

Week 1. Introduction, Logistics & ARM: Architecture

18-342: Fundamentals of Embedded Systems

Rajeev Gandhi

INI & ECE
Carnegie Mellon University

Carnegie Mellon



Overview of Lecture

- Welcome to 18-342!
- Introduction to the course
 - Logistics, grading criteria, policies
- Course support: TA, mailing list
- Expectations
- Brief introduction to embedded systems

Course TAs

- 4 TAs (Office hours TBA on Blackboard)
- Sujay Jain sujayj@andrew.cmu.edu
- Puneet Pruthi ppruthi@andrew.cmu.edu
- Amit Chandra amitc@andrew.cmu.edu
- Chinmay Kamat ckamat@andrew.cmu.edu
- Office hours for TAs
 - 2 hours per week for each TA
 - A TA will be available for you in the lab 4 out of 5 days of the work-week
 - There will be at least one TA in the lab for two hours

Administrivia

- Lectures
 - Tues & Thurs 9:00 - 10:20 am US East Coast time
 - Friday 10:00-10:50 am US East Cost time
- Labs
 - Hammerschlag Hall 1204
- BlackBoard
 - <http://www.cmu.edu/blackboard>
 - Course ID: F12-Fundamentals of Embedded Systems
 - Log in using your andrew id and password
- Mailing Lists
 - 18342-fall12@lists.andrew.cmu.edu: For course discussions/information with everyone (all students, instructor, all TAs)
 - 18342-fall12-ta@lists.andrew.cmu.edu: For questions to course staff (instructor+TAs)
- Textbooks
 - None!
 - Supplemental readings to help you understand lecture details

Pre-Requisite Knowledge/Skills

- Basic knowledge of digital logic
 - Basic gates and their truth tables
- Basic, working knowledge of computer architecture
- Knowledge of (and experience with) C programming
 - Fundamentals of data structures and algorithms
 - Just knowing Java is not enough
 - Need to know, understand and manipulate “pointers”
- Experience with assembly-language programming
 - XScale (ARM-based) instruction set will be used in 342
 - Knowledge of instruction sets of other modern microprocessors (x86, ARM, PowerPC, 680x0, MIPS) should be a sufficient and necessary starting point
- If you do not know assembly language programming, you need to spend effort in getting up to speed right away
 - Learning materials are available online on BlackBoard course web-site

Quizzes, Exams

- All are open notes, open book
- Quiz and midterm schedule posted on Blackboard already
- 3 Quizzes
 - Duration of quizzes – 75 minutes (or less) in regular class hours
 - Will drop the scores of the worst quiz
 - Will cover material that includes the lecture held on the day of the previous quiz to the lecture held on the day before the quiz
- Mid-term exam
- Held in regular class hours
 - Covers all the material taught until the mid-term
- Final exam
 - Scheduled by the registrar’s office
 - Held in finals week (lasts three hours)
 - Covers **all** the material taught in the course

Grading Criteria

- **Breakdown of grading**

- Class participation = total of 5%
- 2 quizzes, each 7.5% of your grade = total of 15% (we will select the best 2 out of 3)
- 1 Midterm exam, 15% of your grade = total of 15%
- 1 Final exam, 25% of your grade = total of 25%
- 4 lab projects, each 10% of your grade = total of 40%

- **Grading Scale**

– 93% - 100%	A	77% - 80%	C+
– 90% - 93%	A-	73% - 77%	C
– 87% - 90%	B+	70% - 73%	C-
– 83% - 84%	B	60% - 70%	D
– 80% - 83%	B-	Below 60%	R

- **Late Policy**

- 10% of the lab grade per-day penalty (including Saturday and Sunday)
- Last day to turn in everything is the last day of class

Lab Projects

- We have access to a lab in Hammerschlag Hall (HH1204)

- Lab projects also structured to be doable in your dorms with your laptops
- We will give you an entire hardware kit – boards, cables, connectors...and instructions! (kits must be returned intact, to receive your grade)

- Four lab projects

- A warm-up exercise to get you familiar with the hardware and the software tools we will use in the other labs
- Three other projects involve: software and hardware interrupt handling, optimization and profiling, serial communication, real-time multitasking and priority-inversion handling

- All labs must be done in teams

- Teams of 3, depending on class enrollment and availability of hardware
- All members of teams must be present at demonstrations in the lab
 - Should understand the code that the team has written, be able to explain the concepts and thought process behind the solution, answer questions from TAs

Policies on Collaboration

✓ What is okay

- Collaboration is encouraged for group projects within your team
 - Intended to teach you team spirit and the benefits of team work
 - Projects/demos should be substantially the result of your team's efforts
 - Reports should be substantially the result of your team's efforts
- Sharing your findings with others, especially over something tricky
 - Clarifying doubts posted by your fellow-students' questions on BlackBoard (without posting solution or code)

• What is not okay

- Using/borrowing code snippets (modified or otherwise) from other teams in order to make a project deadline
- Letting someone in your team carry the lion's share of the load in the project
- Cheating during quizzes, mid-term exam or final exam
- *Using solutions from prior years*
- *Do not hand over solutions to students for use in successive years*

Cheating and Plagiarism

- Your code may be (and will be) compared against code submitted by other groups and/or code submitted by groups in prior years
- CMU cheating policies
 - Please read and understand the Academic Integrity section of the course syllabus
 - <http://www.cmu.edu/policies/documents/Cheating.html>
- Cheating penalty
 - An "R" grade in the course and notification to the department
- Quizzes, mid-term exam and final exam indicate **individual** ability
 - Any collaboration in these exams will be considered cheating
- Copyright warning
 - Please do not post or otherwise redistribute materials distributed in class.

Expectations

- You are responsible for holding up your end of the educational bargain
 - We expect you to attend classes and to complete assigned lab projects
 - We expect you to learn how to use the tools and to try things out for yourself
 - We expect you to do any *required* reading that we provide off BlackBoard
 - We expect you to manage your time
 - There are three quizzes, one mid-term, one final and four labs
- Learning material in this course requires participation
 - This is not a sit-back-and-listen kind of course; class participation is going to help you learn more and to get what you came here for!
 - Questions are welcome and appreciated
 - If you see something that is a problem, you need to tell us right away
 - Project issues, classroom issues,
- **Feedback**
 - **Talk to us** if you have any concerns regarding *anything* related to the course.
 - I will want to **talk to you** if *you* are **not** doing well in class.
 - **Mid-term survey** to get your feedback – your opinion matters to me!
 - You can also provide **anonymous feedback anytime**

Getting Help

- “Showing up is 80% of life.” -- Woody Allen
 - We expect an “average” CMU student will put in 12 hours/week
 - If you start to see yourself exceed the number of hours greatly, then
 - You might need to brush up on some background knowledge
 - You might be approaching some lab projects in the wrong way
 - We want to know, either way – please come and talk to us!

What *are* Embedded Systems anyway?

Embedded Systems: An Introduction

- What is an embedded system?
- Several definitions
 - “Computer systems in other devices where the presence of a computer is not immediately obvious”
 - A computer-based (microprocessor-based) system that is used to control a variety of functions but is not designed to be a general-purpose computer
 - More than just a computer, often interacts with physical world/environment
- The user “sees” a smart (special-purpose) system as opposed to the computer inside the system
 - “How does it do that?”
 - The end-user typically does not or cannot modify or upgrade the internals
- Anything that uses a microprocessor but isn't a general-purpose computer
 - Cell Phones, Set-top Boxes, Televisions, Video Games, Refrigerators, Cars, Planes, Elevators, Remote Controls, Alarm Systems ...

Embedded in Your Daily Life

- Average American household has about 40 microprocessors
 - 50, if you count your PC and its baggage
 - Bathroom scale with a digital readout
 - Iron that turns itself off automatically
 - Electronic toothbrush (with ~3000 lines of code)
 - Cooking range
 - Laundry machine
 - Toaster
 - Microwave
 - Furby (the toy that has more processing power than the original lunar lander)
- Statistics from Dataquest
 - Number of embedded processors sold in 1998 was 4.8 billion
 - Only about 120 million of these (~2.5%) were intended for PCs
 - In five years, the average number of chips in the home could grow to 280 and the number sold to 9.4 billion
- “It may seem like the era of the personal computer. It is just as

Source: Honey, I programmed the blanket, Katie Hefner, *New York Times*, 1999

What Makes Embedded Systems Different?

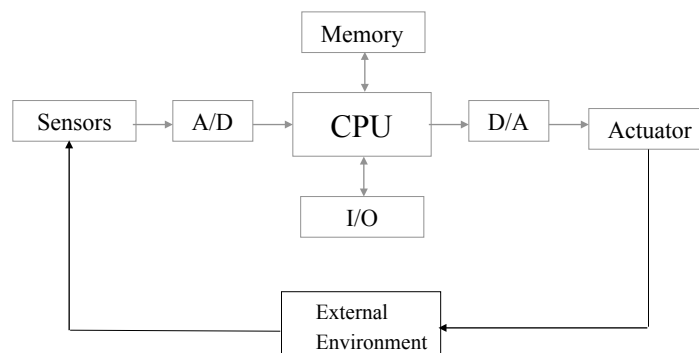
- What makes embedded systems different?
 - Embedded systems are typically single-functioned i.e., the system usually executes a single function (or a few functions) repeatedly
 - Only those hardware components are retained that are essential to provide the application's functionality
 - The application software is carefully tuned to the system
 - Many embedded systems are used in safety critical applications
 - Embedded systems are reactive
 - Continually react to changes in the system's environment and must compute certain results repeatedly
 - Embedded systems often involve control-loops
 - Many embedded systems are expected to have additional properties like real-time operation
 - Embedded systems are often tightly constrained and many constraints often compete with each other

Constraints On Embedded Systems

- Low memory footprint
 - Memory can be a substantial portion of system costs
- Small Size, Low Weight
 - Hand-held electronics
- Low Power
 - Limited cooling may limit power even if AC power available
- High Performance
 - Often the system must compute results in relatively short period of time
- Low costs
 - For high volume systems, even \$0.05 increase in price can add up
- Harsh environment
 - Heat, vibration, shock
- Safety-critical operation
 - Must function correctly
 - Must not function incorrectly

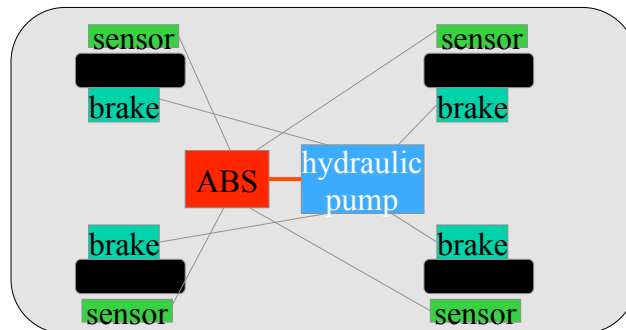
Embedded Systems are Reactive

- Many embedded systems are required to continually react to changes in the system's environment
 - Such embedded systems often involve sensors and actuators in the system architecture



Antilock Braking System (ABS) – An Example of Reactive Embedded Systems

- **Problem** – When brakes are applied, car wheels stop turning suddenly, car starts skidding and is harder to maintain control over the car
- ABS – Temporarily release the brake on the wheel when it rotates too slowly
- Interdisciplinary: There are safety, reliability, timing issues!



Source: "Computers as Components" by W. Wolf

Embedded Systems Are Often Time-Critical

- **Real-Time: timing correctness is part of system correctness**
 - **Hard real-time**
 - Absolute deadline, beyond which answer is useless
 - May include minimum time as well as maximum time
 - **Soft real-time**
 - Missing a deadline is not catastrophic
 - Utility of answer degrades with time difference from deadline
 - **Example:**
 - In a digital set-top box, time to process a frame is limited (~30 msec) before the next frame arrives

“Real-Time” Does Not Mean “Fast”

- “Real-time” does not necessarily mean “fast”
 - Real-time = guarantee that timing constraints will be met all the time
 - Fast = doing something quickly (usually on average)
- Real-time system can be slow
- Of course, most real-time systems are designed to be fast and to operate at high speeds

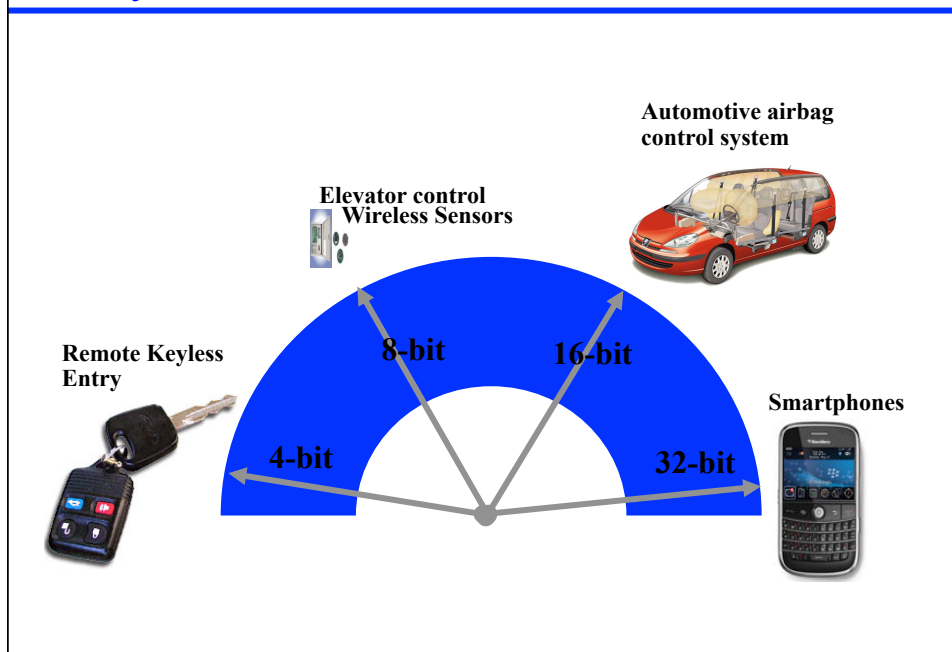
Embedded Applications Require Robustness

- Example: Ariane-5
 - Development cost \$7 billion
 - Rocket and cargo worth \$500 million
 - Software error in inertial reference system caused the failure
 - 64-bit float converted to 16-bit signed integer
 - Improperly handled exception caused by variable overflow

Embedded Systems Design

- Embedded problems typically solved using one of the following three approaches
 - Use a combination of custom designed hardware and possibly some software on an embedded processor that is integrated with the hardware
 - Use custom software designed to run on an off-the-shelf embedded processor
 - Use an application specific processor (a processor that has been optimized to run the specific class of applications efficiently) with custom software

Variety of Embedded Processors



Why Should I Learn About Embedded Systems?

- I am taking other software/systems courses, would n't I learn embedded systems as well?
 - Writing applications that can run without an OS
 - Challenges
 - Where is my `fork()`, `wait()`, `exec()`,....? Why can't I use `connect()`, `bind()`, `accept()`,....?
 - How do I protect applications from one another or the OS from applications?
 - Homogeneity vs. Heterogeneity
 - Enterprise systems are homogeneous in terms of the processor used
 - Embedded systems tend to be heterogeneous
 - Different development/debugging/testing cycle
 - Help! My system does not have a keyboard or a monitor! How do I debug my program?

Course Objectives

High-Level Goals

1. Understand the scientific principles and concepts behind embedded systems, and
2. Obtain hands-on experience in programming embedded systems

By the end of the course, you should be able to

- Obtain direct hands-on experience on both hardware and software elements commonly used in embedded system design.
- Understand basic real-time resource management theory
- Understand, and be able to discuss and communicate intelligently about
 - Embedded processor architecture and programming
 - I/O and device driver interfaces to embedded processors with networks, multimedia cards and disk drives
 - OS primitives for concurrency, timeouts, scheduling, communication and synchronization

What Will You Learn In This Course?

- **Hardware**
 - Embedded processor architectures
 - Timers, RS232, Flash memory
 - Buses
 - I/O
- **Software**
 - Device drivers, interrupt & exception handlers, low-level OS issues, code optimization techniques
 - Scheduling, concurrency and deadlocks, virtual memory and memory protection,...
 - Fundamentals of real-time systems
 - Real-time scheduling, priority inversion
- You are expected to do a lot of low-level programming in this course

What You Should Be Doing **Now**

- ◆ Form your team of 3 people

Up Next: Fundamentals Review & ARM Architecture

- ◆ Fundamentals of computer architecture
- ◆ Most of this stuff you should have covered before
 - Part of the prerequisites for this course
- ◆ Overview of ARM's architecture from a programmer's point of view
 - Processor modes
 - General purpose registers

29

ARM Ltd.

- ◆ Founded in November 1990
 - Spun out of Acorn Computers based in U.K.
- ◆ Designs the ARM range of RISC processors
- ◆ Licenses ARM core designs to semiconductor partners who fabricate and sell to their customers
- ◆ ARM Ltd. does not fabricate processors itself
- ◆ Also develop technologies to assist with the design-in of the ARM architecture
 - Software tools, boards, debug hardware, application software, bus architectures, peripherals etc.

30



Machine Code

Code in
high-level
language
like C

```
void swap(int a,int b)
{
    tmp = a;
    a = b;
    b = tmp;
} /* end swap() */
```

compiler

Assembly
language
program

```
MOV    r2,r0
MOV    r0,r1
MOV    r1,r2
MOV    pc,r14
```

assembler

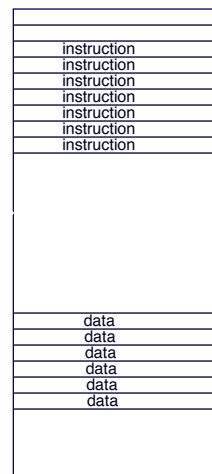
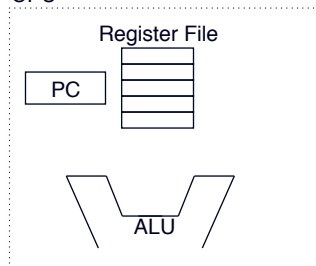
Machine
language
code

```
0xe1a02000
0xe1a00001
0xe1a01002
0xe1a0f00e
```

31

The Big Picture

CPU



Memory

32

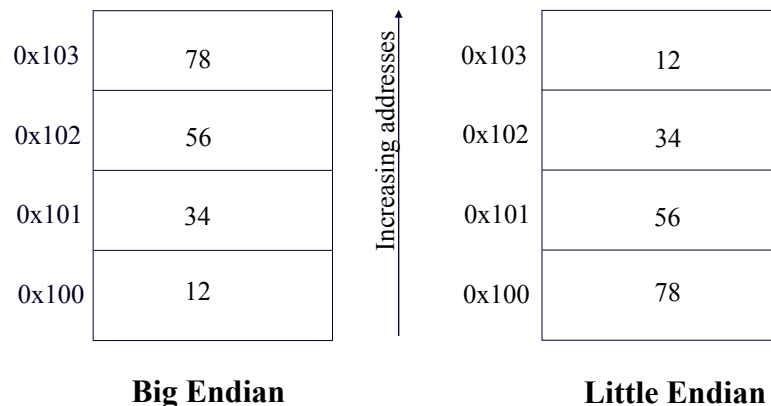
Memory

- ◆ Memory can be viewed as a linear, byte-addressable array
- ◆ Many data-types use multiple bytes for their representation
 - E.g., integers can take 4 bytes
- ◆ **Data alignment**
 - CPUs expect fundamental data-types (e.g., integers) to be stored in memory addresses that are multiples of their length
 - Why? Because CPUs are optimized to access memory this way
 - Unaligned accesses might sometimes be allowed, but at a performance cost
 - In some cases, unaligned accessed might lead to error conditions
 - Padding might be a compiler's way of enforcing alignment
 - Alignment and padding rules depend on the processor
- ◆ If a data-type uses multiple bytes, how are those multiple bytes stored in memory?
 - Most significant byte could be at lowest address or least significant byte could be at the lowest address

33

Big Endian vs. Little Endian

- ◆ How is a word, say 0x12345678, stored in memory?



34

Big Endian vs. Little Endian

◆ Big-endian

- Most significant byte of any multi-byte data field is stored at the lowest memory address
- Big-end first
- Reading from left to right
 - Natural way you read
- SPARC & Motorola processors

◆ Little-endian

- Least significant byte of any multi-byte data field is stored at the lowest memory address
- Little-end first
- Reading from right to left instead
- Intel processors

- ◆ Some processors can be configured either way, e.g., PowerPC

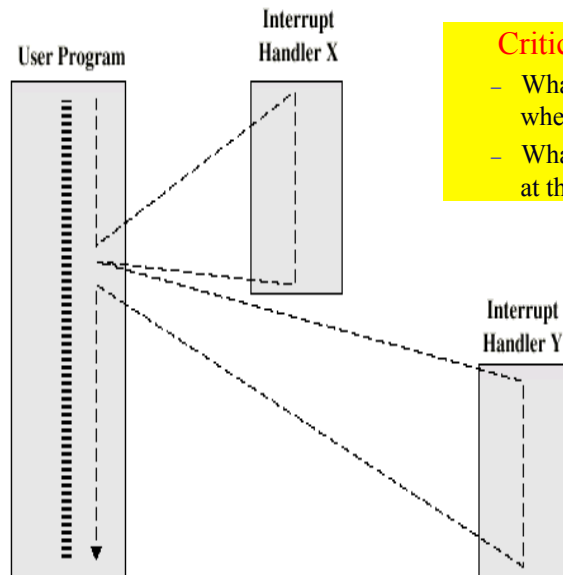
35

Interrupts

- ◆ Mechanism by which other entities can interrupt the normal sequence of processing by the CPU
- ◆ Source can be input-output
 - I/O controller can interrupt when data is ready
- ◆ Source can be the timer
 - New task needs to be scheduled
 - A time-driven event needs to occur
- ◆ Source can be the application program
 - New task created

36

Interrupt Handlers



Critical Thinking

- What happens to normal execution when an interrupt occurs?
- What if you get multiple interrupts at the same time?



37

Microprocessor Architecture types -- CISC vs. RISC

- ◆ RISC – Reduced Instruction Set Computers
- ◆ CISC – Complex Instruction Set Computers
- ◆ Different architectures for doing the same operations
- ◆ Suppose you wanted to multiply two numbers in memory locations *mem0* & *mem1* and store the results back in *mem0*

CISC Approach

```
mul mem0, mem1
```

RISC Approach

```
ldr r0, mem0  
ldr r1, mem1  
mul r0, r0, r1  
str mem0, r0
```

- ◆ Same result but the complexity of operations and the number of steps used in the two cases differ

38

RISC versus CISC architectures

CISC	RISC
<ul style="list-style-type: none"> ◆ Example – Intel x86 chips ◆ Large number of instructions ◆ Variable length instructions, instructions can range from 1-15 bytes <p>Example</p> <pre>push %ebp → 0x55 add %eax, 0x8049464 → 0x01 0x05 0x64 0x94 0x04 0x08</pre> <ul style="list-style-type: none"> ◆ Some instructions can have long execution times 	<ul style="list-style-type: none"> ◆ Examples – SPARC, PowerPC, ARM ◆ Few instructions, typically less than 100 ◆ Fixed length instructions, all instructions have the same number of bytes <p>Example</p> <pre>mov r2, r0 → 0xe1a02000 ldr r15, r15, #100 → 0xe559FF100</pre> <ul style="list-style-type: none"> ◆ No instruction with a long execution time

RISC versus CISC architectures (contd.)

CISC	RISC
<ul style="list-style-type: none"> ◆ Arithmetic and logical operations can be applied to memory and register operands ◆ Stack intensive procedure linkage <ul style="list-style-type: none"> – Stack is used for procedure arguments and return values ◆ Compiled code size is typically small ◆ More transistors used for storing/ executing more instructions → more power is consumed 	<ul style="list-style-type: none"> ◆ Arithmetic and logical operations only use register operands <ul style="list-style-type: none"> – Memory contents have to be loaded into registers first – Referred to as load/store architecture ◆ Register intensive procedure linkage <ul style="list-style-type: none"> – Registers used for procedure arguments and return values ◆ Compiled code size is larger ◆ Fewer transistors are used for storing/ executing fewer instructions → less power consumed



ARM Data Sizes and Instructions

- ◆ The ARM is a 32-bit RISC architecture
- ◆ When used in relation to the ARM
 - **Byte** means 8 bits
 - **Halfword** means 16 bits (two bytes)
 - **Word** means 32 bits (four bytes)
- ◆ Most ARM's implement two instruction sets
 - 32-bit ARM Instruction Set
 - 16-bit Thumb Instruction Set
- ◆ Can be configured to view words stored in memory as either Big-endian or Little-endian format

41

ARM Programmer's Model

- ◆ What is a mode?
 - Characterized by specific behavior, privileges, associated registers
 - Triggered by some action (e.g., exception, interrupt)
- ◆ The ARM has seven basic operating modes:
 - **User** : normal program execution mode
 - **FIQ** : used for handling a high priority (fast) interrupt
 - **IRQ** : used for handling a low priority (normal) interrupt
 - **Supervisor** : entered on reset and when a Software Interrupt instruction is executed
 - **Abort** : used for handling memory access violations
 - **Undefined** : used for handling undefined instructions
 - **System** : a privileged mode that uses the same registers as the user mode

42

ARM's Register Set

- ◆ ARM has a total of 37 registers all of which are 32-bits long
 - 30 general purpose registers
 - 1 dedicated program counter (pc)
 - 1 dedicated current program status register (cpsr)
 - 5 dedicated saved program status registers (spsr)
- ◆ In any mode, only a subset of the 37 registers are visible
 - The hidden registers are called **banked registers**
 - The current processor mode governs which registers are accessible

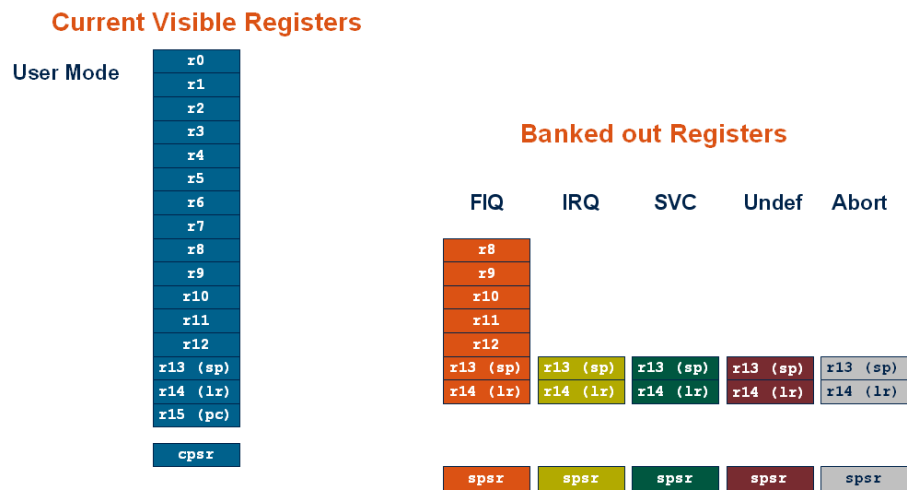
43

ARM's Register Set

- ◆ Each mode can access
 - A particular set of r0–r12 registers
 - A particular r13 (the stack pointer, sp) and r14 (the link register, lr)
 - The program counter, r15 (pc)
 - The current program status register, cpsr
- ◆ Privileged modes (except System) can also access
 - A particular saved program status register (spsr)

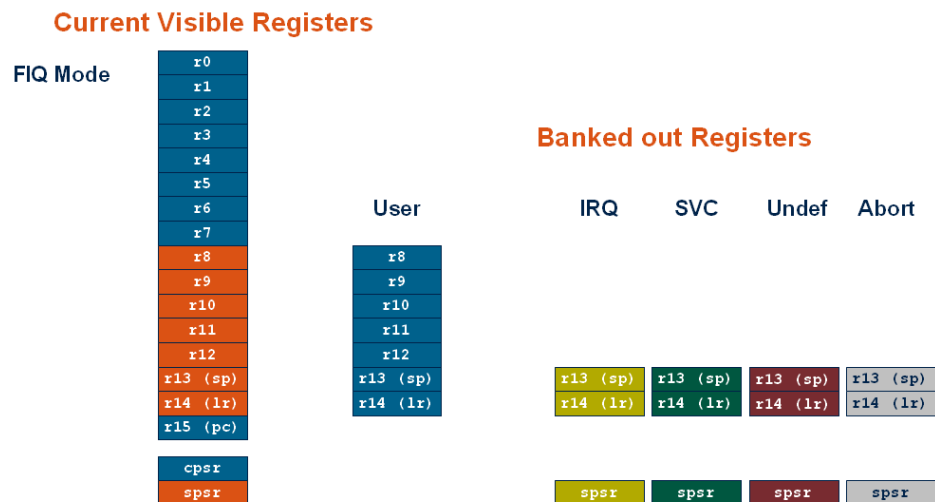
44

ARM Register Set (User Mode)



45

ARM Register Set (FIQ Mode)



46


Banked Registers

- ◆ Banking of registers implies
 - The specific register depends not only on the number (r0, r1, r2 ... r15) but also on the processor mode
- ◆ The values stored in banked registers are preserved across mode changes
- ◆ Example – assume that the processor is executing in user mode
 - In user mode, assume that the processor writes 0 in r0 and 8 in r8
 - Processor changes to FIQ mode
 - In FIQ mode, the value of r0 is
 - If processor overwrites both r0 and r8 with 1 in FIQ mode and changes back to user mode
 - The new value stored in r0 (in user mode) is
 - The new value stored in r8 (in user mode) is

47

ARM's Register Organization

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8-fiq	R8	R8	R8	R8
R9	R9-fiq	R9	R9	R9	R9
R10	R10-fiq	R10	R10	R10	R10
R11	R11-fiq	R11	R11	R11	R11
R12	R12-fiq	R12	R12	R12	R12
R13	R13-fiq	R13-svc	R13-abt	R13-irq	R13-und
R14	R14-fiq	R14-svc	R14-abt	R14-irq	R14-und
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR-fiq	SPSR-svc	SPSR-abt	SPSR-irq	SPSR-und

 = banked register
 SPSR = State Program Status Register

48

Review of Today' s Lecture

- ◆ Overview of ARM' s architecture from a programmer' s point of view
 - Processor modes
 - General purpose registers
- ◆ Coming up
 - Special purpose registers
 - Exception handling
 - ARM' s instruction set