Arm Architecture

Enrique Secanechia-Santos, Kevin Mesolella
Outline

• History
• What is ARM?
• What uses ARM?
• Instruction Set
  – Registers
  – ARM specific instructions/implementations
  – Stack
  – Interrupts
• Pipeline
ARM History: Acorn

• 1979: Hermann Hauser and Chris Curry, with the support of a group of students and researchers from Cambridge University, set up Acorn Computers to make personal computers in Cambridge, England

• 1985: Acorn Computer Group develops the world's first commercial RISC processor, called ARM (Acorn RISC Machine, later called Advanced RISC Machine)
  – Fabricated by VLSI Technology and used under 25,000 transistors
ARM History: ARM2

- Developed shortly after ARM1 in 1985
- Featured a true 32-bit data bus, and a 26-bit address bus, with 16 registers and no on-chip cache
- The Multiply and Multiply and Accumulate instructions were added to facilitate digital signal processing
  - Used to generate sounds for home and educational computers
- Only 30,000 transistors, one of the simplest RISK processors
- First ARM2 based system was the ARM Development System.
  - Arm Processor, 3 support chips, 4 MB of RAM and Development tools
- Second ARM2 based system was Acorn’s multimedia PC (1987)
  - 8 MHz version of the ARM2 and three support chips (MEMC, VIDC, and IOC), an input/output controller, and a simple operating system
ARM History: Apple and ARM

- Apple entering PDA market with Newton
- Apple wants to use ARM processor but wanted to work with a separate company for competitive reasons.
- Joint venture was negotiated between Apple, VLSI Technology, and Acorn
- Advanced RISC Machines Ltd. (ARM) launched on November 27th, 1990
ARM History

- **1991:** ARM got its first real commercial break when it licensed its products to GEC-Plessey Semiconductor in the UK.
- **1993:** Licensing deal with Texas Instruments (TI).
  - Gave ARM credibility and Proved the successful viability of the company’s novel licensing business model.
  - The deal drove ARM to formalize their licensing business model
  - Drove them to make more cost-effective products.
- **1998:** ARM Holdings floats on the London Stock Exchange and the Nasdaq.
- **1999:** ARM becomes a member of the FTSE 100.
- **2001:** ARM has 76.8% share of the 32-bit embedded RISC chip market.
- **2007:** About 98% of the more than 1 billion mobile phones sold each year use at least one ARM processor.
## ARM History: Major CPUs

<table>
<thead>
<tr>
<th>Example ARM component</th>
<th>Architecture Generation</th>
<th>Example Application</th>
<th>Approximate date of introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM1</td>
<td>ARMv1</td>
<td>Acorn Computer in internal testing</td>
<td>1985</td>
</tr>
<tr>
<td>ARM2</td>
<td>ARMv2</td>
<td>Acorn Archimedes (Macintosh-era PC)</td>
<td>1987</td>
</tr>
<tr>
<td>ARM6</td>
<td>ARMv3</td>
<td>Apple Newton MessagePad 100 series</td>
<td>1994</td>
</tr>
<tr>
<td>ARM7TDMI</td>
<td>ARMv4</td>
<td>Game Boy Advance, Nintendo DS*, iPod</td>
<td>2001</td>
</tr>
<tr>
<td>ARM9E</td>
<td>ARMv5</td>
<td>Nintendo DS*, Nokia N-Gage, Airport Extreme N basestation</td>
<td>2004</td>
</tr>
<tr>
<td>ARM11</td>
<td>ARMv6</td>
<td>iPhone, iPhone 3G, iPod touch</td>
<td>2007</td>
</tr>
<tr>
<td>Cortex-A8</td>
<td>ARMv7</td>
<td>Palm Pre, iPhone 3GS</td>
<td>2009</td>
</tr>
</tbody>
</table>

* *Nintendo DS incorporates both processors for different uses.*
ARM History: Timeline

- 1979: Acorn Established
- 1985: First commercial RISC processor called ARM (Acorn RISC Machine)
- 1990: Apple enters PDA market and plans to use ARM CPUs
- November 1990: ARM Ltd. (Advanced RISC Machines) Founded.
- 1993: Licensing Deal with Texas Instruments
- 2001: ARM holds 76.8% of embedded RISC CPU market.
- 2007: 98% of 1 billion mobile phones sold each year have at least one ARM CPU.
What is ARM?

- ARM Ltd.
- Headquartered in Cambridge, UK
- Designs ARM CPUs. Does not manufacture Silicon chips.
- Licenses Intellectual Property to Silicon manufacturers. (Broadcom, Freescale, NEC, Texas Instruments, etc...)
- ARM CPUs are known for low power consumption and high performance.
## ARM Chip Manufacturers

- AMD
- Samsung
- Broadcom
- Texas Instruments
- Freescale
- Toshiba
- Fujitsu
- STMicroelectronics
- nVIDIA
- eSilicon
- ST-Ericsson
- ZiiLABS
- Open-Silicon
- NXP
- Infineon
- Fuzhou Rocketship Electronics CO.
- Zilog
What Uses ARM?

○ CORTEX-A - ARM Application Processors
  ■ Smartphones
  ■ Feature Phones
  ■ Tablets / eReaders
  ■ Advanced Personal Media Players
  ■ Digital Television
  ■ Set-top Boxes & Satellite Receivers
  ■ High-End Printers
  ■ Personal Navigation Devices
  ■ Server
  ■ Enterprise
What Uses ARM? (Continued)

- CORTEX-R - ARM Embedded Real-time Processors
  - Automotive Control Systems
  - Wireless and Wired Sensor Networks
  - Wireless base station infrastructure
  - Mass Storage Controllers
  - Printers
  - Network Devices
What Uses ARM? (continued)

- CORTEX-M - ARM Embedded Processors
  - Merchant MCUs
  - Automotive Control Systems
  - Motor Control Systems
  - White Goods controllers
  - Smart Meters
  - Sensors
  - Internet of Things
What Uses ARM? (continued)

○ SecurCore - ARM Secure Processors
  ■ SIMs
  ■ Smart Cards
  ■ Advanced Payment Systems
  ■ Electronic Passports
  ■ Electronic Ticketing
  ■ and Transportation
Why Low Power/Efficient

- Most RISC processor vendors were designing large chips, while ARM opted for small-scale processor
- Not enough resources available for ARM to allow for creation of large and complex devices
- Began as a necessity for development team which was talented but inexperienced
  - Programmers and board-level circuit designers
Instruction Set: Registers

- 37 Registers, all 32 bits long.
- 1 Dedicated Program Counter.
- 1 Dedicated Current Program Status Register. (CPSR)
- 5 Dedicated Saved Program Status Registers. (SPSR)
- 30 General Purpose Registers.
Instruction Set: General Registers and Program Counter

### General Registers

<table>
<thead>
<tr>
<th>User32 / System</th>
<th>FIQ32</th>
<th>Supervisor32</th>
<th>Abort32</th>
<th>IRQ32</th>
<th>Undefined32</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>r0</td>
<td>r0</td>
<td>r0</td>
<td>r0</td>
<td>r0</td>
</tr>
<tr>
<td>r1</td>
<td>r1</td>
<td>r1</td>
<td>r1</td>
<td>r1</td>
<td>r1</td>
</tr>
<tr>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td>r2</td>
</tr>
<tr>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
</tr>
<tr>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
</tr>
<tr>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
</tr>
<tr>
<td>r6</td>
<td>r6</td>
<td>r6</td>
<td>r6</td>
<td>r6</td>
<td>r6</td>
</tr>
<tr>
<td>r7</td>
<td>r7</td>
<td>r7</td>
<td>r7</td>
<td>r7</td>
<td>r7</td>
</tr>
<tr>
<td>r8</td>
<td>r8_fiq</td>
<td>r8</td>
<td>r8</td>
<td>r8</td>
<td>r8</td>
</tr>
<tr>
<td>r9</td>
<td>r9_fiq</td>
<td>r9</td>
<td>r9</td>
<td>r9</td>
<td>r9</td>
</tr>
<tr>
<td>r10</td>
<td>r10_fiq</td>
<td>r10</td>
<td>r10</td>
<td>r10</td>
<td>r10</td>
</tr>
<tr>
<td>r11</td>
<td>r11_fiq</td>
<td>r11</td>
<td>r11</td>
<td>r11</td>
<td>r11</td>
</tr>
<tr>
<td>r12</td>
<td>r12_fiq</td>
<td>r12</td>
<td>r12</td>
<td>r12</td>
<td>r12</td>
</tr>
<tr>
<td>r13 (sp)</td>
<td>r13_fiq</td>
<td>r13_svc</td>
<td>r13_abt</td>
<td>r13_irq</td>
<td>r13_undef</td>
</tr>
<tr>
<td>r14 (lr)</td>
<td>r14_fiq</td>
<td>r14_svc</td>
<td>r14_abt</td>
<td>r14_irq</td>
<td>r14_undef</td>
</tr>
<tr>
<td>r15 (pc)</td>
<td>r15 (pc)</td>
<td>r15 (pc)</td>
<td>r15 (pc)</td>
<td>r15 (pc)</td>
<td>r15 (pc)</td>
</tr>
</tbody>
</table>

### Program Status Registers

- cpsr
- spsr_fiq
- spsr_svc
- spsr_abt
- spsr_irq
- spsr_undef
Instruction Set: Status Registers (CPSR and SPSRs):

- N - Negative (Set by ALU)
- Z - Zero (Set by ALU)
- C - ALU Operation Carried Out.
- V - ALU Operation Overflowed.
- I - Interrupt Disable (I = 1 Disables IRQ).
- F - Fast Interrupt Disable (F = 1 Disables FIQ).
- T - Architecture v4T only (0 = ARM state, 1 = Thumb state)
## Instruction Set: Instruction Format

<table>
<thead>
<tr>
<th>Instruction type</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data processing / PSR Transfer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiply</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Long Multiply</td>
<td>(v3M / v4 only)</td>
<td></td>
</tr>
<tr>
<td>Swap</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load/Store Byte/Word</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load/Store Multiple</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Halfword transfer: Immediate offset (v4 only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Halfword transfer: Register offset (v4 only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Branch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Branch Exchange</td>
<td>(v4T only)</td>
<td></td>
</tr>
<tr>
<td>Coprocessor data transfer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coprocessor data operation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coprocessor register transfer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software interrupt</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Instruction Format

<table>
<thead>
<tr>
<th>Cond</th>
<th>Opcode</th>
<th>S</th>
<th>Rn</th>
<th>Rd</th>
<th>Operand2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1</td>
<td>0 0 0 0 0 0 A S</td>
<td>Rd</td>
<td>Rn</td>
<td>Rs</td>
<td>1 0 0 1</td>
</tr>
<tr>
<td>0 0 0 1 U A S</td>
<td>RdHi</td>
<td>RdLo</td>
<td>Rs</td>
<td>1 0 0 1</td>
<td>Rm</td>
</tr>
<tr>
<td>0 0 0 1 0 B 0 0</td>
<td>Rn</td>
<td>Rd</td>
<td>0 0 0 0 1 0 0 1</td>
<td>Rm</td>
<td></td>
</tr>
<tr>
<td>0 1 1 0 U B W L</td>
<td>Rn</td>
<td>Rd</td>
<td>Offset</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0 0 U S W L</td>
<td>Rn</td>
<td>Register List</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 P U 1 W L</td>
<td>Rn</td>
<td>Rd</td>
<td>Offset1</td>
<td>1 S H</td>
<td>1 Offset2</td>
</tr>
<tr>
<td>0 0 0 P U 0 W L</td>
<td>Rn</td>
<td>Rd</td>
<td>0 0 0 0 1 S H</td>
<td>1 Rm</td>
<td></td>
</tr>
<tr>
<td>1 0 1 1</td>
<td></td>
<td></td>
<td>Offset</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 1 0 0 1 0</td>
<td>1 l l l l l l l l</td>
<td>1 l l l</td>
<td>0 0 0 1</td>
<td>Rn</td>
<td></td>
</tr>
<tr>
<td>1 1 0 0 P U N W L</td>
<td>Rn</td>
<td>CRd</td>
<td>CPNum</td>
<td>Offset</td>
<td></td>
</tr>
<tr>
<td>1 1 1 0</td>
<td>Op1</td>
<td>CRn</td>
<td>CRd</td>
<td>CPNum</td>
<td>Op2</td>
</tr>
<tr>
<td>1 1 1 0</td>
<td>Op1</td>
<td>L</td>
<td>CRn</td>
<td>Rd</td>
<td>CPNum</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td></td>
<td></td>
<td>SWI Number</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Instruction Set: Conditional Field

- All instructions contain a condition field which determines whether the CPU will execute them.
- Uses status register flags set in previous instructions. (N, Z, etc...)
- Removes the need for many branches, which stall the pipeline
  - Takes 3 cycles to refill pipeline
  - Non executed instructions using conditional field only take up one instruction
    - Still have to complete cycle for fetching and decoding
## Instruction Set: Conditional Field

<table>
<thead>
<tr>
<th>Cond</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>EQ - Equal (Z set)</td>
</tr>
<tr>
<td>0001</td>
<td>NE - Not Equal (Z clear)</td>
</tr>
<tr>
<td>0010</td>
<td>HS/CS - Unsigned Higher or Same (C set)</td>
</tr>
<tr>
<td>0011</td>
<td>LO/CC - Unsigned Lower (C clear)</td>
</tr>
<tr>
<td>0100</td>
<td>MI - Negative (N set)</td>
</tr>
<tr>
<td>0101</td>
<td>PL - Positive (P set)</td>
</tr>
<tr>
<td>0110</td>
<td>VS - Overflow (V set)</td>
</tr>
<tr>
<td>0111</td>
<td>VC - No overflow (V clear)</td>
</tr>
<tr>
<td>1000</td>
<td>HI - Unsigned Higher (C set &amp; Z clear)</td>
</tr>
<tr>
<td>1001</td>
<td>LS - Unsigned lower or same (C clear or Z)</td>
</tr>
<tr>
<td>1010</td>
<td>GE - &gt; or = (N &amp; V set or N &amp; V clear)</td>
</tr>
<tr>
<td>1011</td>
<td>LT - &gt; (N set V clear or N &amp; V clear)</td>
</tr>
<tr>
<td>1100</td>
<td>GT - &gt; (Z clear &amp; either N set V set or N clear V set)</td>
</tr>
<tr>
<td>1101</td>
<td>LE - &lt; or = (Z set, or N set and V clear, or N clear V set)</td>
</tr>
<tr>
<td>1110</td>
<td>AL - always execute</td>
</tr>
<tr>
<td>1111</td>
<td>NV - reserved</td>
</tr>
</tbody>
</table>
Instruction Set: Conditional Field Example

Normal Execution:
ADD r0, r1, r2 ; r0 = r1 + r2
or
ADDL r0, r1, r2

Execute only if zero flag is set:
ADDEQ r0, r1, r2 ; If zero flag is set

; r0 = r1 + r2
Instruction Set: Data Processing Instructions

- Arithmetic operations, comparisons, logical operations, data movement between registers.
- Load/Store architecture
  - Instructions only work on registers, not memory
- Each perform specific operation on one or two operands
  - First operand always a register
  - Second operand sent to ALU through the barrel shifter.
Instruction Set: Barrel Shifter

- No actual shift instructions. ARM uses a barrel shifter instead.
- Shifts can be applied to operands of other instructions.
  - e.g. r0 = r1 * 5
  - ADD r0, r1, r1, LSL #2 ;r0 = r1 + (r1 * 4)
- LSL - Logical Shift Left
- LSR - Logical Shift Right
- ASR - Arithmetic Shift Right (Preserves Sign Bit)
- ROR - Rotate Right (Last bit rotated is also used as Carry Out.
- ROX - Rotate Right Through Carry. (C is 33rd bit)
Instruction Set: Pre and Post-indexed Addressing

Pre-indexed Addressing

* Example: STR r0, [r1,#12]
Instruction Set: Pre and Post-indexed Addressing

Post-indexed Addressing

* Example: STR r0, [r1], #12

![Diagram showing example of STR instruction with pre-indexed and post-indexed addressing.]
In most architectures, including ARM, a stack grows down in memory
  - Last value “pushed” to lowest address

ARM supports ascending stacks
  - Stack structure grows up through memory
  - Last value “pushed” to highest address

As in most architectures, ARM uses stacks for subroutines.
Instruction Set: Software Interrupts

- SWI Instruction causes software interrupt.
- Causes exception trap to SWI hardware vector.
- Causes a change to Supervisor mode.
- The exception handler can examine the “comment” field and decide what to do.
- Used by an Operating System to implement a set of privileged operations which can be requested from User Mode.
Instruction Set: Coprocessors

- ARM architecture supports 16 coprocessors
- Non-intrusive way of extending the instruction set.
- Each coprocessor instruction set occupies part of the ARM instruction set
- 3 types of coprocessor instructions:
  - Coprocessor data processing
  - Coprocessor (to/from ARM) Registers
  - Coprocessor memory transfers (load/store)
- Coprocessors can be implemented in hardware, software or both.
- Integrate hardware using coprocessor mechanism
  - Allows for specialized operations to support specific application/use
Pipelining: Basic 3 Stage Pipeline

- All stages can be broken down into Fetch, Decode and Execute.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>FETCH</td>
</tr>
<tr>
<td>PC - 4</td>
<td>DECODE</td>
</tr>
<tr>
<td>PC - 8</td>
<td>EXECUTE</td>
</tr>
</tbody>
</table>

- Instruction fetched from memory
- Decoding of registers used in instruction
- Register(s) read from Register Bank, Shift and ALU operation, Write register(s) back to Register Bank
Pipelining: ARM1156:T2-S
Pipelining: ARM11 MPCore
Conclusion

- ARM Ltd. has become a popular embedded CPU designer.
- ARM CPUs are well known for their low power usage and efficiency.
- The ARM instruction set has many features for efficient use of hardware, such as the barrel shifter.