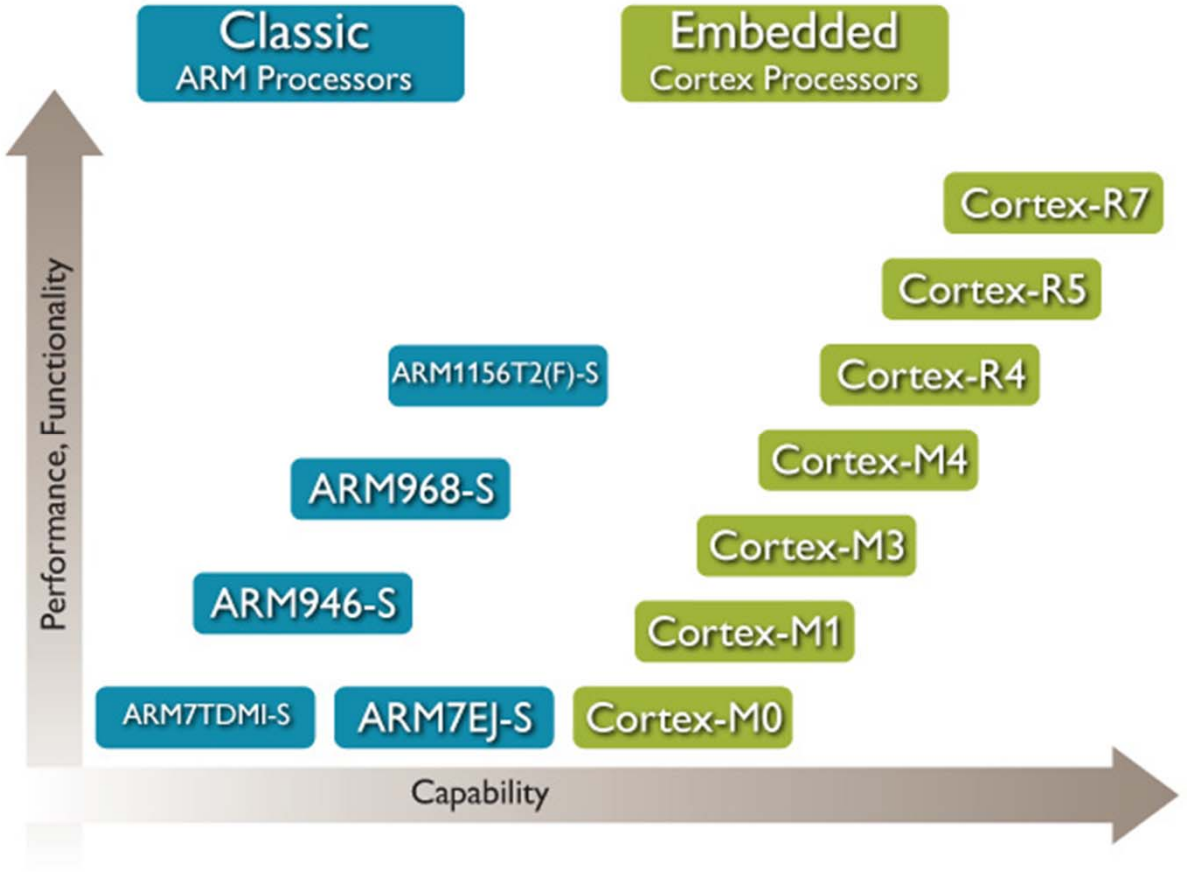
ARM Design Goals

- High performance
- Small code size
- Low power consumption
- Small silicon area

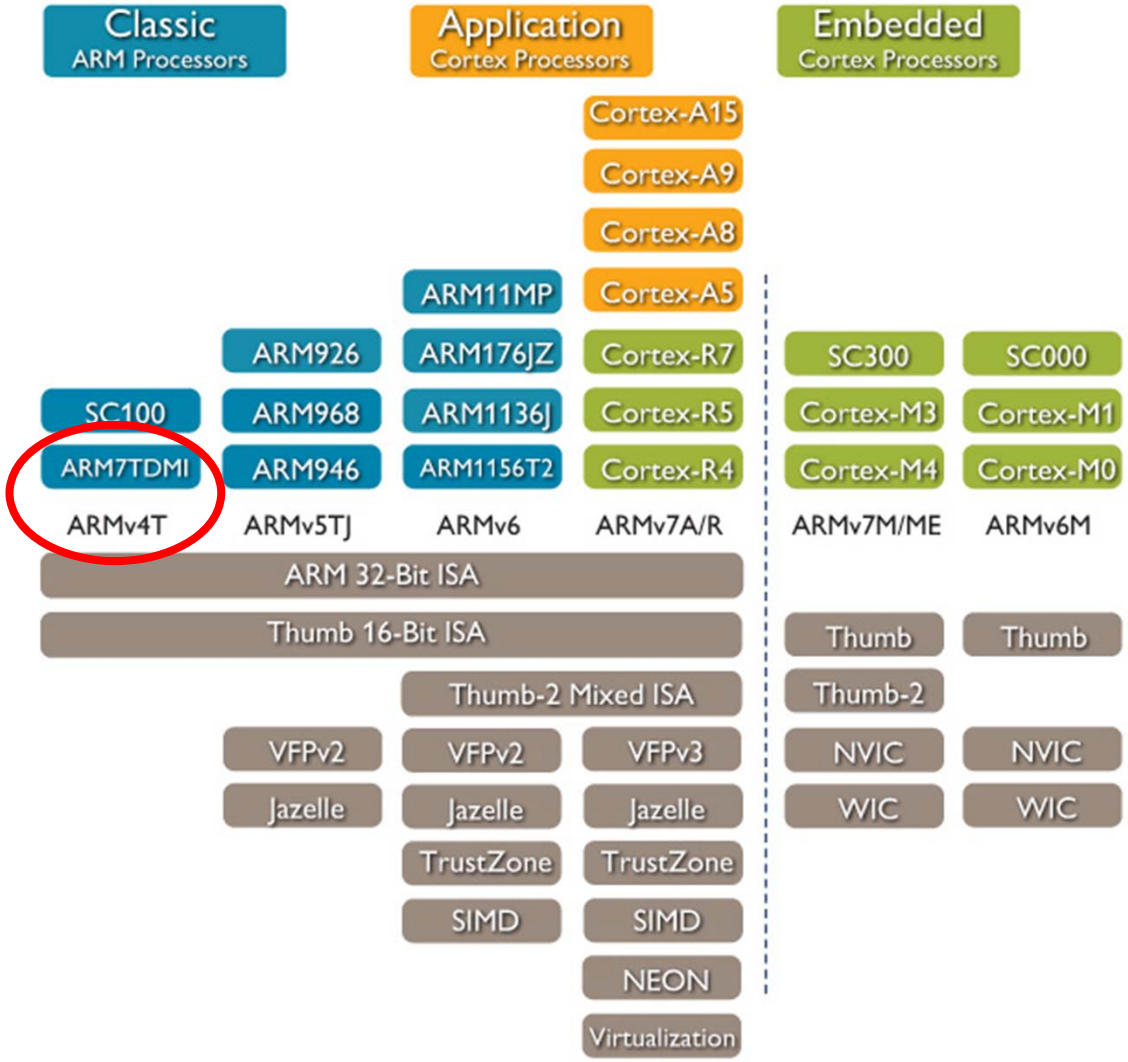
Application Processors



Embedded Processors



Processors Lineup



- Implements the ARMv4T architecture
- Applications
  - iPod from Apple
  - Nintendo DS & Game Boy Advance
  - Most of Nokia's mobile phones
  - Lego Mindstorms NXT
  - iriver portable digital audio players
  - Roomba 500 series from iRobot

ARM Architecture (1)

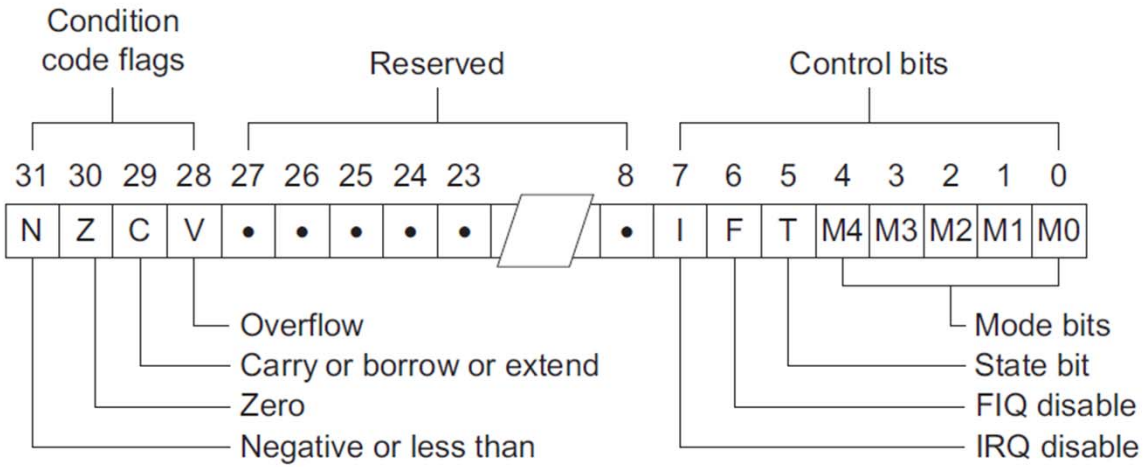
- RISC architecture
  - A large uniform register file
  - A load/store architecture
  - Simple addressing modes
  - Uniform and fixed-length instruction fields

ARM Architecture (2)

- Other features
  - Arithmetic/logical operations combined with a shift
  - Conditional execution of almost all instructions
  - Auto-increment (-decrement) addressing modes
  - Load and Store Multiple instructions

ARM Registers

- 16 user-visible 32-bit registers: r0-r15
  - r13: Stack Pointer (SP)
  - r14: Link Register (LR)
  - r15: Program Counter (PC)
- Current Program Status Register (CPSR)



ARM Status Bits

- Every arithmetic, logical, or shifting operation sets CPSR bits:
  - N (Negative), Z (Zero), C (Carry), V (Overflow)
  - Condition codes updated when S bit = 1
- Example:
  - 0xffffffff + 0x1 = 0x0 ; NZCV = 0110
  - 0x7fffffff + 0x1 = 0x80000000 ; NZCV = 1001
  - 0x0 – 0x1 = 0xffffffff ; NZCV = 1000

Memory Model

- A linear collection of (up to 2^{32}) bytes
- Supports both big-endian & little-endian
  - Controlled by the CFGBIGEND signal (ARM7)
  - CFGBIGEND == 0: Little-endian (default)
  - CFGBIGEND == 1: Big-endian
- Data types
  - word (32-bit): aligned to 4-byte boundary
  - halfword (16-bit): aligned to 2-byte boundary
  - byte (8-bit)

ARM Instructions

- Basic format
  - Two sources, one destination
  - All arithmetic operations have this form

```
ADD  r0, r1, r2          ; r0 = r1 + r2
```

- Operand types
  - Register (r0~r15): `ADD  r0, r1, r2`
  - Immediate: `ADD  r0, r1, #4`
  - Memory: `LDR  r5, [r3, #32]`

Thumb Instruction Sets (1)

- 16-bit instruction set
- A subset of the most commonly used 32-bit ARM instructions
- Significant improved code density at a cost of some reduction in performance
  - Typically 65% of the ARM code size
  - 160% performance of the ARM code when running with a 16-bit memory system

Thumb Instruction Set (2)

- Transparently decompressed to full 32-bit ARM instructions in real time, without performance loss
- A processor executing Thumb instructions can change to ARM instructions for performance critical segments
- #pragma thumb
  gcc -mthumb
  gcc -mthumb-interwork

Thumb Registers

- r8 – r12 registers are not visible
- MOV, CMP, and ADD instructions can access high registers