ARM

Unit 5:
The ARM Architecture

Introduction

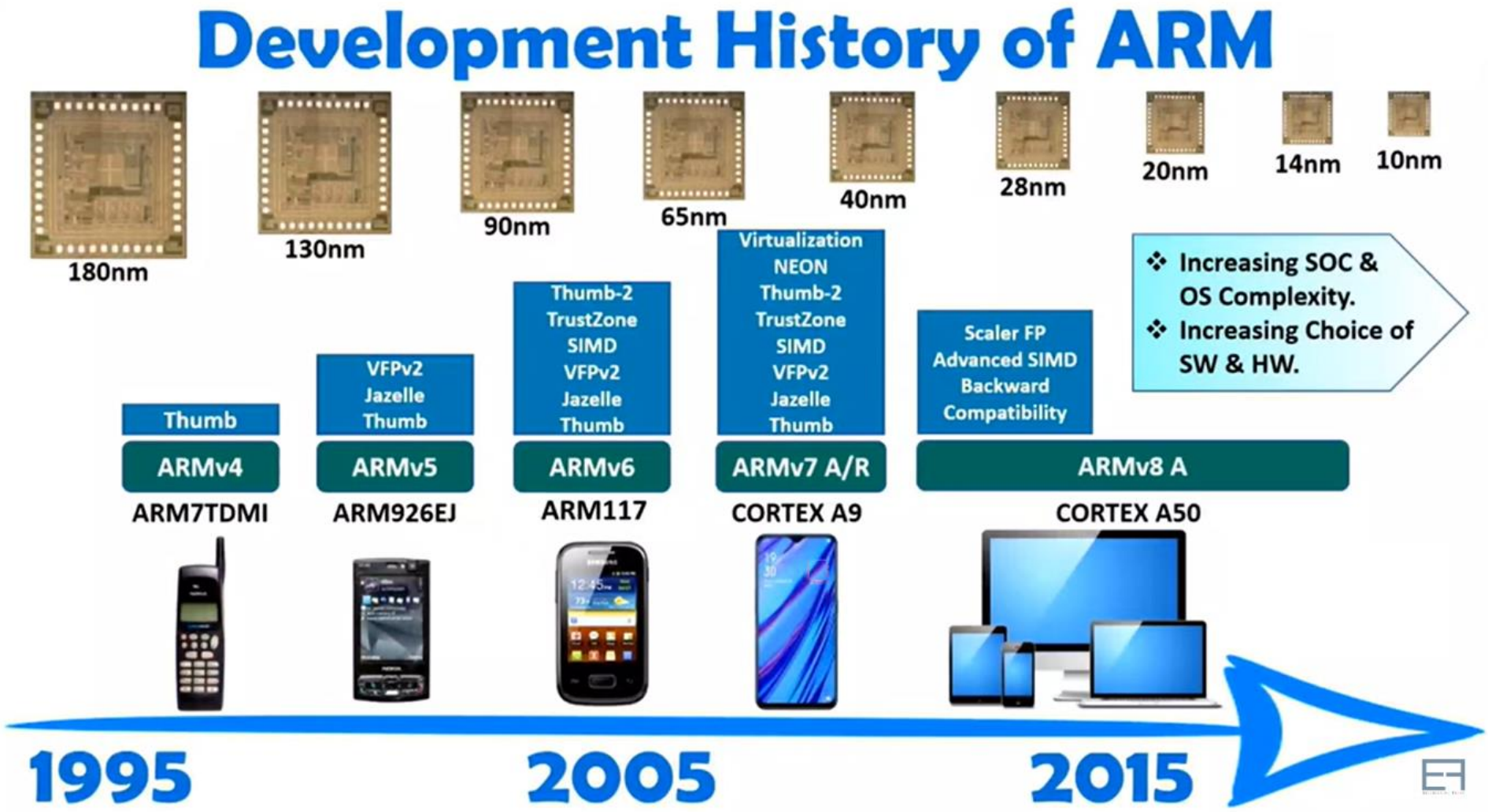
- **Introduction to ARM Ltd**
- Comparison between ARM Controllers
- Features of ARM Controller
- Architecture of ARM controller
- CPSR

Introduction

- **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


Development History of ARM
180nm
130nm
90nm
65nm
40nm
28nm
20nm
14nm
10nm
Thumb
ARMv4
ARM7TDMI
VFPv2
Jazelle
Thumb
ARMv5
ARM926EJ
Thumb-2
TrustZone
SIMD
VFPv2
Jazelle
Thumb
ARMv6
ARM117
Virtualization
NEON
Thumb-2
TrustZone
SIMD
VFPv2
Jazelle
Thumb
ARMv7 A/R
CORTEX A9
Scaler FP
Advanced SIMD
Backward
Compatibility
ARMv8 A
CORTEX A50
Increasing SOC & OS Complexity.
Increasing Choice of SW & HW.
1995
2005
2015

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**

Comparison of ARM Cortex A, M & R CPU

Parameters	Cortex A	Cortex R	Cortex M

Comparison of ARM Cortex A, M & R CPU

Parameters	Cortex A	Cortex R	Cortex M
Performance	❖ Highest	❖ Very Good	❖ Medium

Comparison of ARM Cortex A, M & R CPU

Parameters	Cortex A	Cortex R	Cortex M
Performance	❖ Highest	❖ Very Good	❖ Medium
Response Time	❖ Very Good	❖ Best	❖ Medium
Power Consumption	❖ 80µW/MHz	❖ 120µW/MHz	❖ 8µW/MHz

Comparison of ARM Cortex A, M & R CPU

Parameters	Cortex A	Cortex R	Cortex M
Performance	❖ Highest	❖ Very Good	❖ Medium
Response Time	❖ Very Good	❖ Best	❖ Medium
Power Consumption	❖ 80µW/MHz	❖ 120µW/MHz	❖ 8µW/MHz
Processor	❖ Application Based	❖ RTOS based	❖ Embedded System based

Comparison of ARM Cortex A, M & R CPU

Parameters	Cortex A	Cortex R	Cortex M
Performance	❖ Highest	❖ Very Good	❖ Medium
Response Time	❖ Very Good	❖ Best	❖ Medium
Power Consumption	❖ 80µW/MHz	❖ 120µW/MHz	❖ 8µW/MHz
Processor	❖ Application Based	❖ RTOS based	❖ Embedded System based
Pipeline	❖ Long Pipeline	❖ Medium Pipeline	❖ Short Pipeline
Clock	❖ High	❖ High	❖ Less relatively

Comparison of ARM Cortex A, M & R CPU

Parameters	Cortex A	Cortex R	Cortex M
Performance	❖ Highest	❖ Very Good	❖ Medium
Response Time	❖ Very Good	❖ Best	❖ Medium
Power Consumption	❖ 80µW/MHz	❖ 120µW/MHz	❖ 8µW/MHz
Processor	❖ Application Based	❖ RTOS based	❖ Embedded System based
Pipeline	❖ Long Pipeline	❖ Medium Pipeline	❖ Short Pipeline
Clock	❖ High	❖ High	❖ Less relatively
Memory	❖ Cache Memory with more size.	❖ Cache Memory + Tightly Coupled Memory	❖ Cache Memory with less size.

Comparison of ARM Cortex A, M & R CPU

Parameters	Cortex A	Cortex R	Cortex M
Performance	❖ Highest	❖ Very Good	❖ Medium
Response Time	❖ Very Good	❖ Best	❖ Medium
Power Consumption	❖ 80µW/MHz	❖ 120µW/MHz	❖ 8µW/MHz
Processor	❖ Application Based	❖ RTOS based	❖ Embedded System based
Pipeline	❖ Long Pipeline	❖ Medium Pipeline	❖ Short Pipeline
Clock	❖ High	❖ High	❖ Less relatively
Memory	❖ Cache Memory with more size.	❖ Cache Memory + Tightly Coupled Memory	❖ Cache Memory with less size.
ISA	❖ ARM	❖ ARM	❖ Thumb

Comparison of ARM Cortex A, M & R CPU

Parameters	Cortex A	Cortex R	Cortex M
Performance	❖ Highest	❖ Very Good	❖ Medium
Response Time	❖ Very Good	❖ Best	❖ Medium
Power Consumption	❖ 80µW/MHz	❖ 120µW/MHz	❖ 8µW/MHz
Processor	❖ Application Based	❖ RTOS based	❖ Embedded System based
Pipeline	❖ Long Pipeline	❖ Medium Pipeline	❖ Short Pipeline
Clock	❖ High	❖ High	❖ Less relatively
Memory	❖ Cache Memory with more size.	❖ Cache Memory + Tightly Coupled Memory	❖ Cache Memory with less size.
ISA	❖ ARM	❖ ARM	❖ Thumb
FPU	❖ Yes	❖ Yes	❖ Optional

Applications

Features of ARM Processor {ARM7TDMI}

- Introduction
- Originally developed by Arcon Computers in mid 1980s then renamed to ARM {Advanced RISC Machine}.

Features of ARM Processor {ARM7TDMI}

- Introduction
- Originally developed by Arcon Computers in mid 1980s then renamed to ARM {Advanced RISC Machine}.
- 'ARM Family' has verities of 32bits controllers used in many commercial applications, Embedded Systems, Smart Phones (Cortex A), Printers (Cortex R), Medical Instruments (Cortex M), etc.
- ARM processor accounts more then 75% of 32bits of processors in market share of industry in 2009.

Features of ARM Processor {ARM7TDMI}

Introduction

- Originally developed by Arcon Computers in mid 1980s then renamed to ARM {Advanced RISC Machine}.
- 'ARM Family' has verities of 32bits controllers used in many commercial applications, Embedded Systems, Smart Phones (Cortex A), Printers (Cortex R), Medical Instruments (Cortex M), etc.
- ARM processor accounts more then 75% of 32bits of processors in market share of industry in 2009.

Features of ARM7

- ARM7 has 32 bits of MCU & ALU.
- ARM7 has 32 bits of Data bus with aligned memory space. {Means with each machine cycle it will execute 32bits via data bus. Here aligned addressing is done.}
- ARM7 has all the instructions with 32 bits.

Features of ARM Processor {ARM7TDMI}

Introduction

- Originally developed by Arcon Computers in mid 1980s then renamed to ARM {Advanced RISC Machine}.
- 'ARM Family' has verities of 32bits controllers used in many commercial applications, Embedded Systems, Smart Phones (Cortex A), Printers (Cortex R), Medical Instruments (Cortex M), etc.
- ARM processor accounts more then 75% of 32bits of processors in market share of industry in 2009.

Features of ARM7

- ARM7 has 32 bits of MCU & ALU.
- ARM7 has 32 bits of Data bus with aligned memory space. {Means with each machine cycle it will execute 32bits via data bus. Here aligned addressing is done.}
- ARM7 has all the instructions with 32 bits.
- ARM7 has 32 bits of Address Bus. {So we can interface 4GB memory with it directly.}
- ARM7 follows Von Neuman Model. {So 4GB memory will be common for code and data.} with latest versions they have switched to Harvard Architecture.

Features of ARM Processor {ARM7TDMI}

Introduction

- Originally developed by Arcon Computers in mid 1980s then renamed to ARM {Advanced RISC Machine}.
- 'ARM Family' has verities of 32bits controllers used in many commercial applications, Embedded Systems, Smart Phones (Cortex A), Printers (Cortex R), Medical Instruments (Cortex M), etc.
- ARM processor accounts more then 75% of 32bits of processors in market share of industry in 2009.

Features of ARM7

- ARM7 has 32 bits of MCU & ALU.
- ARM7 has 32 bits of Data bus with aligned memory space. {Means with each machine cycle it will execute 32bits via data bus. Here aligned addressing is done.}
- ARM7 has all the instructions with 32 bits.
- ARM7 has 32 bits of Address Bus. {So we can interface 4GB memory with it directly.}
- ARM7 follows Von Neuman Model. {So 4GB memory will be common for code and data.} with latest versions they have switched to Harvard Architecture.
- ARM7 has three stage Pipelining
  - Fetch
  - Decode
  - Execute
- ARM7 has 37 registers of 32 bits. At time 16 registers available {R0 – R15}.

Features of ARM Processor {ARM7TDMI}

Introduction

- Originally developed by Arcon Computers in mid 1980s then renamed to ARM {Advanced RISC Machine}.
- 'ARM Family' has verities of 32bits controllers used in many commercial applications, Embedded Systems, Smart Phones (Cortex A), Printers (Cortex R), Medical Instruments (Cortex M), etc.
- ARM processor accounts more then 75% of 32bits of processors in market share of industry in 2009.

Features of ARM7

- ARM7 has 32 bits of MCU & ALU.
- ARM7 has 32 bits of Data bus with aligned memory space. {Means with each machine cycle it will execute 32bits via data bus. Here aligned addressing is done.}
- ARM7 has all the instructions with 32 bits.
- ARM7 has 32 bits of Address Bus. {So we can interface 4GB memory with it directly.}
- ARM7 follows Von Neuman Model. {So 4GB memory will be common for code and data.} with latest versions they have switched to Harvard Architecture.
- ARM7 has three stage Pipelining
  - Fetch
  - Decode
  - Execute
- ARM7 has 37 registers of 32 bits. At time 16 registers available {R0 – R15}.
- ARM7 has LOAD – STORE Architecture. {Means for LOAD and STORE operations we have to use separate instructions.}

Features of ARM Processor {ARM7TDMI}

Introduction

- Originally developed by Arcon Computers in mid 1980s then renamed to ARM {Advanced RISC Machine}.
- 'ARM Family' has verities of 32bits controllers used in many commercial applications, Embedded Systems, Smart Phones (Cortex A), Printers (Cortex R), Medical Instruments (Cortex M), etc.
- ARM processor accounts more then 75% of 32bits of processors in market share of industry in 2009.

Features of ARM7

- ARM7 has 32 bits of MCU & ALU.
- ARM7 has 32 bits of Data bus with aligned memory space. {Means with each machine cycle it will execute 32bits via data bus. Here aligned addressing is done.}
- ARM7 has all the instructions with 32 bits.
- ARM7 has 32 bits of Address Bus. {So we can interface 4GB memory with it directly.}
- ARM7 follows Von Neuman Model. {So 4GB memory will be common for code and data.} with latest versions they have switched to Harvard Architecture.
- ARM7 has three stage Pipelining
  - Fetch
  - Decode
  - Execute
- ARM7 has 37 registers of 32 bits. At time 16 registers available {R0 – R15}.
- ARM7 has LOAD – STORE Architecture. {Means for LOAD and STORE operations we have to use separate instructions.}
- ARM7 has 7 operating Modes.
- ARM7 has 7 interrupts/Exceptions.
- ARM7 has 7 Addressing Modes.

Features of ARM Processor {ARM7TDMI}

Introduction

- Originally developed by Arcon Computers in mid 1980s then renamed to ARM {Advanced RISC Machine}.
- 'ARM Family' has verities of 32bits controllers used in many commercial applications, Embedded Systems, Smart Phones (Cortex A), Printers (Cortex R), Medical Instruments (Cortex M), etc.
- ARM processor accounts more then 75% of 32bits of processors in market share of industry in 2009.

Features of ARM7

- ARM7 has 32 bits of MCU & ALU.
- ARM7 has 32 bits of Data bus with aligned memory space. {Means with each machine cycle it will execute 32bits via data bus. Here aligned addressing is done.}
- ARM7 has all the instructions with 32 bits.
- ARM7 has 32 bits of Address Bus. {So we can interface 4GB memory with it directly.}
- ARM7 follows Von Neuman Model. {So 4GB memory will be common for code and data.} with latest versions they have switched to Harvard Architecture.
- ARM7 has three stage Pipelining
  - Fetch
  - Decode
  - Execute
- ARM7 has 37 registers of 32 bits. At time 16 registers available {R0 – R15}.
- ARM7 has LOAD – STORE Architecture. {Means for LOAD and STORE operations we have to use separate instructions.}
- ARM7 has 7 operating Modes.
- ARM7 has 7 interrupts/Exceptions.
- ARM7 has 7 Addressing Modes.
- ARM7 data formats {8bits – Byte, 16bits – Half word, 32bits - Word}

Features of ARM Processor {ARM7TDMI}

Introduction

- Originally developed by Arcon Computers in mid 1980s then renamed to ARM {Advanced RISC Machine}.
- 'ARM Family' has verities of 32bits controllers used in many commercial applications, Embedded Systems, Smart Phones (Cortex A), Printers (Cortex R), Medical Instruments (Cortex M), etc.
- ARM processor accounts more then 75% of 32bits of processors in market share of industry in 2009.

Features of ARM7

- ARM7 has 32 bits of MCU & ALU.
- ARM7 has 32 bits of Data bus with aligned memory space. {Means with each machine cycle it will execute 32bits via data bus. Here aligned addressing is done.}
- ARM7 has all the instructions with 32 bits.
- ARM7 has 32 bits of Address Bus. {So we can interface 4GB memory with it directly.}
- ARM7 follows Von Neuman Model. {So 4GB memory will be common for code and data.} with latest versions they have switched to Harvard Architecture.
- ARM7 has three stage Pipelining
  - Fetch
  - Decode
  - Execute
- ARM7 has 37 registers of 32 bits. At time 16 registers available {R0 – R15}.
- ARM7 has LOAD – STORE Architecture. {Means for LOAD and STORE operations we have to use separate instructions.}
- ARM7 has 7 operating Modes.
- ARM7 has 7 interrupts/Exceptions.
- ARM7 has 7 Addressing Modes.
- ARM7 data formats {8bits – Byte, 16bits – Half word, 32bits - Word}
- ARM7TDMI
  - T – Thumb Instructions
  - D – Hardware Debugging
  - M – Long Multiply instruction
  - I – ICE Microcells for Breakpoints and watch points in program.

What Is ARM?

- Advanced RISC Machine
  - Founded 1990, owned by Acorn, Apple and VLSI
  - A 32-bit RISC processor
- First RISC microprocessor for commercial use
- Used in portable devices due to low power consumption (cost-sensitive embedded applications)

RISC Vs CISC

RISC	CISC
Reduced Instruction Set Computer.	**Complex Instruction Set Computer**.
It emphasizes on software to optimize the instruction set.	It emphasizes on hardware to optimize the instruction set.
It is a hard-wired unit of programming in the RISC Processor.	Microprogramming unit in CISC Processor.
RISC has simple decoding of instruction.	CISC has complex decoding of instruction.
pipelining is simple	Pipelining is difficult
It uses a less number of instructions.	It uses a large number of instruction
Less time to execute the instructions.	The execution time of CISC is longer.
RISC architecture can be used with high-end applications like telecommunication, image processing, video processing, etc.	CISC architecture can be used with low-end applications like home automation, security system, etc.
It has fixed format instructions.	It has variable format instructions.
The program written for RISC architecture takes more space in memory.	Program written for CISC architecture takes less space in memory.
Example of RISC: **ARM**, AVR, ARC and the SPARC.	Examples of CISC: Motorola 68000 family, System/360, AMD and the Intel x86 CPUs.

Versions

- ARMv1 : First version of ARM processor,
  26-bit addressing, didn't have multiply / coprocessor
- ARMv2: ARM2, **First commercial chip**,
  32-bit result multiply instructions
  Coprocessor support
- ARMv2a : ARM3 chip with on-chip cache,
  Added load and store, cache management
- ARMv3: ARM6, 32 bit addressing,
  Virtual memory support

RISC Design Philosophy

- RISC design initiated mainly for microcontrollers. RISC – Reduced Instruction Set Computer:
  - RISC works as per LOAD STORE MODEL. {For LOAD and STORE operations we need to execute separate instructions. LOAD & STORE can not be done in single instruction. We can not perform logical & Arithmetic operation on the data of memory in single instruction with RISC processor.}

RISC Design Philosophy

- RISC design initiated mainly for microcontrollers. RISC – Reduced Instruction Set Computer:
  - RISC works as per LOAD STORE MODEL. {For LOAD and STORE operations we need to execute separate instructions. LOAD & STORE can not be done in single instruction. We can not perform logical & Arithmetic operation on the data of memory in single instruction with RISC processor.}
  - Most Instructions of RISC are registers based, so performance will be fast.

RISC Design Philosophy

- RISC design initiated mainly for microcontrollers. RISC – Reduced Instruction Set Computer:
  - RISC works as per LOAD STORE MODEL. {For LOAD and STORE operations we need to execute separate instructions. LOAD & STORE can not be done in single instruction. We can not perform logical & Arithmetic operation on the data of memory in single instruction with RISC processor.}
  - Most Instructions of RISC are registers based, so performance will be fast.
  - All instructions are of same size, so parallel execution of instructions will be more efficient.

RISC Design Philosophy

- RISC design initiated mainly for microcontrollers. RISC – Reduced Instruction Set Computer:
  - RISC works as per LOAD STORE MODEL. {For LOAD and STORE operations we need to execute separate instructions. LOAD & STORE can not be done in single instruction. We can not perform logical & Arithmetic operation on the data of memory in single instruction with RISC processor.}
  - Most Instructions of RISC are registers based, so performance will be fast.
  - All instructions are of same size, so parallel execution of instructions will be more efficient.
  - All instructions perform operation on same size of data, so parallel execution of instructions will be more efficient.

RISC Design Philosophy

- RISC design initiated mainly for microcontrollers. RISC – Reduced Instruction Set Computer:
  - RISC works as per LOAD STORE MODEL. {For LOAD and STORE operations we need to execute separate instructions. LOAD & STORE can not be done in single instruction. We can not perform logical & Arithmetic operation on the data of memory in single instruction with RISC processor.}
  - Most Instructions of RISC are registers based, so performance will be fast.
  - All instructions are of same size, so parallel execution of instructions will be more efficient.
  - All instructions perform operation on same size of data, so parallel execution of instructions will be more efficient.
  - Fetch, Decode & Execute time for most instructions is of same standards, so pipeline will be more effective.

RISC Design Philosophy

- RISC design initiated mainly for microcontrollers. RISC – Reduced Instruction Set Computer:
  - RISC works as per LOAD STORE MODEL. {For LOAD and STORE operations we need to execute separate instructions. LOAD & STORE can not be done in single instruction. We can not perform logical & Arithmetic operation on the data of memory in single instruction with RISC processor.}
  - Most Instructions of RISC are registers based, so performance will be fast.
  - All instructions are of same size, so parallel execution of instructions will be more efficient.
  - All instructions perform operation on same size of data, so parallel execution of instructions will be more efficient.
  - Fetch, Decode & Execute time for most instructions is of same standards, so pipeline will be more effective.
  - RISC is having fewer addressing modes as most instructions are based on register, so learning of instruction set simpler.
  - As most instructions are based on registers, RISC has more numbers of registers for programming.

RISC Design Philosophy

- RISC design initiated mainly for microcontrollers. RISC – Reduced Instruction Set Computer:
  - RISC works as per LOAD STORE MODEL. {For LOAD and STORE operations we need to execute separate instructions. LOAD & STORE can not be done in single instruction. We can not perform logical & Arithmetic operation on the data of memory in single instruction with RISC processor.}
  - Most Instructions of RISC are registers based, so performance will be fast.
  - All instructions are of same size, so parallel execution of instructions will be more efficient.
  - All instructions perform operation on same size of data, so parallel execution of instructions will be more efficient.
  - Fetch, Decode & Execute time for most instructions is of same standards, so pipeline will be more effective.
  - RISC is having fewer addressing modes as most instructions are based on register, so learning of instruction set simpler.
  - As most instructions are based on registers, RISC has more numbers of registers for programming.
  - Since most instructions are based on registers, it consumes less power for execution of programs.
  - Most RISC architectures are scalable like ARM, SPARK etc.

RISC Design Philosophy

- RISC design initiated mainly for microcontrollers. RISC – Reduced Instruction Set Computer:
  - RISC works as per LOAD STORE MODEL. {For LOAD and STORE operations we need to execute separate instructions. LOAD & STORE can not be done in single instruction. We can not perform logical & Arithmetic operation on the data of memory in single instruction with RISC processor.}
  - Most Instructions of RISC are registers based, so performance will be fast.
  - All instructions are of same size, so parallel execution of instructions will be more efficient.
  - All instructions perform operation on same size of data, so parallel execution of instructions will be more efficient.
  - Fetch, Decode & Execute time for most instructions is of same standards, so pipeline will be more effective.
  - RISC is having fewer addressing modes as most instructions are based on register, so learning of instruction set simpler.
  - As most instructions are based on registers, RISC has more numbers of registers for programming.
  - Since most instructions are based on registers, it consumes less power for execution of programs.
  - Most RISC architectures are scalable like ARM, SPARK etc.
  - RISC processor uses hardwire control, smaller in size & less area on chip. So many features are provided on chip like Timer, ADC, DAC, IO Ports etc.

ARM Processor-Features

- A load-store architecture
- Fixed-length 32-bit instructions
- 3-address instruction format
  - use each address field to specify either a processor register or a memory operand (ADD R1, R2, R3)
- High code density

ARM Processor-Features

- A load-store architecture
- Fixed-length 32-bit instructions
- 3-address instruction format
  - use each address field to specify either a processor register or a memory operand (ADD R1, R2, R3)
- High code density
- 32-bit architecture till V7.
  - Word/Half word/Byte

Consistent 32-bit processors across all architectures

ARM Architecture

Load Store Architecture

- Only load and store instructions access the memory, all other instructions use registers as operands
  - LDR R0, [R1] ADD R0, R1, R2

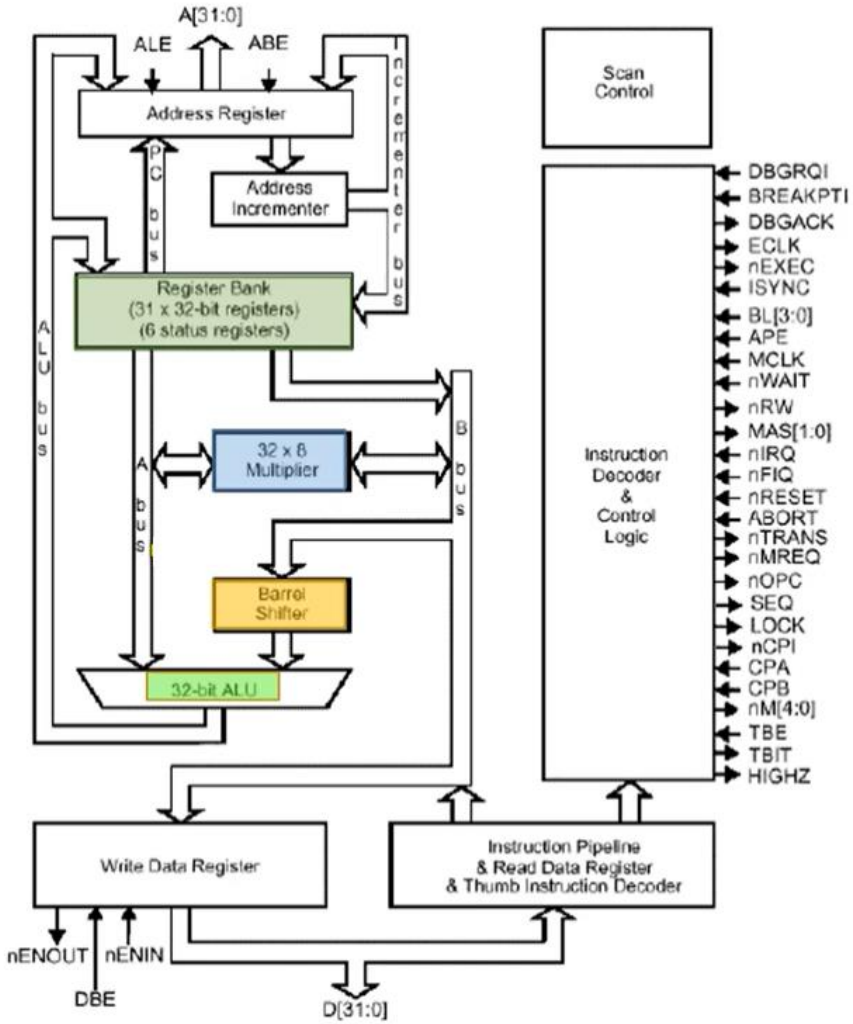
- The instructions will only process values which are in registers and will always place results of such processing into a register.
- The only operations which apply to memory are that **copy memory values into registers** (load instructions) or **copy register values into memory**(store instructions)

To increase speed

ARM Architecture

Architecture overview

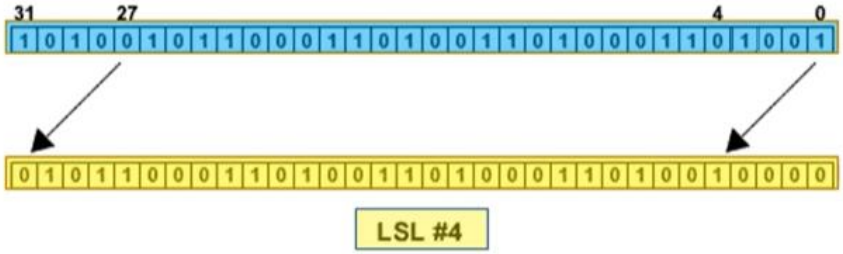
- 32-bit ALU
- Register bank
- Multiplier
- Barrel Shifter
- A-bus and B-bus

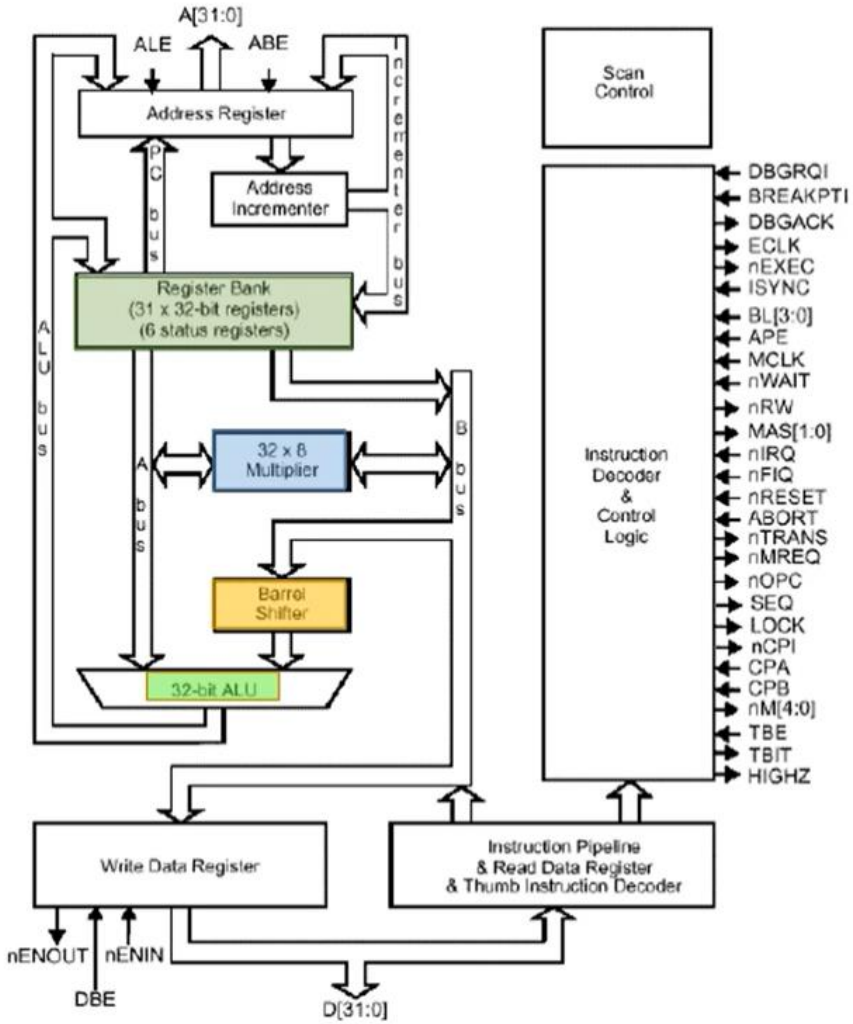


ARM Architecture

Architecture overview

- 32-bit ALU
- Register bank
- Multiplier
- Barrel Shifter
- A-bus and B-bus





The Barrel Shifter

LSL : Logical Left Shift

Multiplication by a power of 2

ASR: Arithmetic Right Shift

Division by a power of 2,
preserving the sign bit

LSR : Logical Shift Right

Division by a power of 2

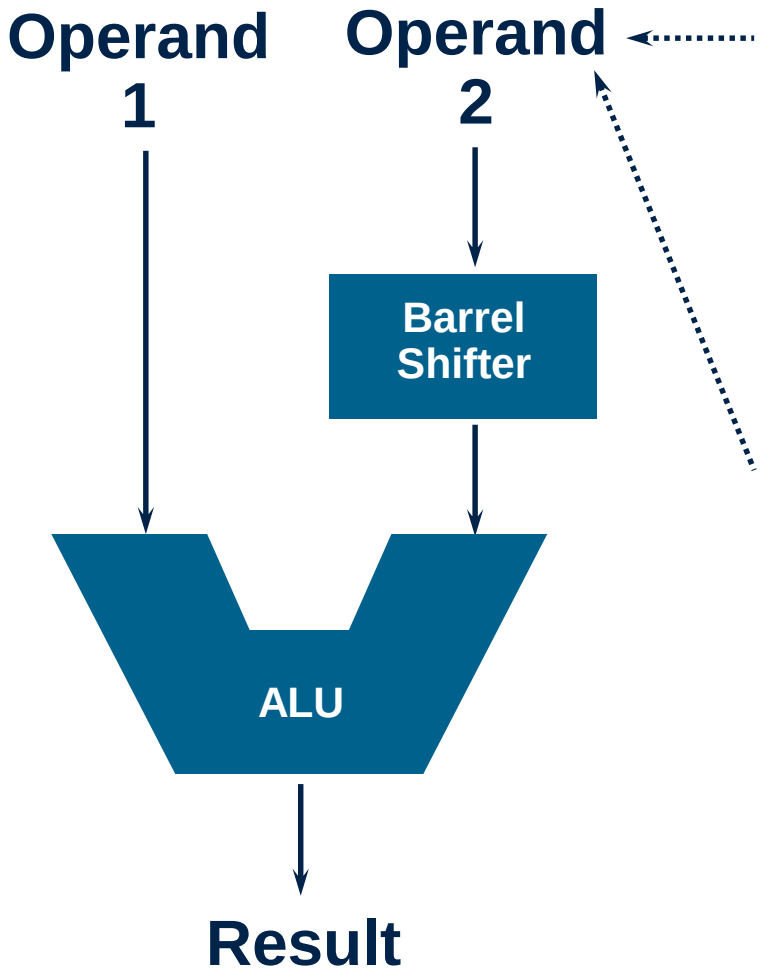
ROR: Rotate Right

Bit rotate with wrap around
from LSB to MSB

RRX: Rotate Right Extended

Single bit rotate with wrap around
from CF to MSB

Using the Barrel Shifter: The Second 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

ARM7 CPSR Flag Register

D31 D30 D29 D28 ..... D7 D6 D5 D4 D3 D2 D1 D0

ARM7 CPSR Flag Register

D31 D30 D29 D28 ..... D7 D6 D5 D4 D3 D2 D1 D0

Mode

Modes of ARM7

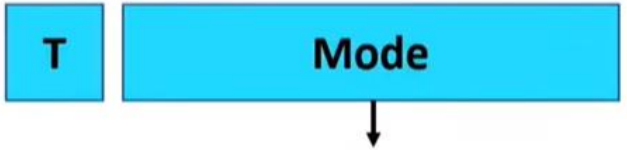
Mode Bits					Mode
1	0	0	0	0	User
1	0	0	0	1	FIQ
1	0	0	1	0	IRQ
1	0	0	1	1	Supervisor
1	0	1	1	1	Abort
1	1	0	1	1	Undefined
1	1	1	1	1	System

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

ARM7 CPSR Flag Register

D31 D30 D29 D28 ..... D7 D6 D5 D4 D3 D2 D1 D0

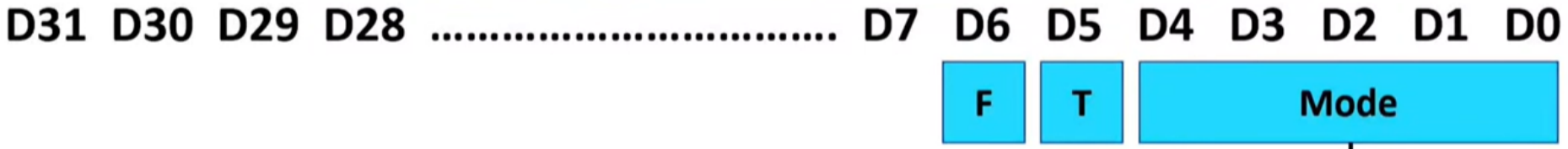


- T – Thumb State
  - 1 = Thumb State
  - 0 = ARM State
- In Thumb State, we have 16bits of instruction, it useful for increasing code density.

Modes of ARM7

Mode Bits					Mode
1	0	0	0	0	User
1	0	0	0	1	FIQ
1	0	0	1	0	IRQ
1	0	0	1	1	Supervisor
1	0	1	1	1	Abort
1	1	0	1	1	Undefined
1	1	1	1	1	System

ARM7 CPSR Flag Register



- **T – Thumb State**
  - 1 = Thumb State
  - 0 = ARM State
- In Thumb State, we have 16bits of instruction, it useful for increasing code density.
- **F – Fast Interrupt Mask**
  - 1 = Fast Interrupts Masked
  - 0 = Fast Interrupts Unmasked
- Fast Interrupts is given on nFIQ Pin.

Modes of ARM7

Mode Bits					Mode
1	0	0	0	0	User
1	0	0	0	1	FIQ
1	0	0	1	0	IRQ
1	0	0	1	1	Supervisor
1	0	1	1	1	Abort
1	1	0	1	1	Undefined
1	1	1	1	1	System

ARM7 CPSR Flag Register

- ❖ T – Thumb State
  - 1 = Thumb State
  - 0 = ARM State
- ❑ In Thumb State, we have 16bits of instruction, it useful for increasing code density.
- ❖ F – Fast Interrupt Mask
  - 1 = Fast Interrupts Masked
  - 0 = Fast Interrupts Unmasked
- ❑ Fast Interrupts is given on nFIQ Pin.
- ❖ I – Interrupt Mask
  - 1 = Interrupts Masked
  - 0 = Interrupts Unmasked
- ❑ Normal Interrupts is given on nIRQ Pin.

Modes of ARM7

Mode Bits					Mode
1	0	0	0	0	User
1	0	0	0	1	FIQ
1	0	0	1	0	IRQ
1	0	0	1	1	Supervisor
1	0	1	1	1	Abort
1	1	0	1	1	Undefined
1	1	1	1	1	System

ARM7 CPSR Flag Register

- **T – Thumb State**
  - 1 = Thumb State
  - 0 = ARM State
- In Thumb State, we have 16bits of instruction, it useful for increasing code density.
- **F – Fast Interrupt Mask**
  - 1 = Fast Interrupts Masked
  - 0 = Fast Interrupts Unmasked
- Fast Interrupts is given on nFIQ Pin.
- **I – Interrupt Mask**
  - 1 = Interrupts Masked
  - 0 = Interrupts Unmasked
- Normal Interrupts is given on nIRQ Pin.

- **V – Overflow Flag**
  - 1 = Signed Overflow
  - 0 = No Signed Overflow
- **C – Carry Flag**
  - 1 = Carry After MSB
  - 0 = No Carry After MSB

Modes of ARM7

Mode Bits					Mode
1	0	0	0	0	User
1	0	0	0	1	FIQ
1	0	0	1	0	IRQ
1	0	0	1	1	Supervisor
1	0	1	1	1	Abort
1	1	0	1	1	Undefined
1	1	1	1	1	System

ARM7 CPSR Flag Register

- ❖ T – Thumb State
  - 1 = Thumb State
  - 0 = ARM State
- ❑ In Thumb State, we have 16bits of instruction, it useful for increasing code density.
- ❖ F – Fast Interrupt Mask
  - 1 = Fast Interrupts Masked
  - 0 = Fast Interrupts Unmasked
- ❑ Fast Interrupts is given on nFIQ Pin.
- ❖ I – Interrupt Mask
  - 1 = Interrupts Masked
  - 0 = Interrupts Unmasked
- ❑ Normal Interrupts is given on nIRQ Pin.

- ❖ V – Overflow Flag
  - 1 = Signed Overflow
  - 0 = No Signed Overflow
- ❖ C – Carry Flag
  - 1 = Carry After MSB
  - 0 = No Carry After MSB
- ❖ Z – Zero Flag
  - 1 = Result is Zero
  - 0 = Result is not Zero
- ❖ N – Negative Flag
  - 1 = Result is Negative
  - 0 = Result is not Negative

Modes of ARM7

Mode Bits					Mode
1	0	0	0	0	User
1	0	0	0	1	FIQ
1	0	0	1	0	IRQ
1	0	0	1	1	Supervisor
1	0	1	1	1	Abort
1	1	0	1	1	Undefined
1	1	1	1	1	System

ARM7 CPSR Flag Register

- **T – Thumb State**
  - 1 = Thumb State
  - 0 = ARM State
- In Thumb State, we have 16bits of instruction, it useful for increasing code density.
- **F – Fast Interrupt Mask**
  - 1 = Fast Interrupts Masked
  - 0 = Fast Interrupts Unmasked
- Fast Interrupts is given on nFIQ Pin.
- **I – Interrupt Mask**
  - 1 = Interrupts Masked
  - 0 = Interrupts Unmasked
- Normal Interrupts is given on nIRQ Pin.

8 bits signed numbers range is -80H to 7FH.

+53H	0101 0011	V = 1
+42H	0100 0010	N = 0
+95H	1001 0101	

- **V – Overflow Flag**
  - 1 = Signed Overflow
  - 0 = No Signed Overflow
- **C – Carry Flag**
  - 1 = Carry After MSB
  - 0 = No Carry After MSB
- **Z – Zero Flag**
  - 1 = Result is Zero
  - 0 = Result is not Zero
- **N – Negative Flag**
  - 1 = Result is Negative
  - 0 = Result is not Negative

Modes of ARM7

Mode Bits					Mode
1	0	0	0	0	User
1	0	0	0	1	FIQ
1	0	0	1	0	IRQ
1	0	0	1	1	Supervisor
1	0	1	1	1	Abort
1	1	0	1	1	Undefined
1	1	1	1	1	System

ARM7 CPSR Flag Register

D31	D30	D29	D28	.....	D7	D6	D5	D4	D3	D2	D1	D0
N	Z	C	V	Not Used in ARM7TDMI		I	F	T	Mode			

- **T – Thumb State**
  - 1 = Thumb State
  - 0 = ARM State
- In Thumb State, we have 16bits of instruction, it useful for increasing code density.
- **F – Fast Interrupt Mask**
  - 1 = Fast Interrupts Masked
  - 0 = Fast Interrupts Unmasked
- Fast Interrupts is given on nFIQ Pin.
- **I – Interrupt Mask**
  - 1 = Interrupts Masked
  - 0 = Interrupts Unmasked
- Normal Interrupts is given on nIRQ Pin.

- **V – Overflow Flag**
  - 1 = Signed Overflow
  - 0 = No Signed Overflow
- **C – Carry Flag**
  - 1 = Carry After MSB
  - 0 = No Carry After MSB
- **Z – Zero Flag**
  - 1 = Result is Zero
  - 0 = Result is not Zero
- **N – Negative Flag**
  - 1 = Result is Negative
  - 0 = Result is not Negative

Modes of ARM7

Mode Bits					Mode
1	0	0	0	0	User
1	0	0	0	1	FIQ
1	0	0	1	0	IRQ
1	0	0	1	1	Supervisor
1	0	1	1	1	Abort
1	1	0	1	1	Undefined
1	1	1	1	1	System

8 bits signed numbers range is -80H to 7FH.

+53H	0101 0011
+42H	0100 0010
+95H	1001 0101

V = 1
N = 0

8 bits signed numbers range is -80H to 7FH.

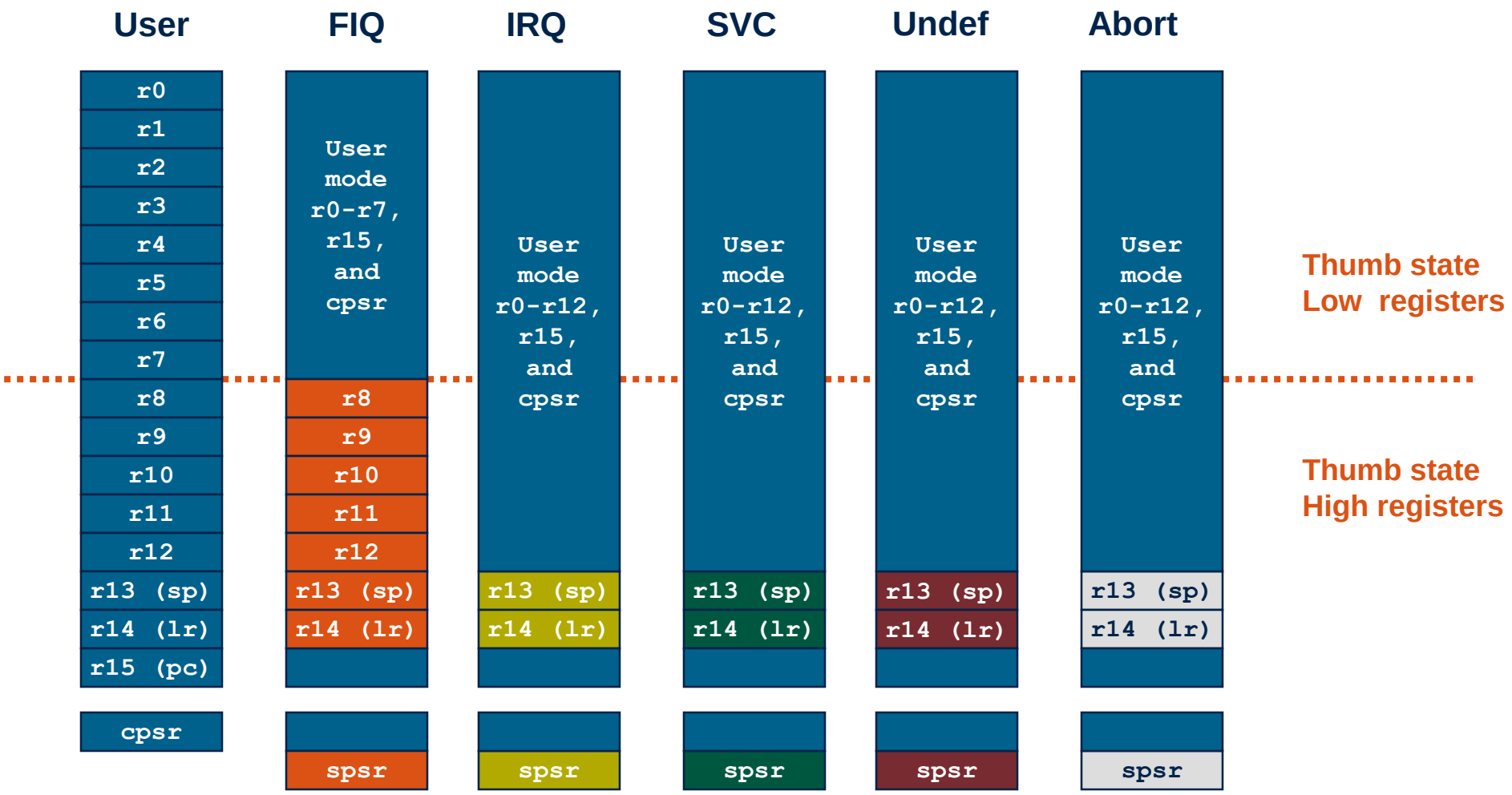
+26H	0010 0110
+33H	0011 0011
+59H	0101 1001

V = 0
N = 0


ARM®
THE ARCHITECTURE
FOR THE DIGITAL WORLD™


ARM®
THE ARCHITECTURE
FOR THE DIGITAL WORLD™

Register Organization Summary



Note: System mode uses the User mode register set

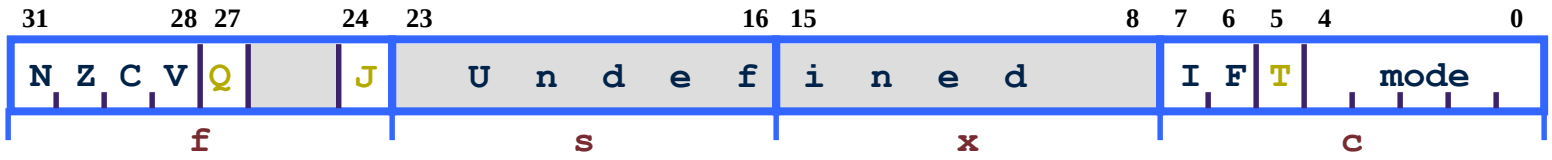
The Registers

- **ARM has 37 registers all of which are 32-bits long.**
  - 1 dedicated program counter
  - 1 dedicated current program status register
  - 5 dedicated saved program status registers
  - 30 general purpose registers

- **The current processor mode governs which of several banks is accessible. 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 spsr (saved program status register)

Program Status Registers



- **Condition code flags**
  - N = Negative result from ALU
  - Z = Zero result from ALU
  - C = ALU operation Carried out
  - V = ALU operation oVerflowed
- **Sticky Overflow flag - Q flag**
  - Architecture 5TE/J only
  - Indicates if saturation has occurred
- **J bit**
  - Architecture 5TEJ only
  - J = 1: Processor in Jazelle state
- **Interrupt Disable bits.**
  - I = 1: Disables the IRQ.
  - F = 1: Disables the FIQ.
- **T Bit**
  - Architecture xT only
  - T = 0: Processor in ARM state
  - T = 1: Processor in Thumb state
- **Mode bits**
  - Specify the processor mode

Program Counter (r15)

- **When the processor is executing in ARM state:**
  - All instructions are 32 bits wide
  - All instructions must be word aligned
  - Therefore the **pc** value is stored in bits [31:2] with bits [1:0] undefined (as instruction cannot be halfword or byte aligned).

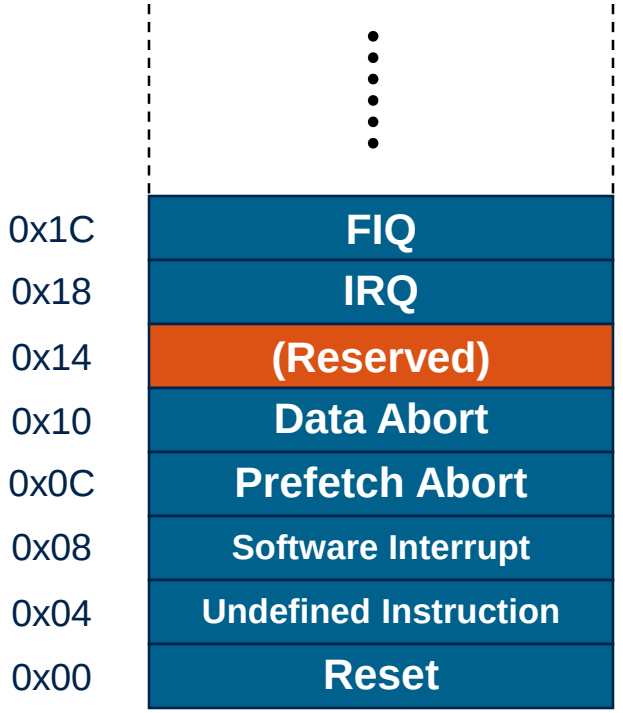
- **When the processor is executing in Thumb state:**
  - All instructions are 16 bits wide
  - All instructions must be halfword aligned
  - Therefore the **pc** value is stored in bits [31:1] with bit [0] undefined (as instruction cannot be byte aligned).

- **When the processor is executing in Jazelle state:**
  - All instructions are 8 bits wide
  - Processor performs a word access to read 4 instructions at once

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>

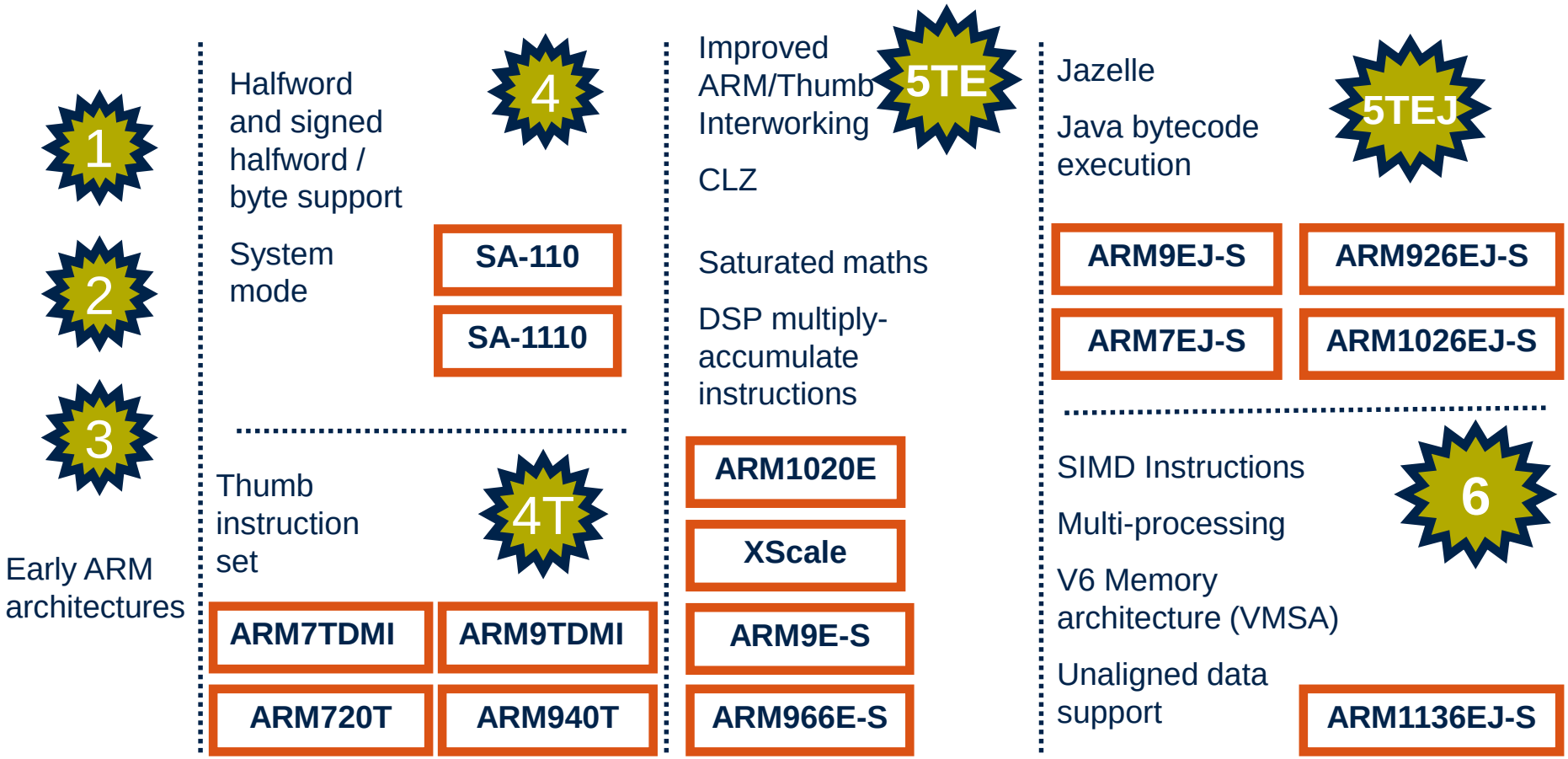
  This can only be done in ARM state.



Vector Table

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

Development of the ARM Architecture



Agenda

- Introduction to ARM Ltd
- Programmers Model
- **Instruction Sets**
- System Design
- Development Tools

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
  BEQ   skip
  ADD   r0,r1,r2
skip
```

```
CMP   r3,#0
 ADDNE r0,r1,r2
```

- **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
  BNE loop
```

decrement r1 and set flags

if Z flag clear then branch

Condition Codes

- **The possible condition codes are listed below:**
  - Note AL is the default and does not need to be specified

Suffix	Description	Flags tested
EQ	Equal	Z=1
NE	Not equal	Z=0
CS/HS	Unsigned higher or same	C=1
CC/LO	Unsigned lower	C=0
MI	Minus	N=1
PL	Positive or Zero	N=0
VS	Overflow	V=1
VC	No overflow	V=0
HI	Unsigned higher	C=1 & Z=0
LS	Unsigned lower or same	C=0 or Z=1
GE	Greater or equal	N=V
LT	Less than	N!=V
GT	Greater than	Z=0 & N=V
LE	Less than or equal	Z=1 or N=!V
AL	Always	

Examples of conditional execution

- Use a sequence of several conditional instructions

```
if (a==0) func(1);
    CMP     r0,#0
    MOVEQ   r0,#1
    BLEQ    func
```

- Set the flags, then use various condition codes

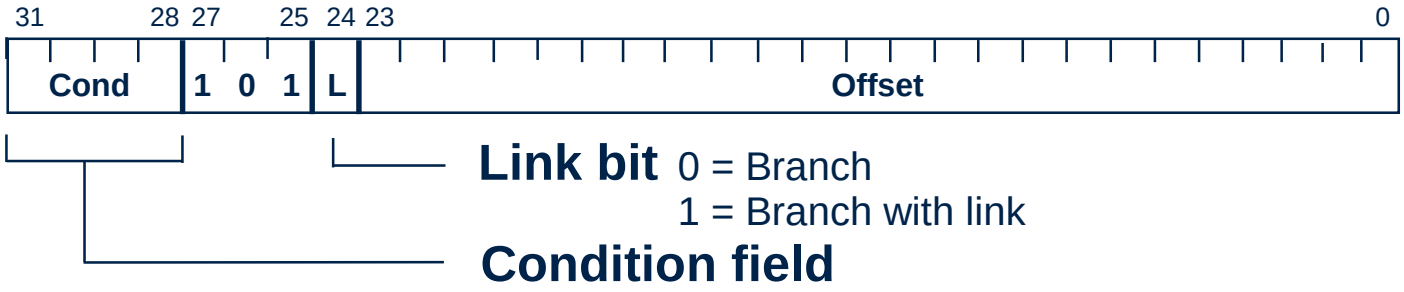
```
if (a==0) x=0;
if (a>0)  x=1;
    CMP     r0,#0
    MOVEQ   r1,#0
    MOVGT   r1,#1
```

- Use conditional compare instructions

```
if (a==4 || a==10) x=0;
    CMP     r0,#4
    CMPNE   r0,#10
    MOVEQ   r1,#0
```

Branch instructions

- **Branch :** `B{<cond>} label`
- **Branch with Link :** `BL{<cond>} subroutine_label`



- **The processor core shifts the offset field left by 2 positions, sign-extends it and adds it to the PC**
  - ± 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.**

The Barrel Shifter

LSL : Logical Left Shift

Multiplication by a power of 2

ASR: Arithmetic Right Shift

Division by a power of 2,
preserving the sign bit

LSR : Logical Shift Right

Division by a power of 2

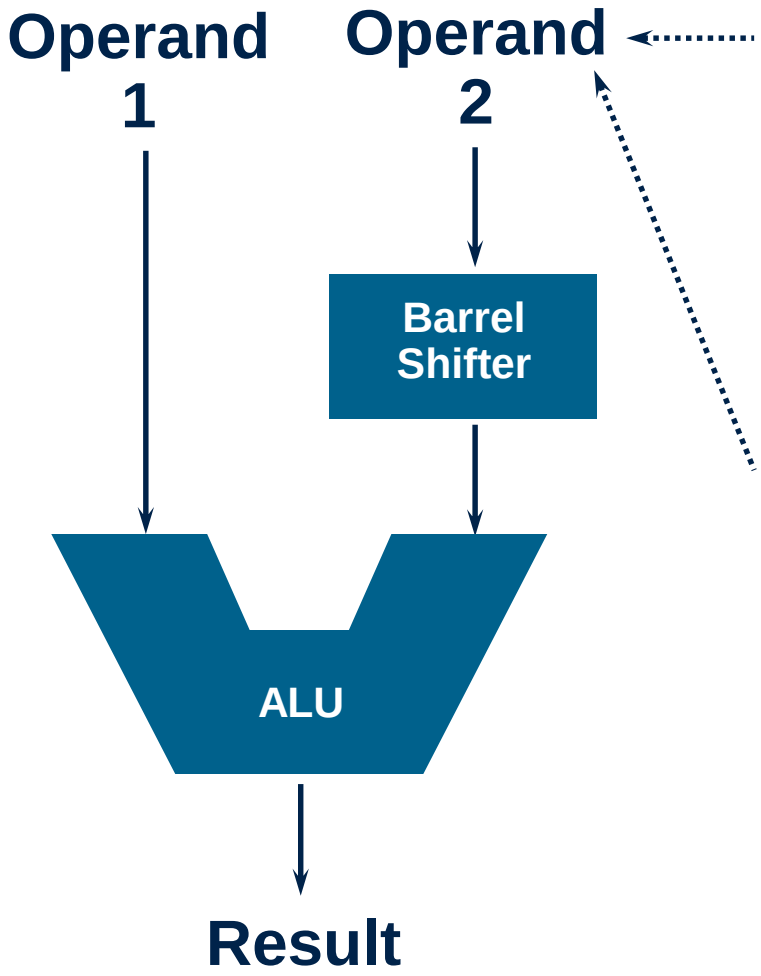
ROR: Rotate Right

Bit rotate with wrap around
from LSB to MSB

RRX: Rotate Right Extended

Single bit rotate with wrap around
from CF to MSB

Using the Barrel Shifter: The Second Operand



Register, optionally with shift operation

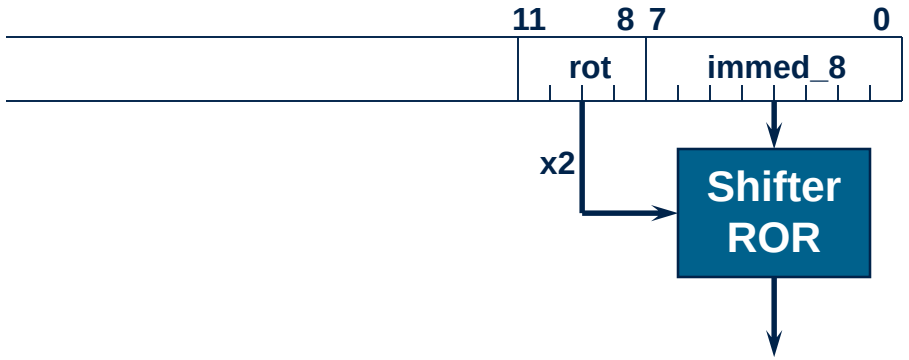
- Shift value can be either be:
  - 5 bit unsigned integer
  - Specified in bottom byte of another register.
- Used for multiplication by constant

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

Immediate constants (1)

- **No ARM instruction can contain a 32 bit immediate constant**
  - All ARM instructions are fixed as 32 bits long
- **The data processing instruction format has 12 bits available for operand2**



Quick Quiz:

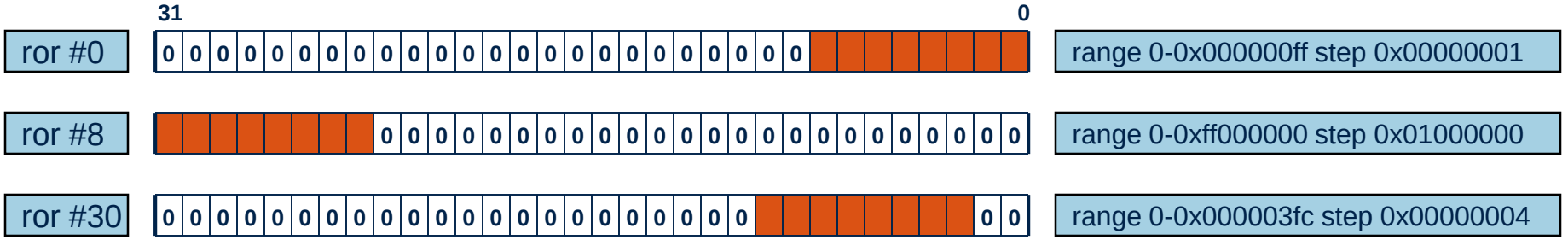
`0xe3a004ff`

`MOV r0, #???`

- **4 bit rotate value (0-15) is multiplied by two to give range 0-30 in steps of 2**
- **Rule to remember is "8-bits shifted by an even number of bit positions".**

Immediate constants (2)

- **Examples:**



- **The assembler converts immediate values to the rotate form:**
  - `MOV r0,#4096 ; uses 0x40 ror 26`
  - `ADD r1,r2,#0xFF0000 ; uses 0xFF ror 16`

- **The bitwise complements can also be formed using MVN:**
  - `MOV r0, #0xFFFFFFFF ; assembles to MVN r0,#0`

- **Values that cannot be generated in this way will cause an error.**

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).

  or
  - 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]`

```
LDR r0,[PC,#Imm12]
...
...
DCD 0x55555555
```

- **This is the recommended way of loading constants into a register**

Multiply

- **Syntax:**
  - MUL{<cond>}{S} Rd, Rm, Rs — Rd = Rm * Rs
  - MLA{<cond>}{S} Rd,Rm,Rs,Rn — Rd = (Rm * Rs) + Rn
  - [U|S]MULL{<cond>}{S} RdLo, RdHi, Rm, Rs — RdHi,RdLo := Rm*Rs
  - [U|S]MLAL{<cond>}{S} RdLo, RdHi, Rm, Rs — RdHi,RdLo := (Rm*Rs)+RdHi,RdLo

- **Cycle time**
  - Basic MUL instruction
    - 2-5 cycles on ARM7TDMI
    - 1-3 cycles on StrongARM/XScale
    - 2 cycles on ARM9E/ARM102xE
  - +1 cycle for ARM9TDMI (over ARM7TDMI)
  - +1 cycle for accumulate (not on 9E though result delay is one cycle longer)
  - +1 cycle for “long”

- **Above are “general rules” - refer to the TRM for the core you are using for the exact details**

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

Address accessed

- **Address accessed by LDR/STR is specified by a base register plus an offset**
- **For word and unsigned byte accesses, offset can be**
  - An unsigned 12-bit immediate value (ie 0 - 4095 bytes).

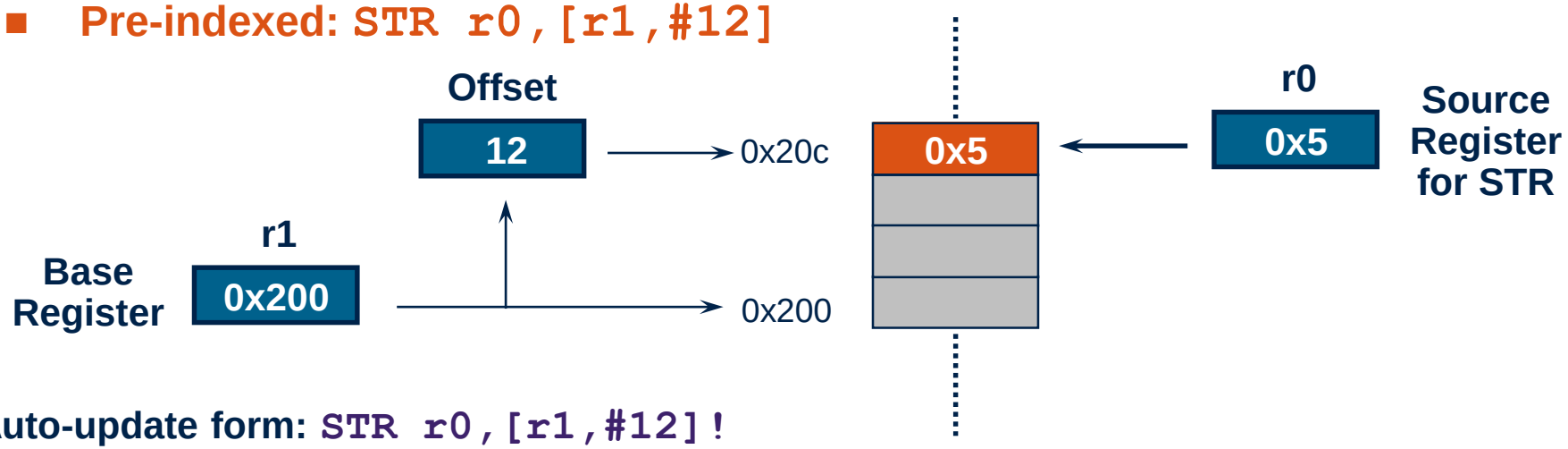
    ```
    LDR r0,[r1,#8]
    ```
  - A register, optionally shifted by an immediate value

    ```
    LDR r0,[r1,r2]
    LDR r0,[r1,r2,LSL#2]
    ```
- **This can be either added or subtracted from the base register:**

  ```
  LDR r0,[r1,#-8]
  LDR r0,[r1,-r2]
  LDR r0,[r1,-r2,LSL#2]
  ```
- **For halfword and signed halfword / byte, offset can be:**
  - An unsigned 8 bit immediate value (ie 0-255 bytes).
  - A register (unshifted).
- **Choice of *pre-indexed* or *post-indexed* addressing**

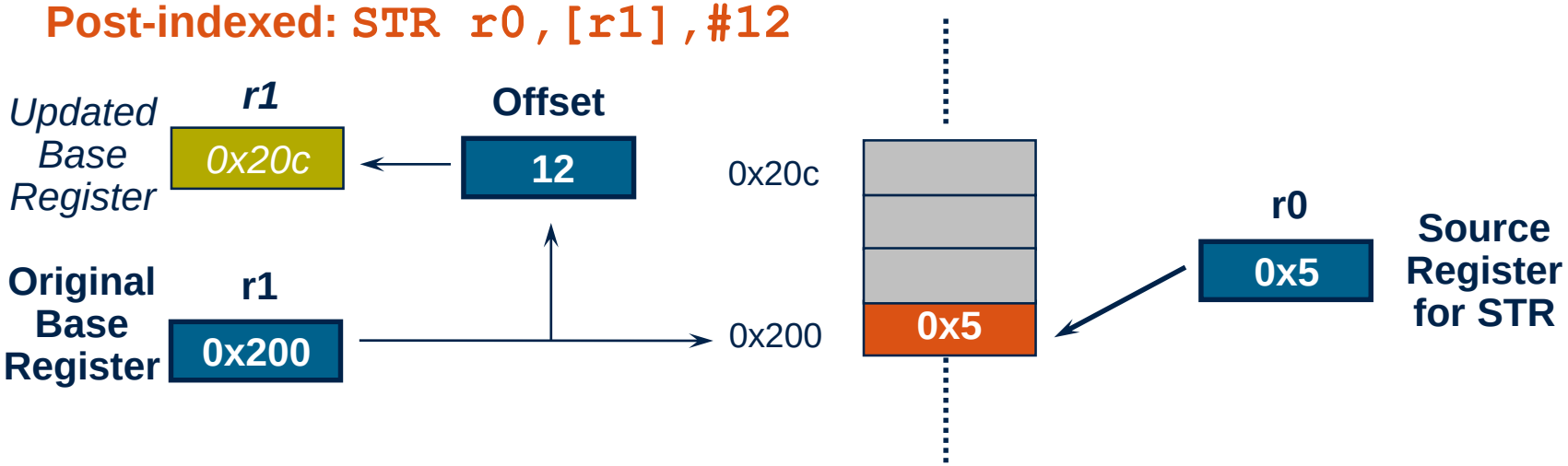
Pre or Post Indexed Addressing?

- Pre-indexed: `STR r0,[r1,#12]`



Auto-update form: `STR r0,[r1,#12]!`

- Post-indexed: `STR r0,[r1],#12`



LDM / STM operation

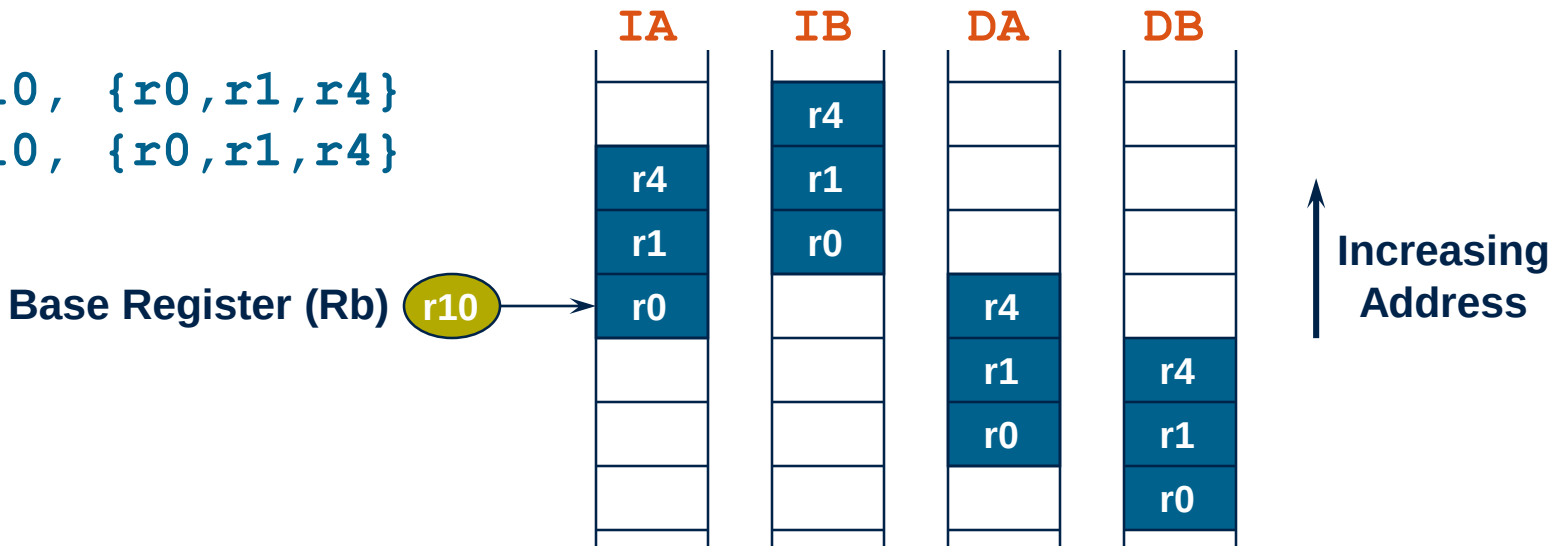
- **Syntax:**

  `<LDM|STM>`{<cond>}<addressing_mode> Rb{!}, <register list>

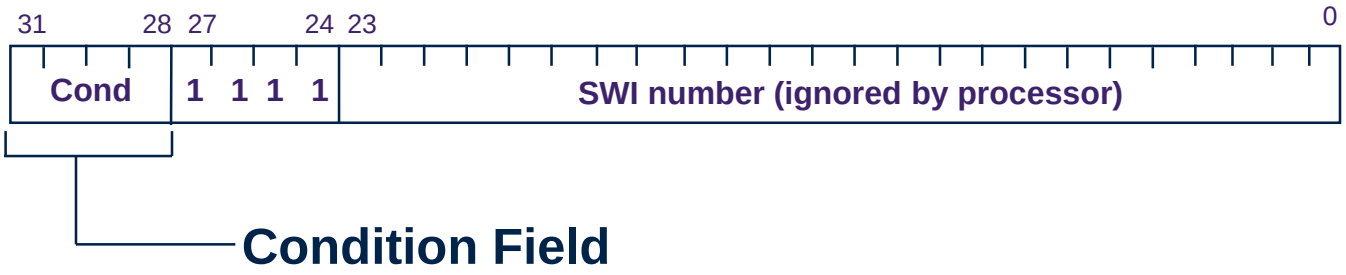
- **4 addressing modes:**

`LDMIA / STMIA`	increment after
`LDMIB / STMIB`	increment before
`LDMDA / STMDA`	decrement after
`LDMDB / STMDB`	decrement before

```
LDMxx r10, {r0,r1,r4}
STMxx r10, {r0,r1,r4}
```

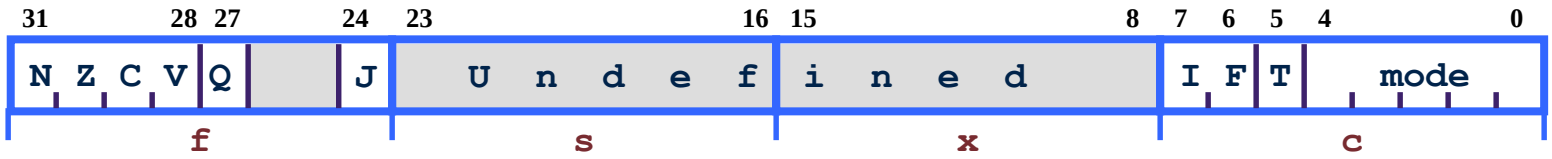


Software Interrupt (SWI)



- Causes an exception trap to the SWI hardware vector
- The SWI handler can examine the SWI number to decide what operation has been requested.
- By using the SWI mechanism, an operating system can implement a set of privileged operations which applications running in user mode can request.
- Syntax:
  - `SWI{<cond>} <SWI number>`

PSR Transfer Instructions

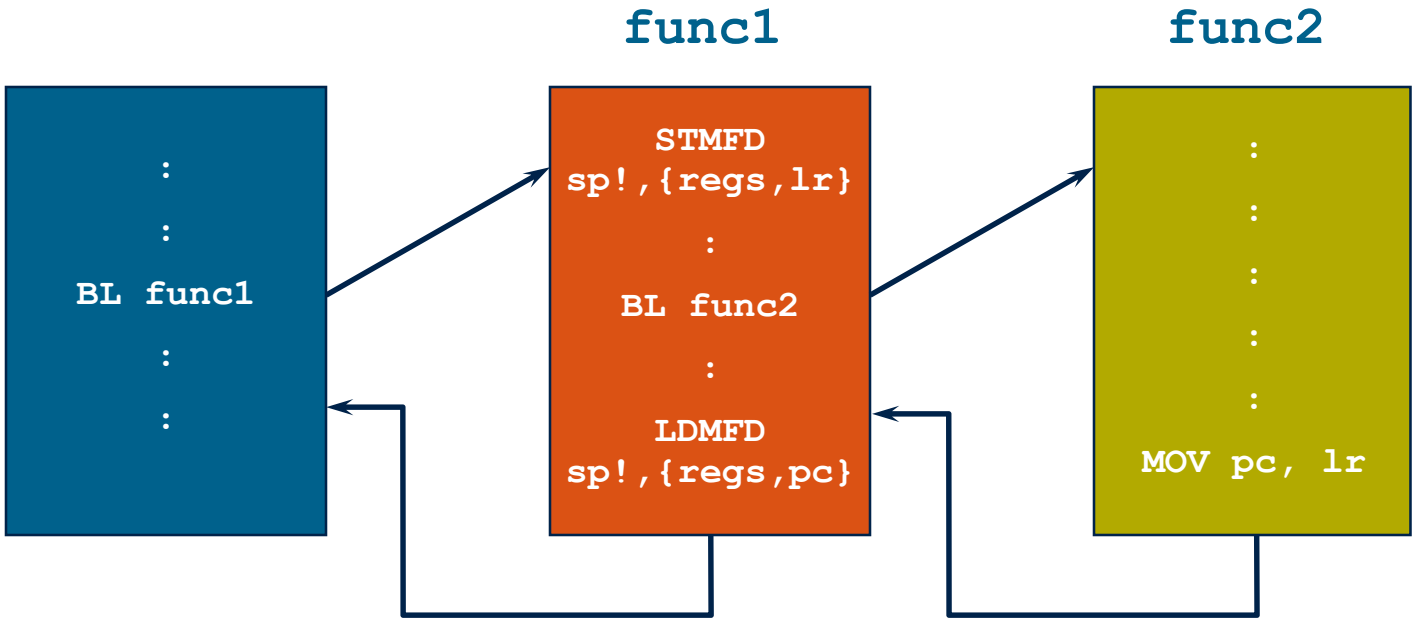


- MRS and MSR allow contents of CPSR / SPSR to be transferred to / from a general purpose register.
- Syntax:
  - `MRS{<cond>} Rd,<psr>  ; Rd = <psr>`
  - `MSR{<cond>} <psr[_fields]>,Rm  ; <psr[_fields]> = Rm`

  where
  - `<psr> = CPSR or SPSR`
  - `[_fields] = any combination of 'fsxc'`
- Also an immediate form
  - `MSR{<cond>} <psr_fields>,#Immediate`
- In User Mode, all bits can be read but only the condition flags (_f) can be written.

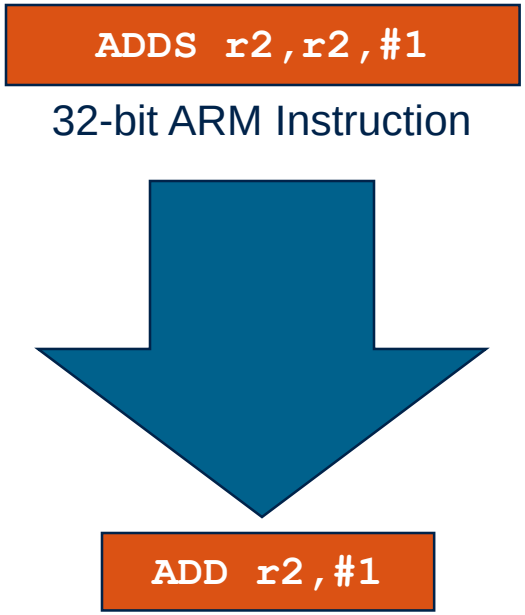
ARM Branches and Subroutines

- **B <label>**
  - PC relative. ±32 Mbyte range.
- **BL <subroutine>**
  - Stores return address in LR
  - Returning implemented by restoring the PC from LR
  - For non-leaf functions, LR will have to be stacked



Thumb

- **Thumb is a 16-bit instruction set**
  - Optimised for code density from C code (~65% of ARM code size)
  - Improved performance from narrow memory
  - Subset of the functionality of the ARM instruction set
- **Core has additional execution state - Thumb**
  - Switch between ARM and Thumb using `BX` instruction



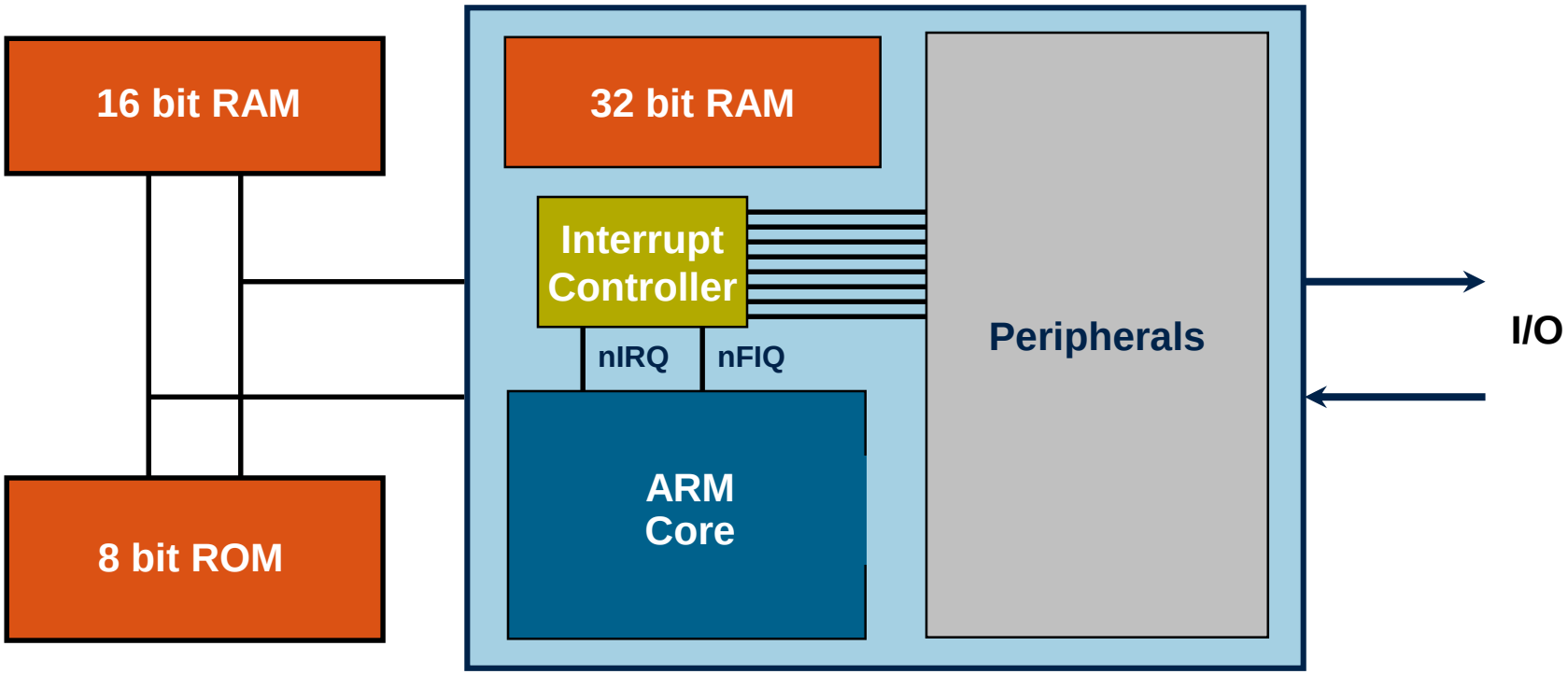
For most instructions generated by compiler:

- Conditional execution is not used
- Source and destination registers identical
- Only Low registers used
- Constants are of limited size
- Inline barrel shifter not used

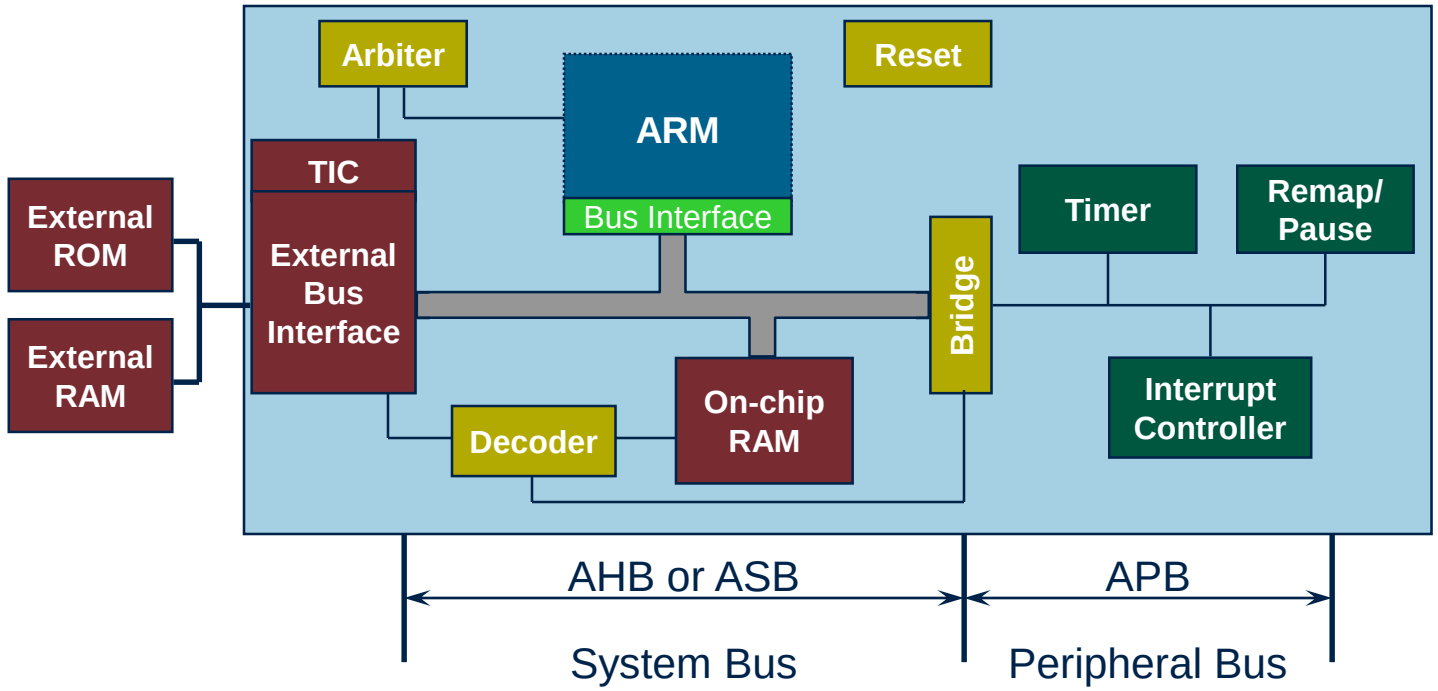
Agenda

- Introduction
- Programmers Model
- Instruction Sets
- **System Design**
- Development Tools

Example ARM-based System



AMBA



- **AMBA**
  - Advanced Microcontroller Bus Architecture
- **ADK**
  - Complete AMBA Design Kit
- **ACT**
  - AMBA Compliance Testbench
- **PrimeCell**
  - ARM's AMBA compliant peripherals

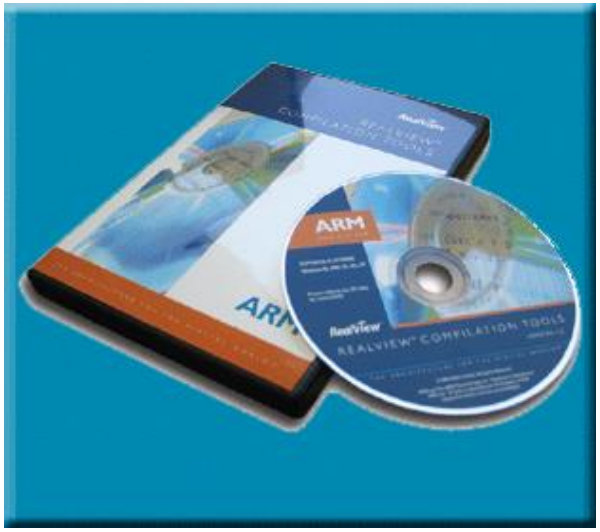
Agenda

- Introduction
- Programmers Model
- Instruction Sets
- System Design
- **Development Tools**

The RealView Product Families

Compilation Tools

ARM Developer Suite (ADS) – Compilers (C/C++ ARM & Thumb), Linker & Utilities



RealView Compilation Tools (RVCT)

Debug Tools

AXD (part of ADS)

Trace Debug Tools

Multi-ICE

Multi-Trace

RealView Debugger (RVD)

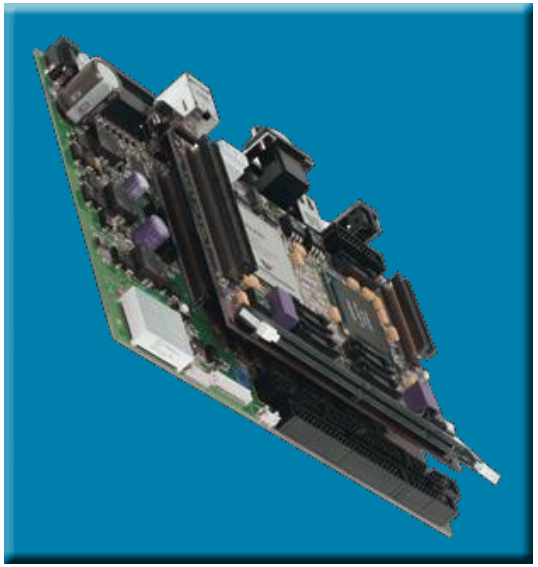
RealView ICE (RVI)

RealView Trace (RVT)

Platforms

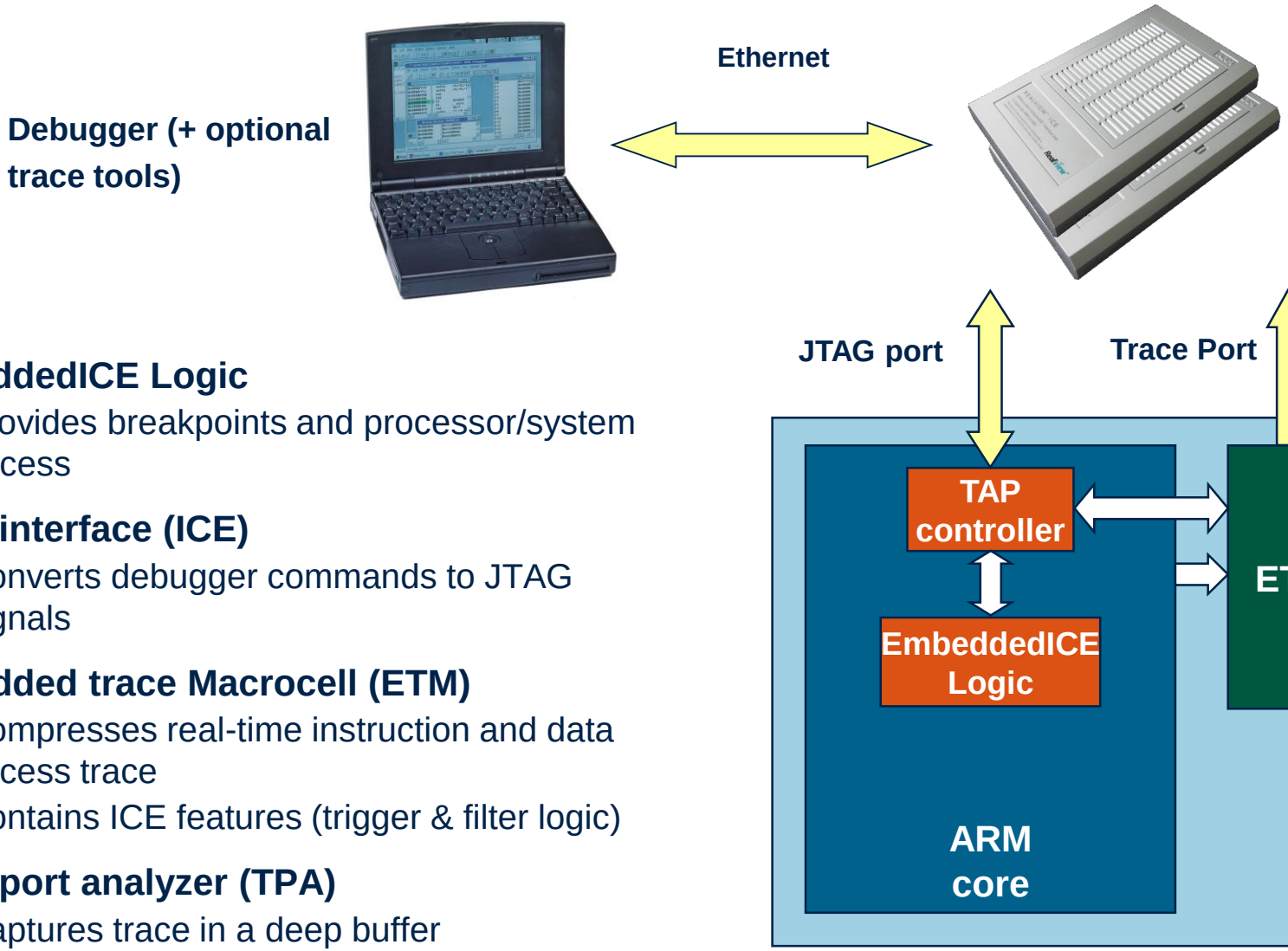
ARMulator (part of ADS)

Integrator™ Family



RealView ARMulator ISS (RVISS)

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


ARM®
THE ARCHITECTURE
FOR THE DIGITAL WORLD™