ARM Architecture and Multimedia Applications

Joško Rokov, dipl.ing.
RIZ-Transmitters Co.
Zagreb, Croatia
josko.rokov@riz.hr

Abstract—An overview of ARM processor architecture evolution with emphasis to specific instruction sets capable of executing the demanding high speed real-time multimedia applications is provided. Basic ARM architecture is presented, together with the hardware architecture evolution. Specifics of the instruction set extensions based on SIMD style processing are described. Most important performance issues and speed restriction problems are analyzed.

Keywords – ARM, Hardware architecture, ISA, SIMD, NEON, Multimedia applications

I. INTRODUCTION

The ARM processor is a key component of many embedded systems [1]. First ARM prototype was introduced in 1985 under the name of Acorn RISC Machine (afterwards renamed to Advanced RISC Machine). The relative simplicity of ARM processors made them suitable for low consumption applications. Currently ARM processor based solutions power over 90 percent of current 3G baseband products [2].

The paper is structured as follows. Section II shows the description of basic ARM architecture. It includes presentation of processor datapath, pipeline, register file, basic instructions and data types. Section III describes the evolution process of ARM architecture through hardware architecture changes and instruction set extensions. Section IV gives an insight to ARM multimedia instruction sets. Section V gives an overview of the performance issues and bottlenecks encountered with multimedia instruction sets. The conclusion is summarized in section VI.

II. BASIC ARM ARCHITECTURE

A. General

The basic ARM architecture can be analyzed on ARM7 core (ARMv4 revision) which is a 32-bit Reduced Instruction Set Computer (RISC) [3]. It incorporates these typical RISC architecture features:

- A large uniform register file
- A load/store architecture, where data-processing operations only operate on register contents, not directly on memory contents.
- Simple addressing modes, with all load/store addresses being determined from register contents and instruction fields only
- Uniform and fixed-length instruction fields, to simplify instruction decode

In addition the ARM architecture provides:

- Control over both the Arithmetic Logic Unit (ALU) and shifter in most data-processing instructions to maximize the use of ALU and a shifter
- Auto-increment and auto-decrement addressing modes to optimize program loops
- Load and Store Multiple instructions to maximize data throughput
- Conditional execution of almost all instructions to maximize execution throughput

Block diagram of the basic ARM architecture dataflow is shown on Figure 1.

![Figure 1. ARM core dataflow model](image.png)

Data and instructions share the same bus which makes the Von-Neumann type of architecture (changed to Harvard architecture in ARM9 – ARMv4T revision). The pipeline has three stages:

- Fetch – loads an instruction from memory
- Decode – identifies the instruction to be executed
- Execute – processes the instruction and writes the result back to a register
The hardware of each stage is designed to be independent so up to three instructions can be processed simultaneously.

B. Registers

The ARMv4 core has 37 general purpose 32-bit registers as shown on Figure 3. Only 16 registers are accessible to all register specifiers in ARM instructions and are called User mode registers. Three of the 16 visible registers have special roles:

- r13 - Stack pointer
- r14 - Link register
- r15 - Program counter

These 16 User mode registers constitute a storage bank called the register file.

C. Basic instructions and datatypes

Different ARM architecture revisions support different instructions. Basic instructions can be grouped into following:

- Data processing instructions such as move, logical and arithmetic shift, rotation, etc
- Branch instructions used for changing flow of execution
- Load/store instructions used for transferring data between memory and processor registers
- Software interrupt instruction used for causing software interrupt exception
- Program status register instructions used for direct control of program status register bits

Most of the instructions are executed conditionally based on flag bits state in program status register which are the product of previously executed operation. Basic flags are N (Negative), C (Carry), Z (Zero) and V (Overflow).

The ARMv4 core supported data types are signed and unsigned words (32-bit), halfwords (16-bit) and bytes.

D. Thumb instruction set

Thumb instruction code encodes a subset of the 32-bit ARM instructions into a 16-bit instruction set space.

Coding using Thumb operations has higher code-density than coding using normal ARM operations if operations are executed on both 32-bit and smaller data. Taking in consideration that an average code takes up around 30% less memory than the equivalent ARM implementation it was ideal for usage in memory-constrained embedded systems.

Every Thumb operation is related to a 32-bit ARM instruction and Thumb-instruction decoder is placed in pipeline as shown on Figure 4.

Figure 2. ARMv4 pipeline

Figure 3. ARMv4 full register set

Figure 4. Thumb instruction decoding in ARM pipeline

Thumb-2 extension was introduced in ARMv6T2 revision and it included an upgrade with additional 32-bit instructions to
the Thumb instruction set to improve performance and code density.

III. ARM ARCHITECTURE EVOLUTION

The ARM architecture has continued to evolve since the first ARM processor. Table 1 shows the significant architecture enhancements [4], [5], [6] from the original architecture version 1 to the current version 7 architecture.

<table>
<thead>
<tr>
<th>Revision</th>
<th>Example core implementation</th>
<th>ISA enhancement</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARMv1</td>
<td>ARM1</td>
<td>First ARM processor</td>
</tr>
<tr>
<td>ARMv2</td>
<td>ARM2</td>
<td>26-bit addressing</td>
</tr>
<tr>
<td>ARMv2a</td>
<td>ARM3</td>
<td>32-bit multiplier</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32-bit coprocessor support</td>
</tr>
<tr>
<td>ARMv3</td>
<td>ARM6 and ARM7DI</td>
<td>On-chip cache</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Atomic swap instruction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Coprocessor 15 for cache management</td>
</tr>
<tr>
<td>ARMv3M</td>
<td>ARM7M</td>
<td>32-bit addressing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Separate cpsr and spsr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New modes—undefined instruction and abort</td>
</tr>
<tr>
<td>ARMv4</td>
<td>StrongARM</td>
<td>Signed and unsigned long multiply instructions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Signed halfwords/bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New mode—system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reserve SWI space for architecturally defined operations</td>
</tr>
<tr>
<td>ARMv4T</td>
<td>ARM7TDMI and ARM9T</td>
<td>26-bit addressing mode no longer supported</td>
</tr>
<tr>
<td>ARMv5TE</td>
<td>ARM9E and ARM10E</td>
<td>Thumb instruction set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Superset of the ARMv4T</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Extra instructions added for changing state between ARM and Thumb</td>
</tr>
<tr>
<td>ARMv5TEJ</td>
<td>ARM7EJ and ARM92EJ</td>
<td>Enhanced multiply instructions</td>
</tr>
<tr>
<td></td>
<td>ARM11</td>
<td>Extra DSP-type instructions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Faster multiply accumulate</td>
</tr>
<tr>
<td>ARMv6</td>
<td></td>
<td>Java acceleration</td>
</tr>
<tr>
<td>ARMv7m</td>
<td>Cortex-M4, Cortex-R5 and</td>
<td>Improved multiprocessor instructions</td>
</tr>
<tr>
<td>ARMv7r</td>
<td>Cortex-A9</td>
<td>Unaligned and mixed endian data handling</td>
</tr>
<tr>
<td>ARMv7a</td>
<td></td>
<td>New multimedia instructions – SIMD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enhanced exception handling</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enhanced Thumb-2 instruction set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Introduced NEON – advanced SIMD extension</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Introduced 3 different profiles: M-microcontroller, R-real-time, A-application</td>
</tr>
</tbody>
</table>

With each new revision new enhancements to existing operations were added, new instructions introduced, architecture changed and improved with additional elements.

ARMv4T introduced transition from Von-Neumann to Harvard architecture which divided the instruction and data bus enabling efficient memory management.

With this change pipeline stages also started to grow from 3 to 5 in ARMv4T, 6, 7, 15 (ARM Cortex A9) and massively 24 in the latest ARM Cortex A15 architecture.

Starting with latest ARMv7 revision ARM core implementations were divided in 3 different profiles:

- ARMv7M - microcontroller profile
- ARMv7R - real-time profile
- ARMv7A - application profile

The microcontroller profile supports only Thumb instruction set and is suitable for systems where overall size and deterministic operation for an implementation are more important than absolute performance.

The real-time profile and application profile both support ARM and Thumb instruction sets with a distinction that systems implemented using real-time profile require physical address only support in the memory management model, while systems implemented using application profile require virtual address support (based on memory management unit).

Most powerful ARM processor currently is Cortex-A15 [7]. It is targeted to high-end wireless, Smartphone, gaming, networking and server applications. Basic architectural updates are large physical addressing (up to 1TB of memory), virtualization, ISA extensions (integer divide, fused MAC), multiprocessing, AMBA 4 bus architecture [8] and Error Correction Control (ECC) on L1 and L2 memories.

Multiprocessing is supported (quad cores) with processor coherency in L2 cache (up to 4MB) connected externally to 128-bit AMBA 4 interface as shown on Figure 5.

![Quad Cortex-A15 MPCore](image)

Cortex-A15 has a 15-stage integer pipeline with 4 extra cycles for multiply and load/store instructions, and 2-10 extra cycles for complex media instructions (NEON).

Out-Of-Order execution is used to increase instruction parallelism and reduce mispredict penalty. Also, execution stage is broken down to multiple clusters defined by instruction type:

- Simple cluster – single cycle integer operations
- Complex cluster – NEON and Floating point data processing operations
- Branch cluster
• Multiply and Divide cluster
• Load/Store cluster

All these stated improvements lead to dramatic increase in single-thread and overall performance in comparison to other previous versions of ARM architecture system implementations.

IV. MULTIMEDIA INSTRUCTION SETS

Multimedia applications have become the dominating workload for modern computer systems. They require fast and parallel execution of simple operations on small data usually narrower than the general purpose processors (including embedded) data path buses. This means that usage of normal operations causes that only a fraction of data path and functional units are actually utilized [9]. Multimedia extensions recognize that by partitioning functional units processing resources which can be utilized more efficiently. These extensions are based on the SIMD style processing method [10]. SIMD processing uses a single instruction to perform the same operation in parallel on multiple data elements.

SIMD operations have been implemented in both the Intel x86 (Intel 64, IA-32) and the ARM architecture. MMX instruction set [11] was the first introduced in Intel architecture. Streaming SIMD Extensions (SSE, SSE2, SSE3 and SSE4) [12] were added gradually. Most recent extension to the Intel architecture is the Advanced Vector Extension (AVX) [13].

First SIMD extension was included in ARMv6 architecture, and was further expanded with the NEON extension in the ARMv7 architecture.

A. ARM SIMD extension

ARMv6 architecture introduced a small set of SIMD instructions operating on multiple 16-bit or 8-bit values packed into standard 32-bit general purpose registers [14]. This permits execution of certain operations twice or four times as quickly, without implementing additional computation units. The mnemonics for these instructions are recognized by having 8 or 16 appended to the base form, indicating the size of data values operated on. An example of ADD operation which performs parallel addition of four lanes of 8-bit elements is shown on Figure 6.

B. ARM NEON extension

ARMv7 architecture introduced the Advanced SIMD extension as an optional extension to the ARMv7-A and ARMv7-R profiles. It extends the SIMD concept by defining groups of instructions operating on vectors stored in 64-bit, doubleword, registers and 128-bit Q, quadword, vector registers.

The implementation of the Advanced SIMD extension used in ARM processors is called NEON, and this is the common terminology used outside architecture specifications. The NEON instructions perform memory accesses, data copying between NEON and general purpose registers, data type conversions and data processing. An example of ADD operation which performs parallel addition of four lanes of 8-bit elements is shown on Figure 7.

Figure 7. ARM NEON operation example

The NEON instructions support 8-bit, 16-bit, 32-bit and 64-bit signed and unsigned integers. NEON also supports 32-bit single-precision floating point elements, and 8-bit and 16-bit polynomials. The NEON register bank consists of 32 64-bit registers. If both Advanced SIMD and Vector Floating Point version 3 (VFPv3) are implemented, they share this register bank.

The NEON unit can view the same register bank as:
• Sixteen 128-bit quadword registers, Q0-Q15
• Thirty-two 64-bit doubleword registers, D0-D31

Data inside the register can be packed in following manners:
• 16 integer bytes (16 x 8 bits)
• 8 integer words (8 x16 bits)
• 8 half-precision floating point numbers (8 x 16 bits)
• 4 integer doublewords (4 x 32 bits)
• 4 single precision floating point numbers (4 x 32 bits)
• 2 integer quad words (2 x 64 bits)

V. PERFORMANCE ISSUES

Talla et al. [15] in their analysis of SIMD style media processing stated that it is not possible to get significant amounts of additional performance improvements by merely increasing the SIMD resources. Based on the investigation of loop structures and access patterns in multimedia algorithms
they found that a significant amount of parallelism lies outside the innermost loops and it is difficult for SIMD units to exploit the parallelism.

There are two distinct sets of operations: the true/core computations and the overhead/supporting instructions. True/core computation instructions for the e.g. DCT routine are simple multiply and accumulates. All other instructions are denoted as overhead. This overhead component includes loop branches and reduction operations in addition to the memory access task. The instructions contributing to the overhead are following:

- Address generation – address calculations required to access the components of the data structures/arrays
- Address transformation – transformation of the physical data patterns into the logical access sequence
- Loads and stores – so called access overhead – data transfer between the memory and the processor
- Branches – performing control transfers

Increasing the number of SIMD execution units does not impact performance positively, which leads to conclusion that resources for overhead/supporting instructions need to be scaled.

A nonaligned or irregular data access operation incurs many overhead cycles for data alignment. Some propose split line buffer and packing buffer [16], while others concentrate on solution using data permutations patterns [17]. Data alignment is the main obstacle for automatic vectorization of programs for partitioned SIMD processors.

Other important overhead problem arises in data packing and unpacking due to opposite requirements for minimum bits stored and maximum of result precision (storage and calculated data are different in some cases). Solution was proposed which would perform pack and unpack operations within the load and store operations.

VI. CONCLUSION

This paper presents an overview of ARM architecture starting from initial design with 3-stage pipeline and Von-Neumann architecture to newest ARM Cortex based embedded systems. Special emphasis is given to ARM extensions (SIMD and NEON) aimed for optimization of multimedia applications.

Some problems are brought out to attention and could serve as a starting point for the future improvements of processor architecture and instruction sets aimed specifically for the multimedia applications.

REFERENCES


