20. Wrap-Up

18-349: Embedded Real-Time Systems

Priya Narasimhan
Electrical & Computer Engineering
Carnegie Mellon University

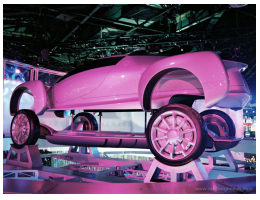
http://www.ece.cmu.edu/~ee349

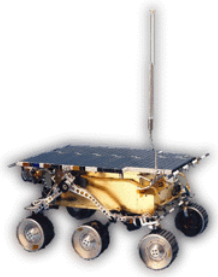
Carnegie Mellon

Electrical & Computer ENGINEERING

Lecture: Introduction to Embedded Systems

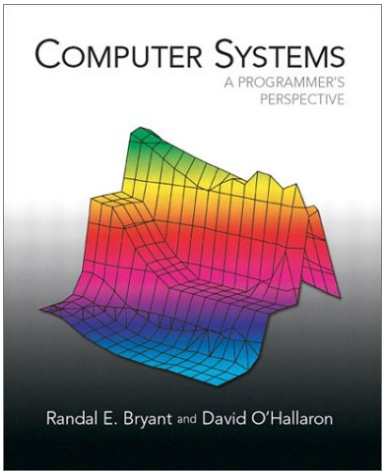
- What is an embedded system?
  - More than just a computer -- it's a system
- What makes embedded systems different?
  - Many sets of constraints on designs
  - Four general types: General-Purpose, Control, Signal Processing, Communications
- What embedded system designers need to know ?
  - **Multi-objective**: cost, dependability, performance, etc.
  - **Multi-discipline**: hardware, software, electromechanical, etc.
  - **Life cycle**: specification, design, prototyping, deployment, support, retirement





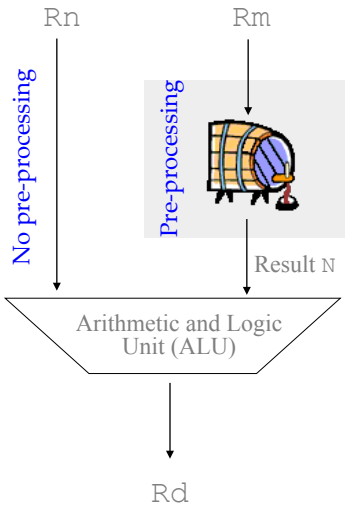
Lecture: Overview of Fundamentals

- Instructions, machine code
- Instruction cycle, fetch cycle
- Basics of interrupts
- The memory pyramid
- Caches
- Pipelining
- Big endian vs. little endian
- Memory areas of a program
- Moore's Law
- RISC vs. CISC



Lecture: ARM Architecture

- The ARM Programmer's Model
- Load-Store Architecture
- Barrel Shifter
- Conditional processing of instructions
- RISC features
- Non-RISC features



Lecture: ARM Assembly

- ARM instructions can be broadly classified as
  - Data Processing Instructions: manipulate data within the registers
  - Branch Instructions: changes the flow of instructions or call a subroutine
  - Load-Store Instructions: transfer data between registers and memory
  - Software Interrupt Instruction: causes a software interrupt
  - Program Status Instructions: read/write the processor status registers

- All instructions can access `r0-r14` directly
- Most instructions also allow use of the `pc`
- Specific instructions to allow access to `cpsr` and `spsr`

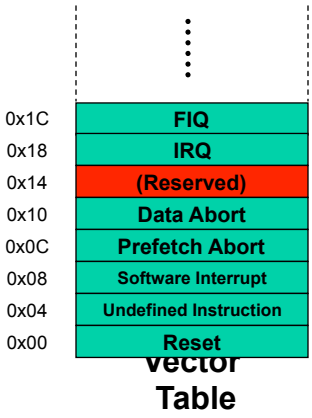
- You might find it useful to bring the additional handout that explains how instructions are encoded

Lecture: Stacks, ATPCS, LDM, STM

- ARM-Thumb Procedure Call Standard (ATPCS)
  - Specifies how routines are called and how registers are allocated
- ARM uses load-store-multiple instructions to accomplish stack operations
- Pop (removing data from a stack) uses load-multiple
- Push (placing data on a stack) uses store-multiple
- Stacks according to ATPCS
  - Full descending
- What does this mean for you?
  - Use `STMFD` to store registers on stack at procedure entry
  - Use `LDMFD` to restore registers from stack at procedure exit
- What do these handy aliases actually represent?
  - `STMFD` = `STMDB` (store-multiple-decrement-before)
  - `LDMFD` = `LDMIA` (load-multiple-increment-after)

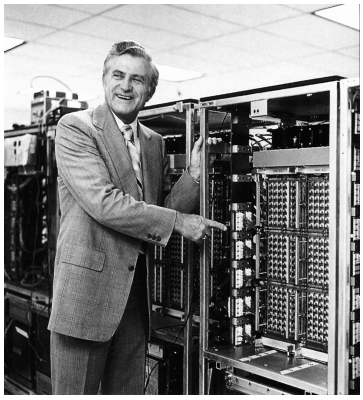
Lecture: Interrupt/Exception Handling

- When an exception occurs, the ARM:
  - Copies `cpsr` into `spsr_<mode>`
  - Sets appropriate `cpsr` bits
    - Change to ARM state
    - 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>`



Lecture: Code Optimization

- Improving program performance
- Standard compiler optimizations
  - common sub-expression elimination
  - dead-code elimination
  - induction variables
- Aggressive compiler optimizations
  - in-lining of functions
  - loop unrolling
- Architectural code optimizations

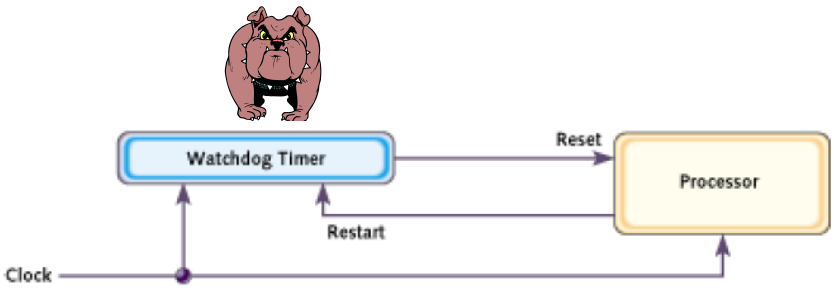


Lecture: SWIs, System Calls

- Software Interrupts (SWIs)
  - What is a SWI?
  - What happens on an SWI?
  - Vectoring SWIs
  - What happens on SWI completion?
  - What do SWIs do?
- Loading Software Interrupt Vectors
- Hardware interrupts
  - interrupt handlers
  - nested interrupts
  - interrupt timing and metrics
  - installing and writing interrupt handlers

Lecture: Timers

- Watchdog: Piece of hardware that can be used to reset the processor in case of anomalies
- Typically a timer that counts to zero
  - Reboots the system if counter reaches zero
  - For normal operation – the software has to ensure that the counter never reaches zero ("kicking the dog")



Source: Introduction to Watchdog Timers, M. Barr, Embedded.com

Lecture: Memory-Mapped I/O

- I/O Registers are NOT like normal memory
  - Device events can change their values (e.g., status registers)
  - Reading a register can change its value (e.g., error condition reset)
    - For example, can't expect to get same value if read twice
  - Some are read-only (e.g., receive registers)
  - Some are write-only (e.g., transmit registers)
  - Sometimes multiple I/O registers are mapped to same address
    - Selection of one based on other info (e.g., read vs. write or extra control bits)

- When polling I/O registers, should tell compiler that value can change on its own and therefore should not be stored in a register
  - `volatile int *ptr;` (or `int volatile *ptr;`)
- A variable should be declared volatile if its value could change unexpectedly
  - Memory-mapped I/O registers
  - Global variables that can be modified by an interrupt service routine
  - Global variables within multi-threaded applications

Lecture: Serial Communications

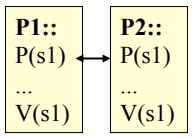
- **Serial Communications**
  - Data communications and modulation
  - Asynchronous protocols
  - Serial port and bit transmission
  - Serial I/O from device drivers
  - Parity

Lecture: Concurrency

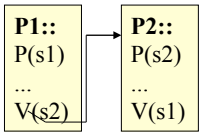
- **Critical sections**
  - the atomicity requirement
- Techniques to enforce critical sections
  - Solution 1: too hard-wired
  - Solution 2: does not work!
  - Solution 3: deadlock!
  - Solution 4: starvation!
  - Solution 5: Dekker's algorithm
  - **Semaphores**: good…
- Implementing Semaphores
  - test-and-set instructions
  - ARM instructions

Lecture: Synchronization and Deadlocks

- Types of Synchronization
- **Deadlocks**
  - conditions for deadlocks
  - dealing with deadlocks
- Some **classical synchronization problems**
  - producer/consumer problems
  - reader/writer problems



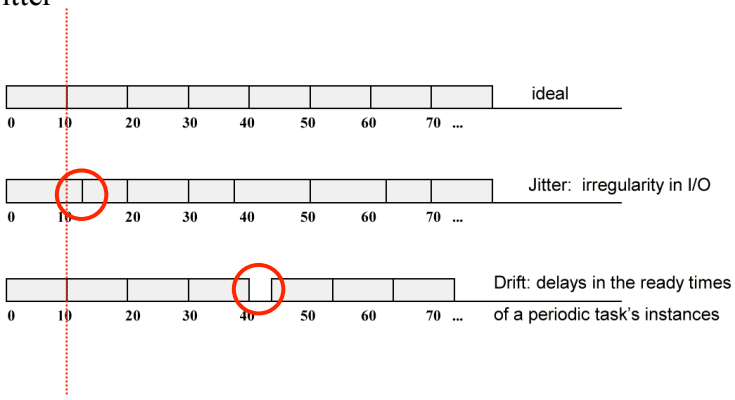
Mutex



Barrier Synchronization

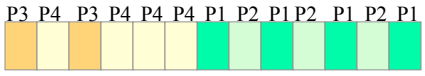
Lectures: Real-Time Basics

- Hard real-time, soft real-time
- How to analyze the schedulability of independent periodic tasks.
- Dealing with transient overload
- Handling context-switching overhead
- Drift, jitter



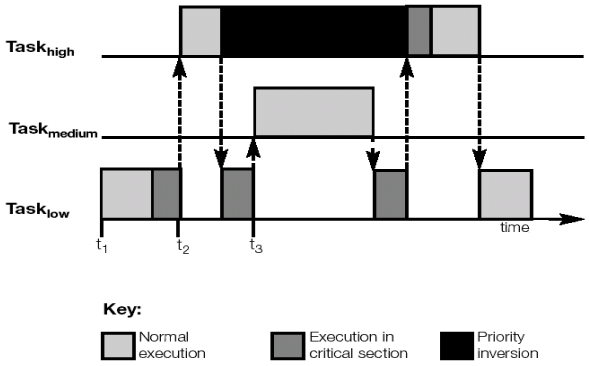
Lecture: Scheduling Algorithms

- What is the scheduler?
  - Part of the operating system that decides which process/task to run next
  - Uses a scheduling algorithm that enforces some kind of policy that is designed to meet some criteria
  - Round-robin, first-come-first-served, earliest-deadline-first, rate-monotonic, ….
- Criteria may vary
  - *CPU utilization* - keep the CPU as busy as possible
  - *Throughput* - maximize the number of processes completed per time unit
  - *Turnaround time* - minimize a process' latency (run time), i.e., time between task submission and termination
  - *Response time* - minimize the wait time for interactive processes
  - *Real-time* - must meet specific deadlines to prevent "bad things" from happening



Lecture: Priority Inversion

- Synchronization in real-time systems
  - Priority inversion
  - Unbounded priority inversion
  - Protocols to bound priority inversion
    - Basic priority inheritance protocol
    - Priority ceiling protocol



Lecture: Real-Time Theory

- Utilization bound (UB) test: a set of *n* independent periodic tasks scheduled by the rate monotonic algorithm will always meet its deadlines, for all task phasings, if

$$\frac{C_1}{T_1} + \ldots + \frac{C_n}{T_n} \leq U(n) = n(2^{1/n} - 1)$$

U(1) = 1.0	U(4) = 0.756	U(7) = 0.728
U(2) = 0.828	U(5) = 0.743	U(8) = 0.724
U(3) = 0.779	U(6) = 0.734	U(9) = 0.720

- As the number of tasks goes to infinity, the bound approaches ln(2) = 0.693
  - Thus, any number of independent periodic tasks will meet their deadlines if the total system utilization is under 69%

Today's Lecture: 18549 Preview

- Course Organization
  - NO lab exercises (you've already had these in 18-348 or 18-349)
  - NO mid-term exam
  - NO final exam
  - Assignments: Design reviews for other teams' projects
  - Project – team-based and focus of the course
- Think of a field-demo scenario that you will be able to show me
  - What would you put into a five-minute video clip to pitch your solution?
- Even better if you can think of a "Before" and "After" scenario
  - How do people do things without your solution now?
- Important to have done a competitive analysis
  - What's out there that competes with your solution?
  - Why is your solution any better or different?
- Talk to me NOW!

Topics Covered

- What are embedded systems?
- CISC vs. RISC
- ARM architecture & assembly
- System Calls (SWIs)
- Program Loading
- Code Optimization
- Interrupt Handling
- Timers, Watchdogs
- Serial Communication
- Memory-Mapped I/O

- Critical Sections
- Basic Concurrency
- Processes and CPU Scheduling
- Mutual Exclusion
- Deadlock
- Semaphores

- Periodic Tasks, RMA
- Real-Time Resource Management
- Scheduling algorithms
- Task Synchronization
- UB test & RT test
- Priority Inversion

We covered a wide spectrum.

Final Exam

- Schedule for the final exam
  - Friday, Dec 14th, 830am-1130am
  - All clarifications will be posted on the screen during the exam
  - Open book, open notes
- Recitation for the final exam
  - Monday, Dec 10th, 7-9pm
  - First time we've actually done three recitations for the course
    - One for Quiz 1, one for Quiz 3, one for the final exam
- Bring your calculators (no laptops will be allowed)
  - Please turn off cell-phones to minimize distraction on your fellow students
- Please make sure to revise

And Finally ….

- The best news of all
  - The course is almost over!
- Your feedback matters
- Please let us know what you liked and what you did not
  - I would appreciate a note so that we know how to improve