Embedded Systems

Ch 8
ARM Architecture

Byung Kook Kim
Dept of EECS
Korea Advanced Institute of Science and Technology

Overview

- 1. The Reduced Instruction Set Computer (RISC)
- 2. The ARM Architecture
- 3. The ACORN RISC Machine
- 4. Architectural Inheritance
- 5. The ARM Programmer's Model
- 6. ARM Development Tools

1. The Reduced Instruction Set Computer

- Patterson and Ditzel, 1980
  - The case for the reduced instruction computer
    - Optimal architecture for a single-chip processor need not be the same as the optimal architecture for a multi-chip processor.

- Berkeley processor design project
  - Berkeley RISC I
    - Incorporated a Reduced Instruction Set Computer (RISC) architecture
    - Much simpler than CISC processors
    - Order of magnitude less design effort to develop, but similar performance

The Reduced Instruction Set Computer (II)

- RISC architecture
  - A fixed (32-bit) instruction size with few formats
    - CISC processors typically had variable length instruction sets with many formats
  - A *load-store architecture* where instructions that process data operate only on registers and are separate from instructions that access memory
    - CISC processors typically allowed values in memory to be used as operands in data processing instructions
  - A large register bank of thirty-two 32-bit registers, all of which could be used for any purpose, to allow the load-store architecture to operate efficiently
    - CISC register sets were getting larger, but none was this large and most had different registers for different purposes (Ex: data and address registers in Motorola 68000)

The Reduced Instruction Set Computer (III)

- RISC organization
  - Hard-wired instruction decode logic
    - CISC processors used large microcode ROMs to decode their instructions
  - Pipelined execution
    - CISC processors allowed little, if any, overlap between consecutive instructions (though they do now)
  - Single-cycle execution
    - CISC processors typically took many clock cycles to complete a single instruction.

The Reduced Instruction Set Computer (IV)

- RISC advantages
  - A smaller die size
  - A shorter development time
  - A higher performance

- RISC drawbacks
  - RISCs generally have poor code density compared with CISCs
    - Due to fixed-length instruction
    - Increasing the cache miss rate
  - RISCs don't execute x86 code
    - For IBM-PC compatible

The Reduced Instruction Set Computer (V)

- ARM processor
  - Based on RISC principles
  - Less poor code density than most other RISCs
  - **Thumb** architecture
    - 16-bit compressed form of the original 32-bit ARM
    - Dynamic decompression hardware in the instruction pipeline
    - Code density better than most CISC processors
- Beyond RISC
  - No development visible at the time of writing
  - Better instruction set
    - Efficient implementation
    - Multimedia support

2. The ARM Architecture

- **ARM processor**
  - Developed at Acorn Computers Limited of Cambridge, England
    - 1983-1985
  - The first RISC microprocessor developed for commercial use
  - ARM Limited established in 1990
    - Licensed to many semiconductor manufacturers
  - Market leader for row-power and cost-sensitive embedded applications
  - Supported by a toolkit
    - Instruction set emulator for hardware modeling, software testing, and benchmarking
    - Assembler, C and C++ compilers, linker, and a symbolic debugger

3. The Acorn RISC Machine

- ARM processor
  - Developed at Acorn Computers Limited, Cambridge, England
    - 1983-1985
    - ARM: Acorn RISC Machine
  - BBC (British Broadcasting Corporation) micro
    - UK personal computer with 8-bit 6502 microprocessor, 1982.
  - Design of a proprietary microprocessor
    - Hundreds of man-years to develop
  - Berkeley RISC I
    - Designed by a few postgraduate students under one year
    - Competitive with the leading commercial offerings
  - ARM was born through a serendipitous combination of factors
  - Acorn expanded to Advanced RISC Machine.

4. Architectural Inheritance

- Features used (from Berkeley RISC design)
  - A load-store architecture
  - Fixed-length 32-bit instructions
  - 3-address instruction formats.

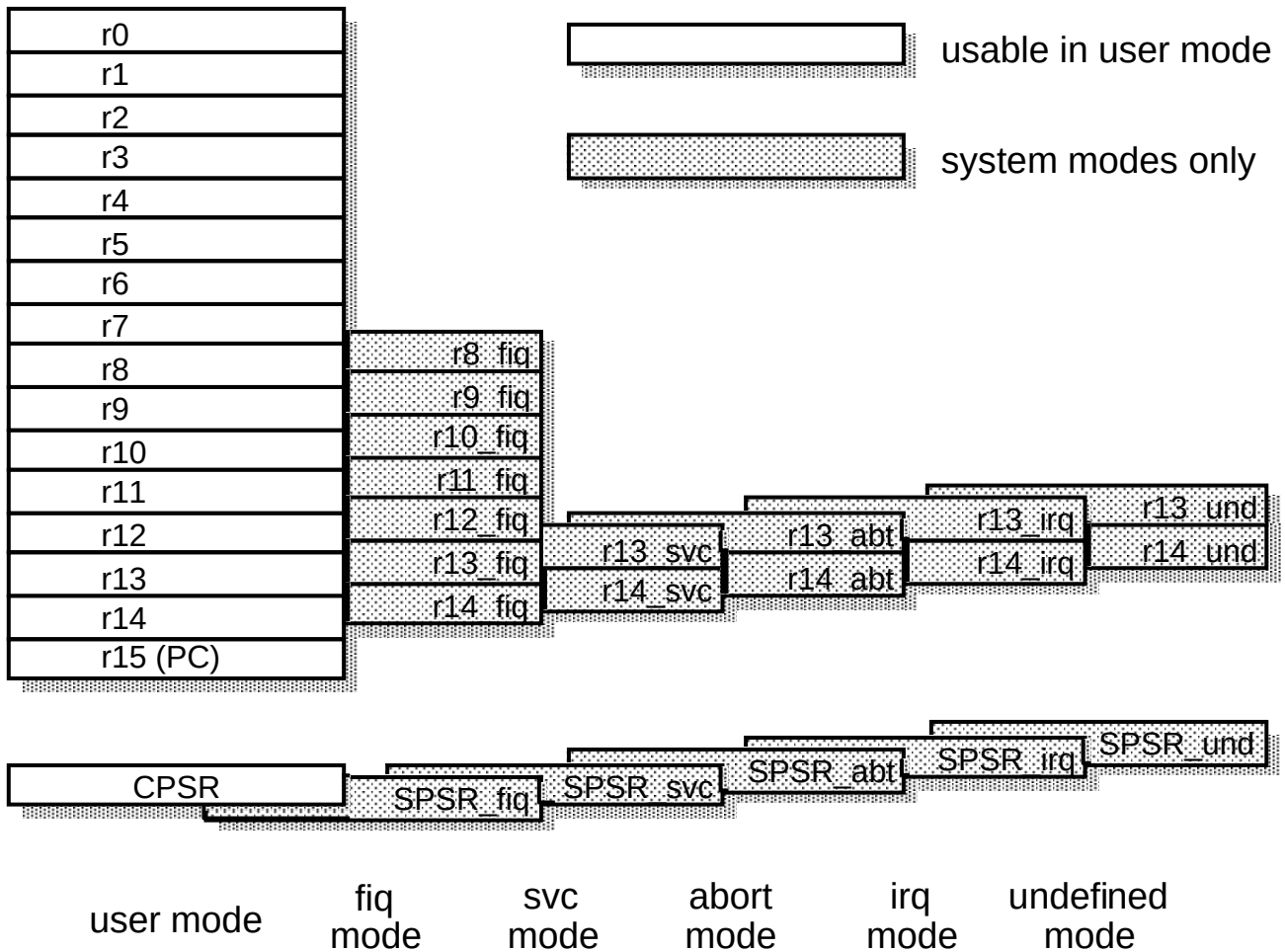
- Features rejected
  - Register windows
    - 32 visible register window out of large register banks
    - Large chip area for registers: Rejected on cost grounds
  - Delayed branches
    - Branch takes effect *after* the following instructions has executed
    - Remove atomicity of individual instructions
    - More complex exception handling
  - Single-cycle execution of all instructions
    - Possible for separate data and instruction memories

Architectural Inheritance (II)

- Simplicity
  - Keep the design simple
    - Full-custom CMOS design
    - Minimize risks which are under your control
  - Simplicity of ARM
    - Instruction set architecture
    - Hardware organization
  - Retain a few CISC features
    - Significantly better code density than a pure RISC
  - Result
    - Power-efficiency
    - Small core size

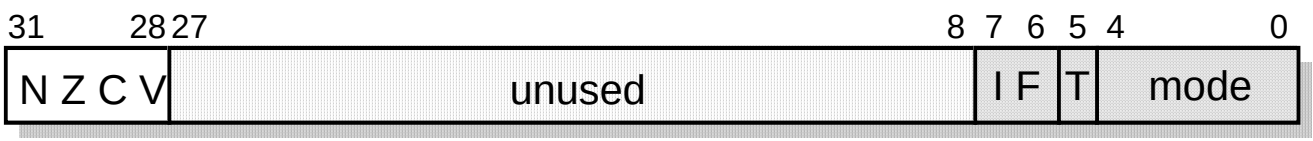
5. The ARM Programmer's Model

- Visible registers in an ARM processor



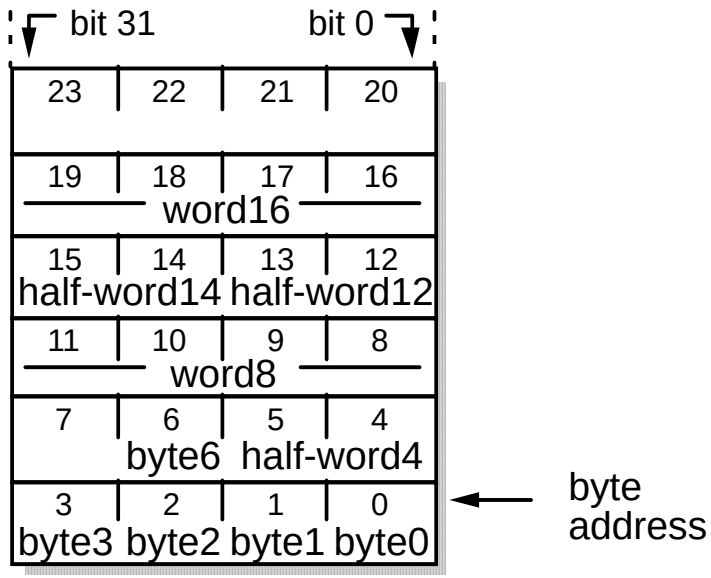
The ARM Programmer's Model (II)

- User-level program
  - 15 general-purpose registers (r0 to r14)
  - Program counter (r15)
  - Current program status register (CPSR)
  - Remaining registers: only for system–level programming and handling exceptions
- CPSR
  - Stores the condition code bits
    - N: Negative. Bit 31
    - Z: Zero. Bit 30
    - C: Carry. Bit 29
    - V: oVerflow. Bit 28



The ARM Programmer's Model (III)

- Memory system
  - Linear array of bytes from 0 to 2^{32} -1
  - Data items: b-bits, 16-bits, 32-bits
  - Words are always aligned on 4-byte boundaries
  - Half-words are aligned on even byte boundaries
  - Memory organization
    - Little endian ->
    - Big endian.



The ARM Programmer's Model (IV)

- Load-store architecture
  - Instruction set processes values in registers, and always place the results into a register.
  - Memory access instructions:
    - Copy memory into registers (Load instruction)
    - Copy register values into memory (Store instruction)
  - No memory-to-memory operations such as
    - Mem + reg -> mem
  - ARM instruction categories
    - Data processing instructions
    - Data transfer instructions: Load, store
    - Control flow instructions: Branch, branch and link, trap into system code (system call)

The ARM Programmer's Model (V)

- Supervisor mode (protected)
  - Ensure that user code cannot gain supervisor privileges
  - System-level functions can only be accessed through specified supervisor calls
    - Access hardware peripheral registers
  - User-level program
    - Algorithms to operate on the data owned by their programs
  - Operating system
    - Handle all transactions with the world outside their programs

The ARM Programmer's Model (VI)

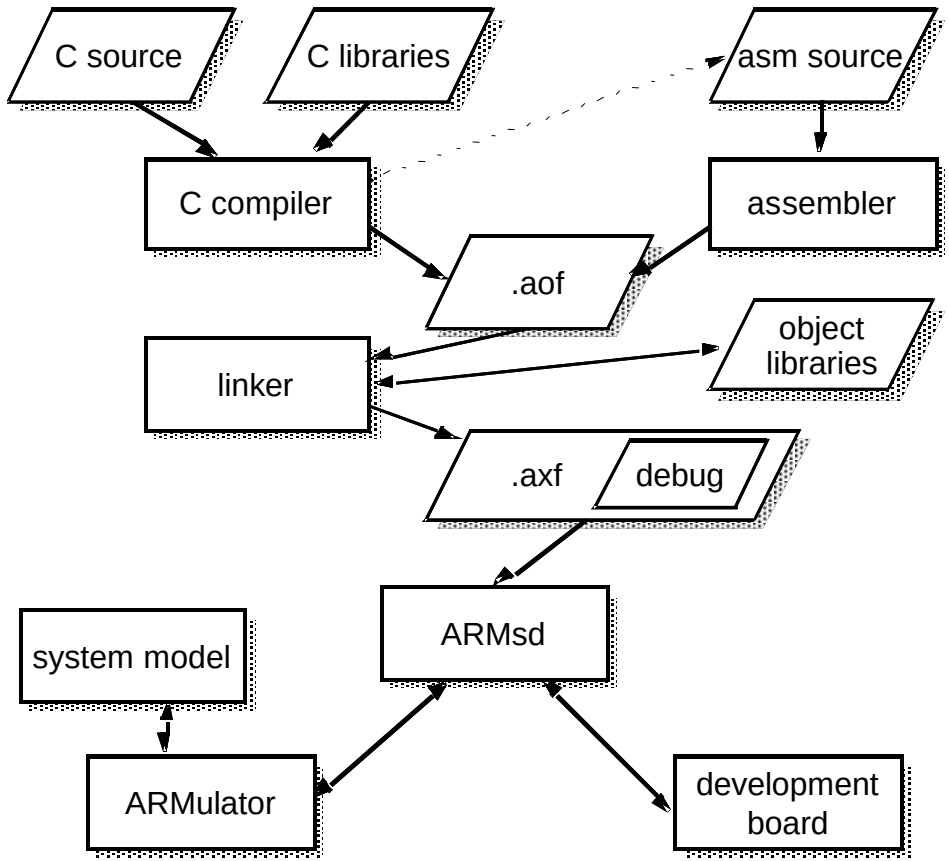
- Features of the ARM instruction set
  - The load-store architecture
  - 3-address data processing instructions
    - 2 source and destination registers
  - Conditional execution of every instruction
  - Inclusion of very powerful load and store multiple register instructions
  - Ability to perform a general shift operation and a general ALU operation in a single instruction that executes in a single clock cycle
  - Open instruction set extension through the coprocessor instruction set, including adding new registers and data types to the programmer's model
  - A very dense 16-bit compressed representation of the instruction set in the Thumb architecture

The ARM Programmer's Model (VI)

- ARM exceptions
  - Supports a range of interrupts, traps, and supervisor calls
  - General handling
    - The current state is saved by copying the PC into r14_exc and the CSPR into SPSR_exc (exc: exception type)
    - The processor operating mode is changed to the appropriate exception mode
    - The PC is force to a value between 00h and 1Ch, the particular value depending on the type of exception (vector addresses).
  - R13_exc
    - Initialized to point to a dedicated stack in memory
    - TO save some user registers for use as work registers
  - Return to user program
    - Restore user registers
    - Instruction to restore the PC and the CSPR.

6. ARM Development Tools

- Software development tools
  - ARM Limited
  - Third party
  - Public domain tools
    - ARM Gcc compiler
- Cross-development
  - Run on a different architecture – PC Windows, Unix workstation
  - C & assembler: .aof files ->
  - Linker: .aif file
  - Symbolic debugger: ARMsd
    - ARM development board
    - ARMulator: software emulation



ARM Development Tools (II)

- ARM C compiler
  - Compliant with the ANSI (American National Standards Institute) standard for C
  - Supported by the appropriate library of standard functions
  - Uses the ARM Procedure Call Standard for all externally available functions
  - Can produce assembly source output instead of ARM object format
    - Inspection, optimization
  - Can produce Thumb code
- ARM assembler
  - Full macro assembler
  - Produces ARM object format output
  - Near machine-level

ARM Development Tools (III)

- The linker
  - Takes one or more object files
  - Combines them into an executable program
  - Resolves symbolic references between the object files
  - Extracts object modules from libraries as needed by the program
  - Output
    - Code to run in RAM or ROM
    - Debug tables
      - Full symbolic debug tables
    - Object library modules
      - Not executable
      - Ready for efficient linking with object files in the future

ARM Development Tools (IV)

- ARMsd (ARM symbolic debugger)
  - A front-end interface to assist in debugging programs running
    - Under emulation (on the ARMulator) or
    - Remotely on a target system (such as ARM development board)
      - Remote debug protocol
      - Via serial line, ethernet, or through JTAG test interface
  - Functions
    - Setting breakpoints
      - Cause execution to halt
      - Processor state can be examined
    - Full source level debugging
      - Debug a program using C source file to specify breakpoints
      - Use variable names from the original program

ARM Development Tools (V)

- ARMulator (ARM emulator)
  - A suite of programs that models the behavior of various ARM processor cores
  - Levels of accuracy
    - Instruction-accurate modeling: without regard to precise timing
    - Cycle-accurate modeling: exact behavior on a cycle-by-cycle basis
    - Timing-accurate modeling: Signals at the correct time within a cycle, allowing logic delays to be accounted for.
  - Run considerably slower than the real hardware
  - Test ARM program without actual hardware
  - Complete, timing-accurate, C model of the target system
  - Core of a timing-accurate ARM behavioral model in a hardware simulation environment such as VHDL.

ARM Development Tools (VI)

- ARM Software Development Toolkit
  - Complete set of tools as above
  - Utility programs and documentation
  - CD-ROM with a PC version & Sun or HP Unix version
    - Full Window-based project manager
  - ARM Project Manager
    - A graphical front-end for the tools
    - Supports building a library or executable image from
      - Source files (C, assembler, and so on)
      - Object files
      - Library files
    - Build options
      - Output optimization: for code size or execution time
      - Debug or release form
      - Target ARM processor.

References

- ARM architecture
  - Steve Furber, “ARM System-on-Chip Architecture”, Addison Wesley, 2000.