AKSHAYA INSTITUTE OF TECHNOLOGY

Lingapura, Tumkur-Koratagere Road, Tumkur-572106.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Prepared by: - Mrs. THIPPAMMA S
Assistant Professor, Department of CSE. Akshaya Institute of Technology, Tumakuru

AKSHAYA INSTITUTE OF TECHNOLOGY

Lingapura, Obalapura Post, Koratagere Road, Tumakuru - 572106

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

VISION

To empower the students to be technically competent, innovative and self-motivated with human values and contribute significantly towards betterment of society and to respond swiftly to the challenges of the changing world.

MISSION

M1: To achieve academic excellence by imparting in-depth and competitive knowledge to the students through effective teaching pedagogies and hands on experience on cutting edge technologies.

M2: To collaborate with industry and academia for achieving quality technical education and knowledge transfer through active participation of all the stake holders.

M3: To prepare students to be life-long learners and to upgrade their skills through Centre of Excellence in the thrust areas of Computer Science and Engineering.

Program Specific Outcomes (PSOs)

After Successful Completion of Computer Science and Engineering Program Students will be able to

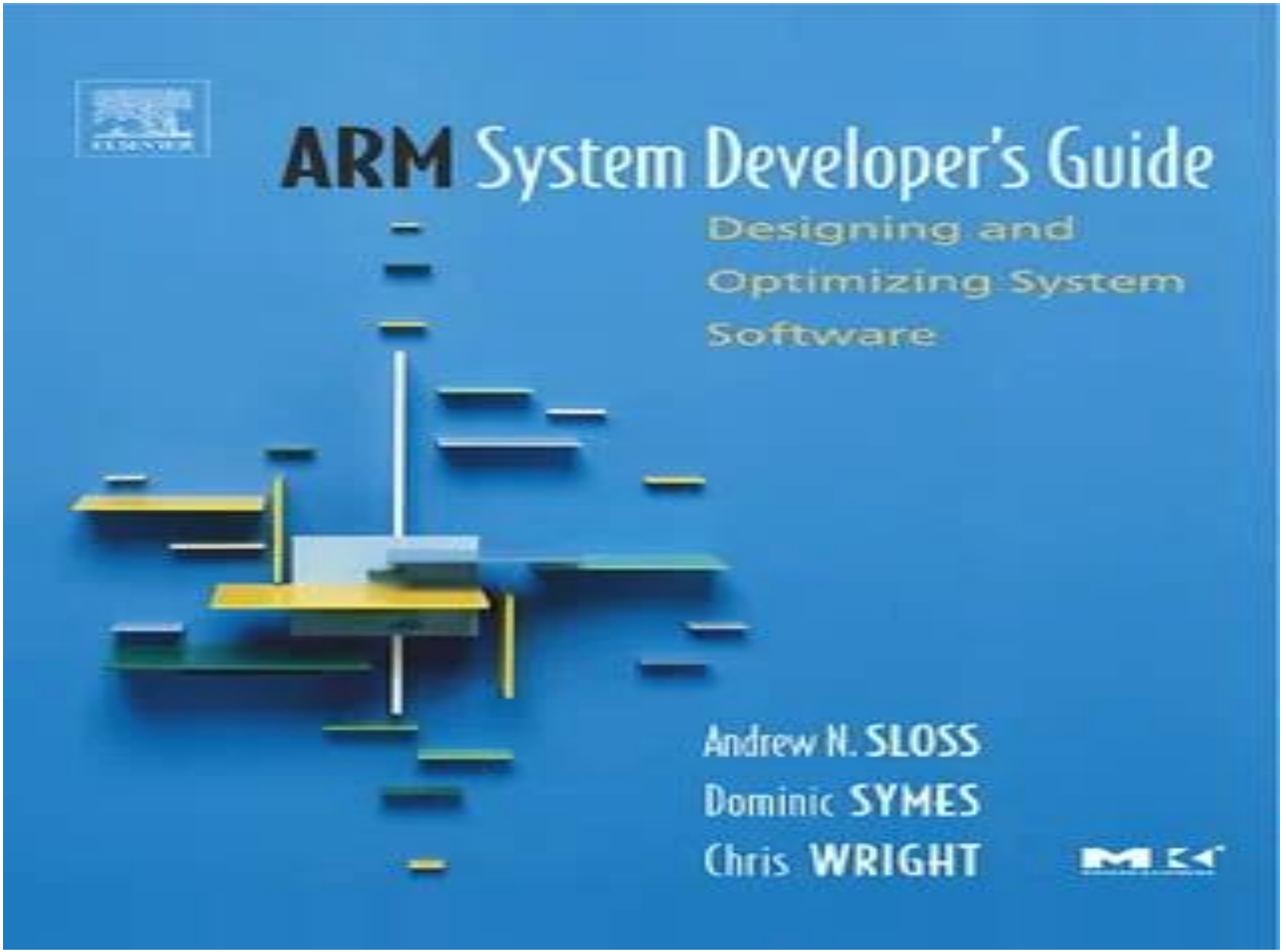
* Apply fundamental knowledge for professional software development as well as to acquire new skills.
* Implement disciplinary knowledge in problem solving, analyzing and decision-making abilities through different domains like database management, networking, algorithms, and programming as well as research and development.
* Make use of modern computer tools for creating innovative career paths, to become an entrepreneur or desire for higher studies.

Program Educational Objectives (PEOs)

PEO1: Graduates expose strong skills and abilities to work in industries and research organizations.

PEO3: Graduates engage in team work to function as responsible professional with good ethical behavior and leadership skills.

PEO3: Graduates engage in life-long learning and innovations in multi disciplinary areas.


ARM System Developer's Guide
Designing and Optimizing System Software
Andrew N. SLOSS
Dominic SYMES
Chris WRIGHT

Course Name :MICROCONTROLLERS

Course Code : BCS402

Semester : 4

CIE Marks : 50

SEE Marks : 50

Total Marks :100

Credits :04

Exam Hours : 3

Examination nature (SEE) : Theory

Teaching Hours/Week (L:T:P: S) : 3:0:2:0

Total Hours of Pedagogy : 40 hours

Course Objectives

CLO 1: Understand the fundamentals of ARM-based systems and basic architecture of CISC and RISC.

CLO 2: Familiarize with ARM programming modules along with registers, CPSR and Flags.

CLO 3: Develop ALP using various instructions to program the ARM controller.

CLO 4: Understand the Exceptions and Interrupt handling mechanism in Microcontrollers.

CLO 5: Discuss the ARM Firmware packages and Cache memory polices.

Course outcomes Course Skill Set)

(At the end of the course, the student will be able to:

- Explain the ARM Architectural features and Instructions.
- Develop programs using ARM instruction set for an ARM Microcontroller.
- Explain C-Compiler Optimizations and portability issues in ARM Microcontroller.
- Apply the concepts of Exceptions and Interrupt handling mechanisms in developing applications.
- Demonstrate the role of Cache management and Firmware in Microcontrollers.

Suggested Learning Resources

Text Books: 1. Andrew N Sloss, Dominic Symes and Chris Wright, ARM system developers guide, Elsevier, Morgan Kaufman publishers, 2008.

Reference Books:

1. Raghunandan.G.H, Microcontroller (ARM) and Embedded System, Cengage learning Publication, 2019.
2. Insider's Guide to the ARM7 based microcontrollers, Hitex Ltd.,1st edition, 2005

Activity Based Learning (Suggested Activities in Class)/ Practical Based Learning Assign the group task to demonstrate the Installation and working of Keil Software

Module 1

ARM Embedded Systems

- The RISC design philosophy
- The ARM Design Philosophy
- Embedded System Hardware
- Embedded System Software.

ARM Processor Fundamentals

Registers

Current Program Status Register

Pipeline

Exceptions

Interrupts

Vector Table

Core Extensions

Textbook 1: Chapter 1 - 1.1 to 1.4, Chapter 2 - 2.1 to 2.5 RBT: L1, L2, L3

Chapter 1

ARM Embedded System

- RISC Design Philosophy
- ARM Design Philosophy
- Embedded Hardware
- Embedded Software

The Days of ARM

- ARM's designers have come a long way from the first ARM1 prototype in 1985.
- Over one billion ARM processors had been shipped worldwide by the end of 2001.
  - simple and powerful original design.

- In fact, the ARM core is not a single core, but a whole family of designs sharing similar design principles and a common instruction set.
  - ARM7TDMI: one of ARM's most successful cores.
    - » 120 Dhrystone MIPS (a small benchmarking program)

Current and Future of ARM

- Cortext-M3
  - Thumb-2 Instruction Set
- MPCore
  - SMP(balanced), Cache consistency, L2 cache
  - 1~4 ARM11
- OptimoDE technique
  - Configurable VLIW, Co-work with ARM core
  - MPEG4, H.264 algorithm

Brief History of ARM Core

- 1985, First ARM (ARM1)
- 1995, ARM7TDMI
  - Most successful ARM core
  - 3-stage pipeline, 120 Dhrystone MIPS
- 1997, ARM9
  - 5-stage pipeline
  - Harvard (I+D cache), MMU (OS's VM)
- 1999, ARM10
  - 6-stage pipeline
  - VFP(Vector Float Point) (7-stage pipeline)
- 2003, ARM11
  - 8-stage pipeline

Versions of ARM Architecture

- ARMv1
  - 26-bit address
- ARMv2
  - 32-bit Multiplier/coprocessor
- ARMv3
  - 32-bit address, cpsr/spsr, MMU, undef/abort Mode
- ARMv4
  - Load/store (sign/half/byte), sys Mode
- ARMv5
  - Superset ARMv4T (Thumb), extend Mul/DSP
- ARMv6
  - Multiprocessor support instr., unaligned/endian/MMX

Others

- StrongARM
  - ARM + Digital Semiconductor
  - Intel Patent
- Xscale
  - 1GHz, V5TE
- SC100
  - Security, Low Power
  - ARM7TDMI, MPU

Nomenclature of ARM

- E.g. ARM7TDMI
  - T: Thumb
  - D: JTAG
  - M: Multiplier (extend)
  - I: ICE
  - E: Extend Instruction (above TDMI)
  - J: Jazelle
  - F: Float point
  - S: Synthetic (soft core)

ARM, a RISC ?

- Philosophy of RISC design
  - Instruction
    - » RISC processor have a Reduced number of instruction classes. These classes provide simple operations that can each execute in a single cycle. The compiler or programmer synthesized complicated operations (e.g. a divide operation) by combine several simple instructions.
  - Pipeline
    - » The processing of instructions is broken down into smaller units (stage) that can be executed in parallel by pipelines. There is no need for an instruction to be executed by a mini-program (microcode) as on CISC processor.
  - Register
    - » RISC have a large General Purpose Registers (GPR) set.
  - Load/store architecture
    - » Separating memory access from data processing.

ARM, a RISC ?

- The ARM Design Philosophy
  - There are a number of physical features that have driven the ARM processor design:
    - » Low Power Consumption: Smallest Core;
    - » Limited Memory: High code density;
    - » Die density: Simple Hardware Executive Unit
  - The ARM core is not a pure RISC architecture because of the constraints of its primary application – the embedded system.
- Simplicity favors regularity ?
  - These design rules allow a RISC processor to be simpler, and thus the core can operate at higher clock frequencies.

Instruction Set for Embedded System

- The ARM instruction set differs from the pure RISC definition in several ways
  - make the ARM suitable for embedded application
    - » Variable cycle execution for certain instruction
      - Not every ARM instruction executes in a single cycle.
    - » More complex instruction (inline barrel shifter)
      - This expands the capability of many instructions to improve the core performance and code density.
    - » Thumb 16-bit instruction set
      - The Thumb instruction improve code density by about 30%.
    - » Conditional execution
      - Improves performance and code density by reducing Branch.
    - » Enhanced instruction
      - DSP instruction were added to the standard ARM instr-set to support fast 16x16-bit multiplier operations and saturation.

Example: SoC with ARM core

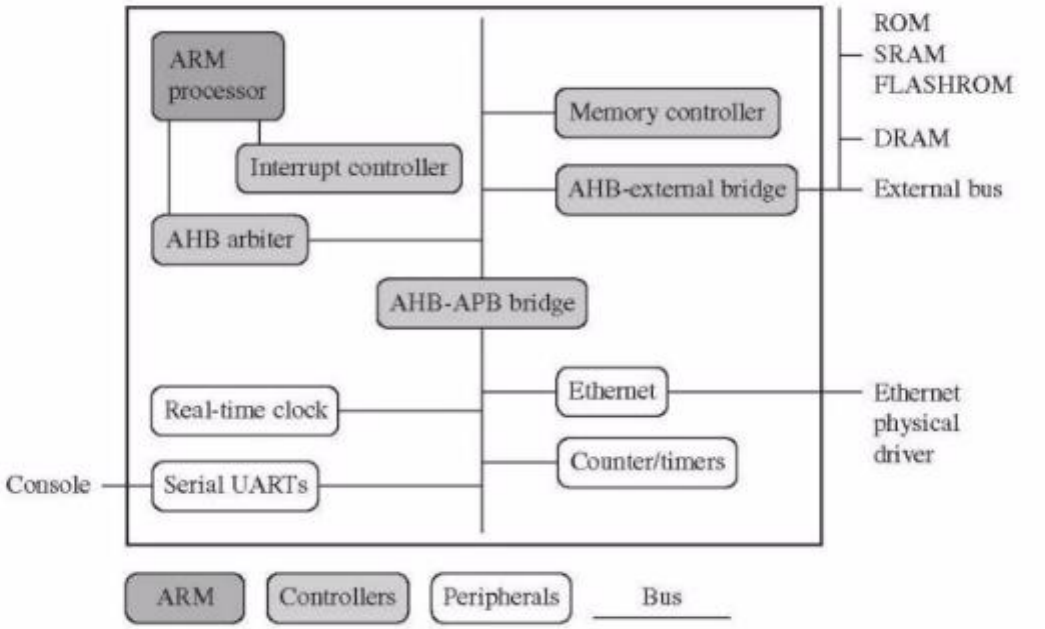


Figure 1.2 An example of an ARM-based embedded device, a microcontroller.

Units inside SoC

- SoC is an embedded device.
- We can separate the device into four main components:
  - ARM Processor: controls the embedded device.
    - » An ARM processor comprises a core (the execution engine that processes instructions and manipulates data), plus the surrounding components (MMU and caches) that interface it with a bus.
  - Controllers: coordinate important functional blocks (e.g. interrupt and memory controllers)
  - Peripherals: USB, LCD, etc.
  - Bus: is used to communicate between different parts of the device.

1.3.1 ARM Bus Technology

- Embedded systems use different bus technologies than those designed for x86 PC.
  - Embedded device use an on-chip bus
  - Core is master who initiates a data transfer.
- A Bus has two architecture levels
  - The First is a physical level that covers the *electrical characteristics* and *bus width* (16, 32, or 64 bits).
  - The Second level deals with *protocol*.- the logical rules governing the communication between processor and peripheral.
- ARM seldom implements the electrical characteristics of the bus, but it routinely specifies the bus protocol.

1.3.2 AMBA

- AMBA Advanced Micro controller Bus Architecture
  - 1996, it's introduced and widely adopted as the on-chip bus architecture for ARM processors.
  - The first AMBA buses introduced were
    - » ASB : ARM System Bus, and
    - » APB : ARM Peripheral Bus
  - Later, ARM introduced another bus design
    - » AHB: ARM High-performance Bus

- Using AMBA,
  - peripheral designers can reuse the same design on multiple projects (with different processor architecture).
  - Plug-and-play

AHB

- AHB
  - provides higher data throughput than ASB. Because
    - » It use a Centralized Multiplexed Bus Scheme (rather than ASB's bidirection bus).
    - » This change allows the AHB bus to run at higher clock speed.
    - » 64/128 bits width.

- Two variations on the AHB bus
  - » Multi-layer AHB, and
    - allows multiple active bus masters,
  - » AHB-Lite: only one master

1.3.3 Memory

- **Memory is necessary**
  - An embedded system has to have some form of memory to store and execute code.
- **You have to consider**
  - price, performance, and power consumption
- **Specific memory characteristics**
  - hierarchy, width, and type

Memory Hierarchy

- Cache
  - is used to speed up data transfer between Core and Main Memory (DRAM);
- But,
  - It makes the performance unpredicted;
  - It doesn't help Real-Time system response;
    - » Note that many small embedded systems do not require the benefit of a cache.
- * Cache
  - Elastic buffer (different speed between Core and Bus);
  - Width adaptive (e.g., 32-bit Core vs. 16-bit BUS)

Memory Types

- **DRAM**
  - the most commonly used RAM for devices;
  - Dynamic: need to have its storage cells refreshed and given a new electronic charge every few milliseconds, so you need to set up a DRAM controller before using the memory.
- **SRAM**
  - is faster than the more traditional DRAM (SRAM does not require a pause between data access).
- **SDRAM**
  - is one of many subcategories of DRAM.
  - accessed pipelined, transferred in a burst.

1.3.4 Peripherals

- Embedded system that interact with the outside world need some form of peripheral device.
  - Peripherals range from a simple serial communication device to a more complex 802.11 wireless device.
- All ARM peripherals are memory mapped – the programming interface is a set of memory addressed register.
- Controllers are specialized peripherals that implement higher level of functionality within an embedded system.
  - Two important types of controllers are
    - » Memory Controller
    - » Interrupt Controller
      - Normal IC
      - Vectoring IC
        - Priority
        - Simple Interrupt Dispatch

Memory Controllers

- **Memory Controllers: Connect different types of memory to the processor bus.**
  - On power-up a memory controller is configured in hardware to allow certain memory device to be active. These memory devices allow the initialization code to be executed.
  - Some memory devices must be set up by software.
    - » e.g. When using DRAM, you first have to set up the memory timings and refresh rate before it can be accessed.

Interrupt Controller

- When a peripheral or device requires attention,
  - it raise an interrupt to the processor.
- An interrupt controller
  - provides a programmable governing policy

- There are two types of interrupt controller available for the ARM processor

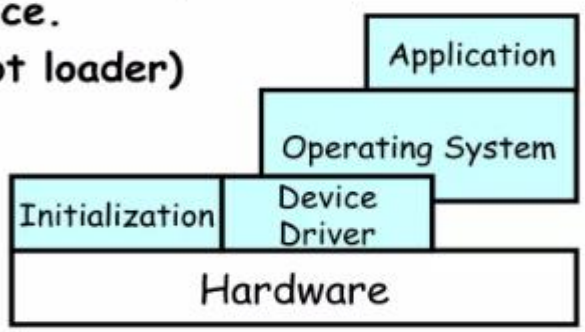
  - Standard interrupt controller
    - » Sends an interrupt signal; Can be programmed to ignore or mask an individual or set of devices.
    - » It's interrupt handler determines which device requiring service.

  - Vector interrupt controller (VIC)
    - » Associate a "priority" and a "handler address" to each interrupt.
    - » Depending on its type, VIC will either call the standard interrupt exception handler (loading the handler address from VIC) or cause core to jump to the handler for the device directly.

1.4 Embedded System Software

- An embedded system needs software to drive it.

- There are four typical software components required to control an embedded device.
  - » Each software component in the stack uses a higher level of abstraction to separate the code from the hardware device.
  - Initialization Code (e.g. Boot loader)
  - Operating System
  - Device Drivers
  - Application



Initialization (BOOT) Code

- Initialization code (or boot code)
  - takes the processor from the reset state to a state (where the operating system can run).
    - » Configuring memory controller, caches
    - » Initializing some devices
    - » * Debug Monitor (replace OS in simple system)
- Three phases
  - Initial hardware configuration
    - » Satisfy the requirements of the booted image
      - e.g. re-organization of the memory map
  - Diagnostics
    - » Fault identification and isolation
  - Booting
    - » Loading an image and handing control over to the image
    - » The boot process may be complicated if the system must boot different operating systems or different versions of the same operating system.

Example: Memory Reorganization

- Start from ROM
- Remap to RAM
  - easy IVT modification

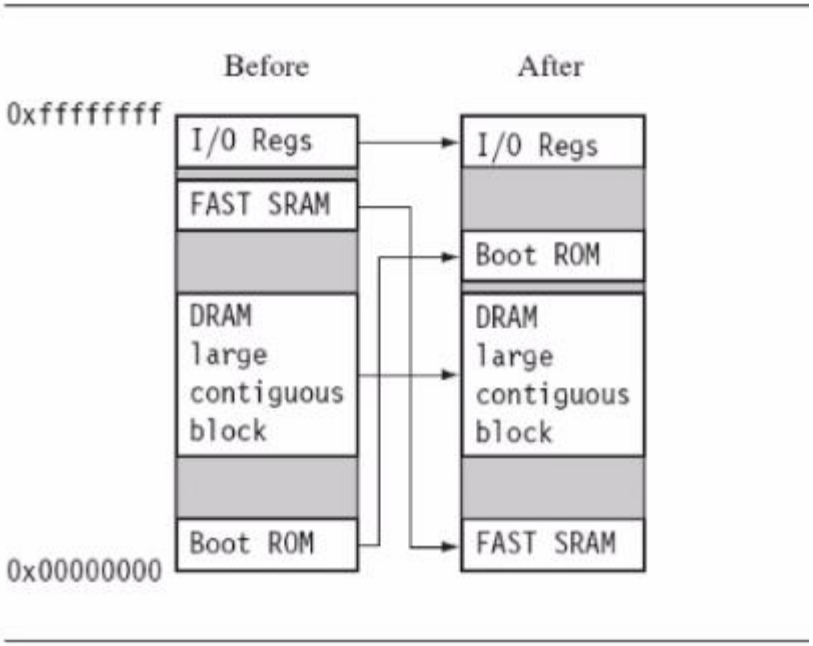


Figure 1.5 Memory remapping.

Operating System

- OS organizes the system resources
  - peripherals, memory, and processing time
    - » With an OS controlling these resources, they can be efficiently used by different applications running within the OS environment.
- ARM processors support over 50 OSes
  - Two main categories: RTOS, platform OS
    - » RTOS: guarantee response times to event
    - » platform OS: require MMU and tend to have secondary storage (for large application).
      - N.B., These two categories of OSes are not mutually exclusive.
  - ARM has developed a set of processor cores that specially target each category.

Applications

- The OS schedules applications
  - code dedicated to handling a particular task.
- ARM processors are found in numerous market segments, including
  - networking, automotive, mobile and consumer devices, mass storage, and imaging.
- In contrast, ARM processors are not found in applications that require leading-edge high performance.

1.5 Summary (1)

- Pure RISC is aimed at high performance
  - But ARM uses a modified RISC design philosophy that also targets
    - » good code density and low power consumption.

- The Key points in a RISC design philosophy are
  - Reducing the complexity of instructions
    - » improve performance;
  - Pipeline
    - » speed up instruction processing;
  - Large register set
    - » store data near core;
  - Load-store architecture;

1.5 Summary (2)

- The ARM design philosophy also incorporates some non-RISC ideas
  - Variable cycle execution on certain instruction
    - » save power, area and code size
  - Barrel Shifter
    - » expand the capability of certain instructions
  - Thumb 16-bit instruction set
    - » improve code density
  - Conditional Executing Instruction
    - » improve code density and performance
  - Enhanced Instructions: e.g. DSP

1.5 Summary (3)

- Hardware Components in ARM Processor
  - Peripherals
    - » accessed via memory-mapped registers
  - Controller (a special type peripheral)
    - » Higher-level functions: e.g. memory and interrupts.
  - AMBA Bus
    - » connect the processor and peripherals
- Software Components
  - Initialization Code
  - Operating System
  - Device Driver
  - Application

Data-path in ARM

- ALU
  - Sources: Rn, Rm(shifted);
  - Destination: Rd
- Shifter
  - Helps to Extend the scope of data or address
- Sign-extend
  - LOAD data from Main memory

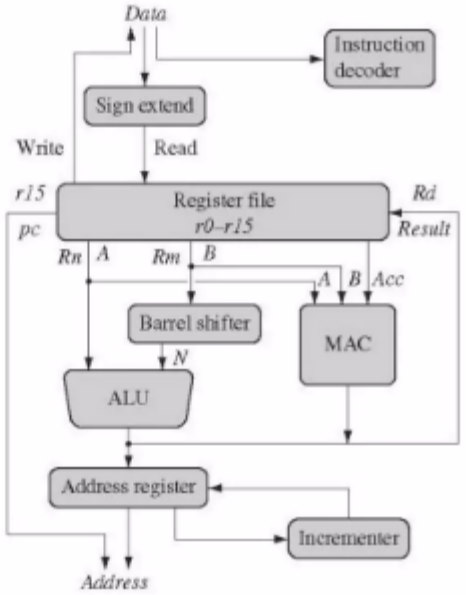


Figure 2.1 ARM core dataflow model.

Register in ARM

- Orthogonal Registers (ref. VAX, PDP-11)
  - We say R0~R13 are orthogonal, for given instruction, if it can use R0, then others can also be used.
- SPRs
  - R13(sp), R14(lr), R15(pc)
- CPSR/SPSR
  - Condition Codes: N, Z, C, V
  - Interruption mask: I(IRQ), F(FIQ)
  - Thumb Enable Bit
  - Mode(5-bit)

Instruction Sets in ARM

- Three Instruction Set (IS) in ARM
  - ARM
  - Thumb: 16-bit
  - Jazelle(closed): 8-bit
    - » 60%: Hardware (JVM)
    - » 40%: Software

Condition Codes

- N(egative), Z(ero), C(arry), Q(ov), V(signed ov).
- EQ = Z, NE = z; HS = C, LO = c
- GE = NV or nv, LT = Nv or nV

Pipeline

- ARM7
  - 3 stages: Fetch, Decode, Execute
- More stages (deeper pipeline)
  - means "More latency", "More Dependence"
- ARM9 (+13% ARM7)
  - 5 stages: FI, DI, EX, M, WB
- ARM10 (+34% ARM7)
  - 6 stages: FI, Issue, DI, EX, M, WB
- ARM7 instruction runs on ARM9/10 ?
  - Yes, same pipeline architecture as ARM7

About Pipeline

- **Enable IRQ**
  - MSR: Clear CPSR's I bit.
  - IRQ is enabled only after MSR's third stage (WB);
- **PC**
  - PC always point to the Current "FIing" instruction;
  - It's a tricky for pipeline when calculating PC offset;
- **When Branch or direct PC updating**
  - ARM core will flush the whole pipeline;
  - ARM10: using Branch Predict Technology;
  - When being IRQed: Instruction in EX will insist to finish, other instructions will be flushed.

Vector Table

- Reset
  - the 1st instr. after power-up;
- Undef
  - cannot be decoded;
- Soft
  - SWI instruction being executed;
- Prefecth Abort (PABT)
  - try to access invalid address for instruction;
- Data Abort (DABT)
  - Try to access invalid address for data;
- IRQ
- FIQ

Core Extension

- Cache & TCM
  - Unified vs. I/D
  - TCM: fast SRAM, very near Core (unwired with AMBA)
- MM interface
  - No MM: for simple embedded system;
  - MPU(Memory Protect Unit): section protection;
  - MMU: Translation table, Fine-grain protection;
- CP interface
  - By Extend Instruction Set vs. CSR register;
  - E.g.
    - » VFP instruction;
    - » CP15: cache, TCM and MMU via load/store like instr.

ARM Instruction Set

- Data
  - Data transfer(MOVE), Arith/Logic, CMP, MUL;
- Branch
  - If-then-else
- Load/Store
- SWI
- MRS/MSR
  - Status (CPSR or SPSR) <-> Register
  - Coprocessor Instruction
    - » CDP (CP Data Processing), MRC/MCR, LDC/STC
- CONST load
- ARMv5E extension
- Condition Executed Instruction
  - E.g. ADDEQ r0, r1, r2

Try to answer these Questions

1) Explain RISC design philosophy

2) Write a note on Registers available in user mode.

3) Explain Concept of Pipeline in ARM

4) Explain a simplified Harvard architecture with TCMs.

5) Explain Software abstraction layers executing on hardware

6) Briefly explain Processor modes

Try to answer these Questions

7) Explain ARM design philosophy

8) Explain ARM core dataflow model

9) Write a note on Exceptions, Interrupts, and the Vector Table

10) Distinguish between CISC vs. RISC.

11) Write a note on Memory of ARM

12) Explain Complete ARM register set