The ARM Architecture
Agenda

- Introduction to ARM Ltd
  ARM Architecture/Programmers Model
  Data Path and Pipelines
  AMBA
  Development Tools
ARM Ltd

- Founded in November 1990
  - Spun out of Acorn Computers

- Designs the ARM range of RISC processor cores
- Licenses ARM core designs to semiconductor partners who fabricate and sell to their customers.
  - ARM does not fabricate silicon 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
ARM Partnership Model
ARM Powered Products
Agenda

- Introduction to ARM Ltd
  - ARM Architecture/Programmers Model
- Data Path and Pipelines
- AMBA
- Development Tools
Architecture Revisions

XScale is a trademark of Intel Corporation
Data Sizes and Instruction Sets

- The ARM is a 32-bit 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

- Jazelle cores can also execute Java bytecode
Processor Modes

- The ARM has seven basic operating modes:
  - **User**: unprivileged mode under which most tasks run
  - **FIQ**: entered when a high priority (fast) interrupt is raised
  - **IRQ**: entered when a low priority (normal) interrupt is raised
  - **Supervisor**: entered on reset and when a Software Interrupt instruction is executed
  - **Abort**: used to handle memory access violations
  - **Undef**: used to handle undefined instructions
  - **System**: privileged mode using the same registers as user mode
The ARM Register Set

Current Visible Registers

Abort Mode
- r0
- r1
- r2
- r3
- r4
- r5
- r6
- r7
- r8
- r9
- r10
- r11
- r12
- r13 (sp)
- r14 (lr)
- r15 (pc)
- cpsr
- spsr

Banked out Registers

User
- r8
- r9
- r10
- r11
- r12
- r13 (sp)
- r14 (lr)
- cpsr
- spsr

FIQ
- r13 (sp)
- r14 (lr)
- spsr

IRQ
- r13 (sp)
- r14 (lr)
- spsr

SVC
- r13 (sp)
- r14 (lr)
- spsr

Undef
- r13 (sp)
- r14 (lr)
- spsr

The ARM Register Set
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>

This can only be done in ARM state.

Vector Table
Vector table can be at 0xFFFF0000 on ARM720T
and on ARM9/10 family devices

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIQ</td>
<td>0x1C</td>
</tr>
<tr>
<td>IRQ</td>
<td>0x18</td>
</tr>
<tr>
<td>(Reserved)</td>
<td>0x14</td>
</tr>
<tr>
<td>Data Abort</td>
<td>0x10</td>
</tr>
<tr>
<td>Prefetch Abort</td>
<td>0x0C</td>
</tr>
<tr>
<td>Software Interrupt</td>
<td>0x08</td>
</tr>
<tr>
<td>Undefined Instruction</td>
<td>0x04</td>
</tr>
<tr>
<td>Reset</td>
<td>0x00</td>
</tr>
</tbody>
</table>
Conditional Execution and Flags

- ARM instructions can be made to execute conditionally by postfixing them with the appropriate condition code field.
  - This improves code density and performance by reducing the number of forward branch instructions.

  ```
  CMP   r3,#0                           CMP   r3,#0
  BEQ   skip                            ADDNE  r0,r1,r2
  ADD    r0,r1,r2                        ADD    r0,r1,r2
  skip
  ```

- By default, data processing instructions do not affect the condition code flags but the flags can be optionally set by using “S”. CMP does not need “S”.

  ```
  loop
  ...
  SUBS  r1,r1,#1                        decrement r1 and set flags
  BNE   loop                             if Z flag clear then branch
  ```
Branch instructions

- Branch: \( B\{<\text{cond}>\} \) label
- Branch with Link: \( BL\{<\text{cond}>\} \) subroutine_label

- The processor core shifts the offset field left by 2 positions, sign-extends it and adds it to the PC
  - \( \pm 32 \) Mbyte range
  - How to perform longer branches?
Data processing Instructions

- Consist of:
  - Arithmetic: ADD, ADC, SUB, SBC, RSB, RSC
  - Logical: AND, ORR, EOR, BIC
  - Comparisons: CMP, CMN, TST, TEQ
  - Data movement: MOV, MVN

- These instructions only work on registers, NOT memory.

- Syntax:

  `<Operation>{<cond>}{S} Rd, Rn, Operand2`

  - Comparisons set flags only - they do not specify Rd
  - Data movement does not specify Rn

- Second operand is sent to the ALU via barrel shifter.
Using a Barrel Shifter: The 2nd Operand

Register, optionally with shift operation
- Shift value can be either be:
  - 5 bit unsigned integer
  - Specified in bottom byte of another register.
- Used for multiplication by constant

Immediate value
- 8 bit number, with a range of 0-255.
  - Rotated right through even number of positions
- Allows increased range of 32-bit constants to be loaded directly into registers
Loading 32 bit constants

- To allow larger constants to be loaded, the assembler offers a pseudo-instruction:
  - `LDR rd, =const`

- This will either:
  - Produce a `MOV` or `MVN` instruction to generate the value (if possible).
  - Generate a `LDR` instruction with a PC-relative address to read the constant from a literal pool (Constant data area embedded in the code).

- For example
  - `LDR r0,=0xFF` => `MOV r0,#0xFF`
  - `LDR r0,=0x55555555` => `LDR r0,[PC,#Imm12]`
    - ...
    - ...
    - `DCD 0x55555555`

- This is the recommended way of loading constants into a register.
Single register data transfer

- **LDR**  **STR**  Word
- **LDRB**  **STRB**  Byte
- **LDRH**  **STRH**  Halfword
- **LDRSB**  Signed byte load
- **LDRSH**  Signed halfword load

- Memory system must support all access sizes

- Syntax:
  - **LDR**{<cond>{<size>}} Rd, <address>
  - **STR**{<cond>{<size>}} Rd, <address>

  e.g. **LDREQB**
Agenda

Introduction to ARM Ltd
ARM Architecture/Programmers Model
- Data Path and Pipelines
  AMBA
Development Tools
The ARM7TDM Core
Pipeline changes for ARM9TDMI

**ARM7TDMI**

- **FETCH**
  - Instruction Fetch

- **DECODE**
  - Thumb→ARM decompress
  - ARM decode
  - Reg Select

- **EXECUTE**
  - Reg Read
  - Shift
  - ALU
  - Reg Write

**ARM9TDMI**

- **FETCH**
  - Instruction Fetch

- **DECODE**
  - ARM or Thumb Inst Decode
    - Reg Decode
    - Reg Read

- **EXECUTE**
  - Shift + ALU

- **MEMORY**
  - Memory Access

- **WRITE**
  - Reg Write
ARM10 vs. ARM11 Pipelines

**ARM10**

<table>
<thead>
<tr>
<th>FETCH</th>
<th>ISSUE</th>
<th>DECODE</th>
<th>EXECUTE</th>
<th>MEMORY</th>
<th>WRITE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch Prediction</td>
<td>ARM or Thumb Instruction Decode</td>
<td>Reg Read</td>
<td>Shift + ALU</td>
<td>Memory Access</td>
<td>Reg Write</td>
</tr>
<tr>
<td>Instruction Fetch</td>
<td></td>
<td></td>
<td>Multiply</td>
<td>Multiply Add</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reg Read</td>
<td></td>
<td>Add</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fetch 1</td>
<td>Fetch 2</td>
<td>Decode</td>
<td>Issue</td>
<td>Write back</td>
<td></td>
</tr>
<tr>
<td>Fetch 1</td>
<td>Fetch 2</td>
<td>Decode</td>
<td>Issue</td>
<td>Write back</td>
<td></td>
</tr>
</tbody>
</table>

**ARM11**

- Shift
- ALU
- Saturate
- MAC 1
- MAC 2
- MAC 3
- Write back
- Address
- Data Cache 1
- Data Cache 2
Agenda

Introduction to ARM Ltd
ARM Architecture/Programmers Model
Data Path and Pipelines

- AMBA
  Development Tools
AMBA

- **AMBA**
  - Advanced Microcontroller Bus Architecture

- **ADK**
  - Complete AMBA Design Kit

- **ACT**
  - AMBA Compliance Testbench
  - PrimeCell
    - ARM’s AMBA compliant peripherals
AHB basic signal timing
Agenda

- Introduction to ARM Ltd
- ARM Architecture/Programmers Model
- Data Path and Pipelines
- AMBA
  - Development Tools
ARM Debug Architecture

- EmbeddedICE Logic
  - Provides breakpoints and processor/system access
- JTAG interface (ICE)
  - Converts debugger commands to JTAG signals
- Embedded trace Macrocell (ETM)
  - Compresses real-time instruction and data access trace
  - Contains ICE features (trigger & filter logic)
- Trace port analyzer (TPA)
  - Captures trace in a deep buffer

Debugger (+ optional trace tools)

ETM

EmbeddedICE Logic

TAP controller

ETM

Trace Port

JTAG port

Ethernet
Keil Development Tools for ARM

- Includes ARM macro assembler, compilers (ARM RealView C/C++ Compiler, Keil CARM Compiler, or GNU compiler), ARM linker, Keil uVision Debugger and Keil uVision IDE
- Keil uVision Debugger accurately simulates on-chip peripherals (I²C, CAN, UART, SPI, Interrupts, I/O Ports, A/D and D/A converters, PWM, etc.)
- Evaluation Limitations
  - 16K byte object code limitation
  - Some linker restrictions such as base addresses for code//constants
  - GNU tools provided are not restricted in any way
- http://www.keil.com/demo/
Keil Development Tools for ARM