A Qualitative Comparison of ARM v5 and AVR32 architectures

Rahul Sharma, Aby Kuruvilla, Graduate students of Computer Engineering, UNC Charlotte.

(Submitted in partial completion of requirements for ECGR5181 – Computer Architecture course at UNC Charlotte)

Table of Contents

Sr. No.	Description	Page No.
1.	Abstract…………………………………………………………………………	2
2.	Introduction………………………………………………………………………..	4
3.	Arithmetic Logic Unit………………………………………………………………..	5
4.	Pipeline…………………………………………………………………………	8
5.	Memory Map and Memory Management Unit………………………………	12
6.	Modes of Operation…………………………………………………………....	14
7.	Instruction execution and Exceptions………………………………………....	16
8.	Instruction sets and addressing modes……………………………………….	18
9.	Benchmarks…………………………………………………………………….	20
10.	Abbreviations……………………………………………………………………..	23
11.	References…………………………………………………………………….	24

ABSTRACT

This project, a comparison of architectures attempts to compare two relevant and current architecture philosophies. ARM has been the dominant 32 bit architecture choice for Embedded Systems engineering for more than 20 years now. The relatively new AVR architecture (derived from Harvard Architecture) promises higher efficiency and flexibility than its competitors. ARM Corporation leases out designs of architectures whereas AVR architectures are developed and products are created by the Atmel Corporation itself. We plan to compare these architectures and suggest a non-conclusive idea of possible applications that can aid selection of microcontrollers for various projects.

Background Study

ARM

The ARM or Advanced RISC Machine (previously Acorn RISC Machine) architecture is a 32 bit RISC processor architecture developed by ARM Limited. It is widely used for embedded systems design. Due to good power saving features, ARM CPUs are used especially in mobile electronics, where power consumption is a great concern. Today the ARM family accounts for approximately 75% of all embedded 32-bit RISC CPUs. Some of its other applications include consumer electronics, portable devices and computer peripherals.

History:

The ARM design was started in 1983 at Acorn Computers Ltd. The design team was led by Roger Wilson and Steve Furber. The team completed development of the ARM1 in April 1985, with ARM 2 being produced the following year. The ARM2, with around 30,000 transistors, was the simplest 32 bit microprocessor in the world, due to no microcode and cache. The simplicity also led to low power usage. The ARM3 had a 4 KB cache, which further improved performance. From the 1980s Apple Computers started working with Acorn, which eventually led to making of the ARM6. The core has largely remained the same size throughout the changes, with optional parts added to produce a complete CPU, thus delivering a lot of performance at low cost. ARM sells IP cores, which licensees use to create microcontrollers and CPUs based on the ARM core.

Features:

- To keep the design clean, simple and fast, it was hardwired without microcode.
- The ARM architecture uses the RISC architecture.
- Uses Load/Store architecture.
- Has an orthogonal instruction set.
- Large 16X32 register file.
- Fixed instruction width of 32 bits to ease decoding and pipelining.
- Mostly single cycle execution.

To compensate for the simple design, it has some unique design features.

- Conditional execution of most instructions, reducing branch overhead.
- Arithmetic instructions alter condition codes only when desired.

- 32-bit barrel shifter for arithmetic instructions and address calculations.
- Powerful indexed addressing modes.
- Simple and fast, 2-priority-level interrupt subsystem with register banks.
- The ARM design uses a 4-bit condition code on the front of every instruction, which makes the execution of every instruction optionally conditional.

AVR

The AVR is a RISC microprocessor architecture designed by Atmel. It has a Harvard Architecture (CPU has separate program and data memory). The AVR32 architecture has several micro-architectures(fixed additions to the Instruction set architecture). The AVR has been designed for extremely efficient code density and performance per clock cycle. Atmel has consistently outperformed many architectures like the ARMv5 16-bit (THUMB) code and ARMv5 32-bit (ARM) code by as much as 50% on code-size and 3X on performance in benchmark tests using the independent benchmark consortium EEMBC. The AVR32 architecture is solely used in Atmel's own products.

History:

The basic AVR architecture is believed to be made by two students, Alf-Egil Bogen and Vegard Wollan, of the Norwegian Institute of Technology. It was known as uRISC(Micro RISC). When the technology was sold to Atmel, the two of them improved the architecture further. The AVR is said to stand for Advanced Virtual RISC, though the company says that the acronym means nothing.

Features:

- On-chip and In-System Flash memory used as program memory. All the processors have on-chip flash program memory. This allows in-system programming (without removing from target system).
- 32X8 general purpose working registers. Thus variables can be stored inside the CPU rather than storing them in memory, which is time consuming. Thus the program executes faster.
- On-chip data memory EEPROM and RAM in most devices. The CPU is Harvard architecture(EEPROM and the RAM are seen as data memory).
- 0 to 10 MHz clock speed operation. Most instructions operate in one clock cycle(leads to 10-times performance improvement over conventional processors like 8051 at equal clock frequency).
- Power On RESET circuit.
- External and internal interrupt resources.
- Programmable watchdog timer with independent oscillator (to recover from software crash).
- On chip programmable timer and separate prescalar (used for timing applications).
- SLEEP and POWER DOWN modes of operation (saves power when processor is idle).
- Many chips with on chip RC oscillator (leads to lower component count).
- Wide device range (from small 8-pin processor to a 68 pin processor).

INTRODUCTION

ARM v5

ARM v5 is a RISC (Reduced Instruction Set Computer) which has been very popular for its small size, high performance feature. It typically encompasses:

a. Large Register File
b. Load/store architecture which works on registers rather than memory spaces.
c. Simple addressing modes
d. Uniform instruction size which helps decoding.
e. It also gives tremendous control over the ALU and shifter operations in data processing operations
f. Optimizations for program loops by auto-increment and auto-decrement.
g. Multiple instruction issue.

There are many variants to the ARM architecture

- T variants: It has the re-encoded instruction set and is half the size of regular ARM instructions, i.e. 16 bits. This gives better code density but usually increases the number of instructions for a particular task. It also misses some of the exception handling capabilities and high level exception handling needs to be done using ARM instructions. It performs well when performance of time critical code is a must.
- M Variants: This instruction set includes 4 extra instructions that can perform 32 bit multiplication, i.e. produce 64 bit output.
- E Variants: These include some instructions which enhance performance on typical DSP (Digital Signal Processing) algorithms. Some of the capabilities associated with E variants are being able to perform load/store and co-processor transfers on 2-word data and cache preload.

Evolution of the ARM

ARMv1	• Contained basic data processing instructions and an obsolete 26 bit address space • Byte, word and multi-word load/store instructions • Never used commercially
ARMv2	• Multiply instructions & co-processor support • More registers in fast interrupt mode • Atomic load/store instructions
ARMv3	• Extended addressing range of 32 bits and compatibility for 26 bit address • CPSR and SPSR introduced for exception handling • Made data abort, prefetch abort and undefined exceptions
ARMv4	• Halfword load/store and Undefined exception refined • Thumb instruction mode and privileged processor mode
ARMv5	• Some versions support Jazelle, a hardware and software support for graphics acceleration and multi-tasking. • Thumb Instruction set improved with Thumb-2
ARMv7	• More comprehensive RISC instruction set and better Thumb instruction set

	• Wide RTOS support including Palm OS, Symbian and Windows CE • Forward compatible code to ARM9 & ARM 10
ARMv9	• Integrated data and instruction cache • Better debug support
ARMv10	• Optional VPF100 co-processor gives floating point operation support • Parallel Load/store capability
ARM v11	• More efficient Java execution • Separate pipelines for load/store and Arithmetic.

AVR32

AVR32 specifies many two architectures A and B which are tailored for specific applications:

AVR32A micro-architecture: It is the cheaper version with architectures in such that there are no return address registers and hardware registers. All activities are handled by the stack. This causes slow interrupt handling but chip area is smaller. During interrupt processing, all registers are pushed to stack regardless of priority. Upon completion, stack items are popped.

AVR32B micro-architecture: It is used for interrupt latency critical applications where use of stack reduces speed.

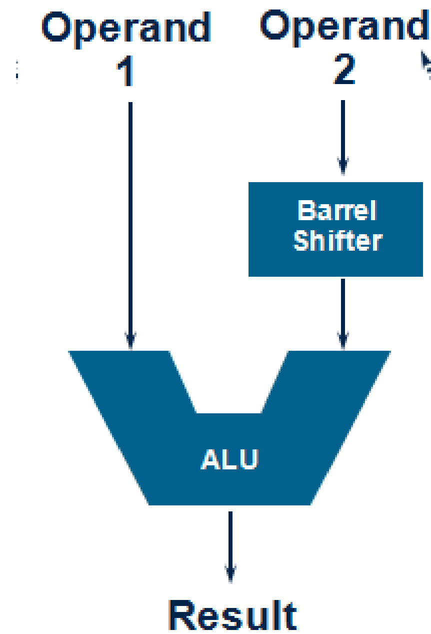
Similarities and Differences:

- AVR and ARM both provide for a large flexibility in the form of space for program and activity.
- AVR gives an option of two architectures. By reducing costs associated with costly register files by replacement with stack.

ARITHMETIC LOGIC UNIT

ARM

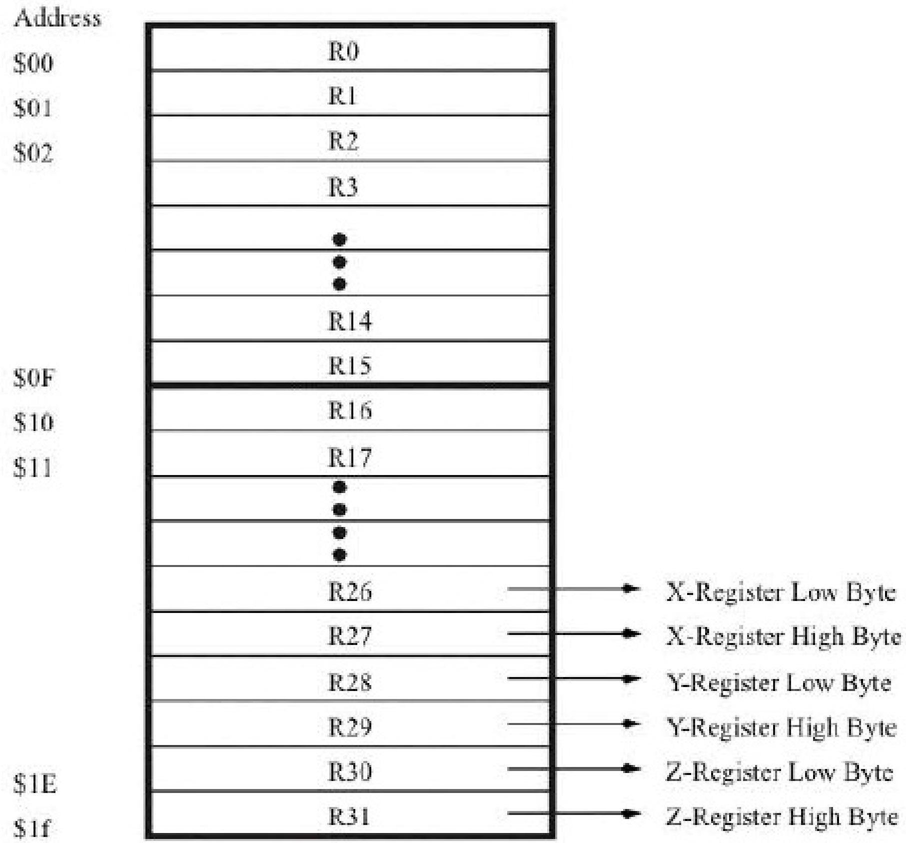
- Data processing instructions are processed within the arithmetic logic unit (ALU).
- A unique and powerful feature of the ARM processor is the ability to shift the 32-bit binary pattern in one of the source registers left or right by a specific number of positions before it enters the ALU.
- This shift increases the power and flexibility of many data processing operations.
- There are data processing instructions that do not use the barrel shift, for example, the MUL (multiply), CLZ (count leading zeros), and QADD (signed saturated 32-bit add) instructions.
- Pre-processing or shift occurs within the cycle time of the instruction is particularly useful for loading constants into a register and achieving fast multiplies or division by a power of 2.



(Source: ARM Architecture Manual & Presentation)

AVR

- The arithmetic logic unit (ALU) performs such operations as bit, arithmetic and logic upon the contents of registers and writes back the result into the register file into the designated register.
- These operations are performed in a single clock cycle. Each ALU operation affects the flags in the STATUS register, depending upon the instruction.
- Once the instruction is fetched, if it is an ALU-related instruction, it can be executed by the ALU in a single cycle.
- On the other hand, the SRAM memory access takes two cycles. This is because the SRAM access uses a pointer register for the SRAM address. The pointer register is one of the pointer registers (X, Y or Z register pairs). The first clock cycle is used to access the register file and to operate upon the pointer register. At the end of the first clock cycle, the ALU performs the calculation, and then this address is used to access the SRAM location and to write into it (or read from it into the destination register).



(Register File of AVR – Source: Programming and Customizing the AVR Microcontroller)

AVR Architecture more Code Efficient:

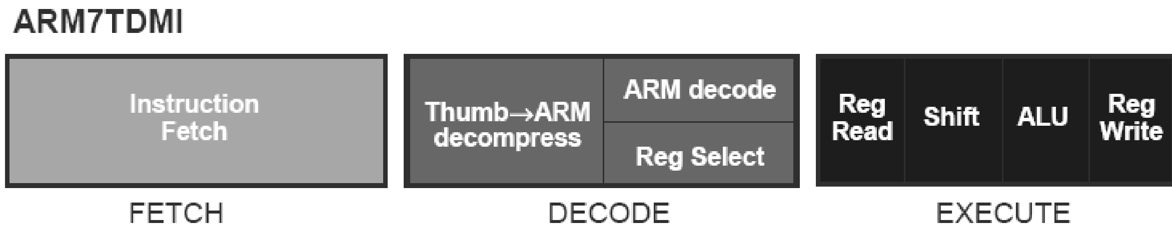
- The AVR core combines a rich instruction set with 32 general purpose working registers.
- All 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle.
- The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

PIPELINE

ARM

Evolution of the ARM Pipeline:

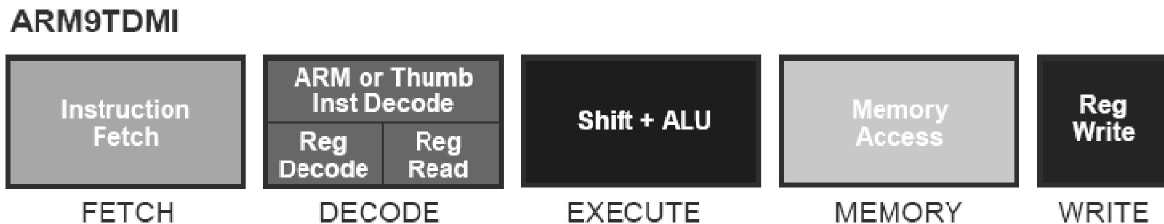
3-stage pipeline:



(Source: The ARM Architecture – http://www.eng.auburn.edu/~strouce/DaTseminar/UniPres07s.pdf)

- The original 3-stage ARM pipeline that remained essentially unchanged from the first ARM processor to the ARM7TDMI core.
- It is a classical fetch-decode-execute pipeline
- In the absence of pipeline hazards and memory accesses it completes one instruction per cycle.
- **Stages**:
  - **Instruction Fetch**: The first pipeline stage reads an instruction from memory and increments the value of the instruction address register, which stores the value of the next instruction to be fetched. This value is also stored in the PC register.
  - **Decode**: The next stage decodes the instruction and prepares control signals required to execute it.
  - **Execute**: The third stage does all the actual work: it reads operands from the register file, performs ALU operations, reads or writes memory, if necessary, and finally writes back modified register values.

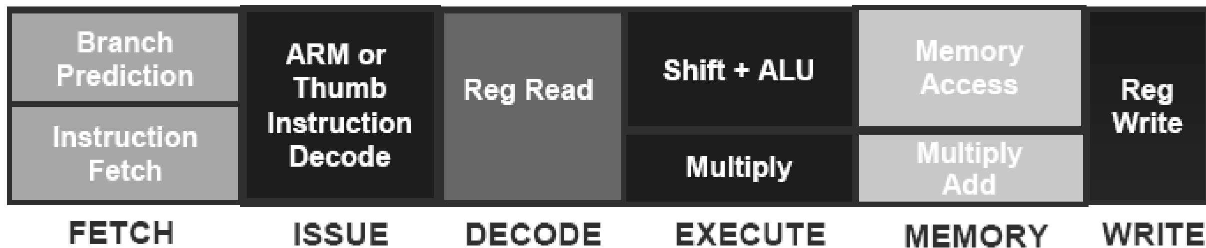
5-stage pipeline:



(Source: The ARM Architecture – http://www.eng.auburn.edu/~strouce/DaTseminar/UniPres07s.pdf)

- For pipeline structures with only one memory port, data transfer instructions would cause a pipeline stall, because the next instruction cannot be fetched while memory is being read or written.
- ARM9TDMI and later micro architectures solved the problem by using separate instruction and data caches. This allows modifying the pipeline to avoid stalls on data transfer instructions.
- First, to make the pipeline more balanced, the register read step was moved to the decode stage, since instruction decode stage was much shorter than the execute stage.
- Second, the execute stage was split into 3 stages. The first stage performs arithmetic computations, the second stage performs memory accesses (this stage remains idle when executing data processing instructions) and the third stage writes the results back to the register file. This results in a much better balanced pipeline, which can run at faster clock rate.
- But there is one new complication — the need to forward data among pipeline stages to resolve data dependencies between stages without stalling the pipeline.

6-stage pipeline:

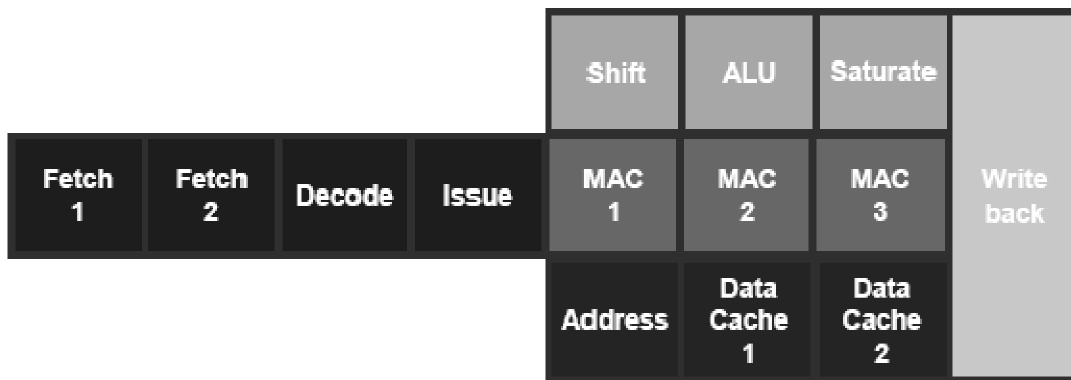


(Source: The ARM Architecture – http://www.eng.auburn.edu/~strouce/DaTseminar/UniPres07s.pdf)

- The ARM10 core made further improvements to the pipeline structure, to reduce the constraints caused by memory bandwidth.
- Therefore they made both instruction and data bus 64-bit wide.
- In the fetch stage, two instructions were fetched on each cycle, which enabled the introduction of a static branch prediction unit.
- The branch predictor tries to look ahead of the instruction stream and predicts backward pointing branches as 'taken' and forward pointing branches as 'not taken'. This eliminates the penalty of branches for loops that execute many times.
- In the execution stage, the 64 bit data bus improved the performance of multiple-register transfer instructions by transferring two registers at a time.
- A separate adder was introduced for multiply-accumulate instructions, instead of using the main ALU to do the addition.
- As multiply instructions do not read or write memory, this adder could be placed to the data stage, which led to a better balanced pipeline and enabled it to run at higher clock rate.

- The memory access stage became the longest running pipeline stage. Thus the clock frequency could not be increased further. To remove the problem another adder was introduced to compute the address.
- As address computation which involves just simple addition could complete in less than 1 cycle, one and a half cycle could be used for memory access.
- Lastly, instruction decoding was made a separate stage.

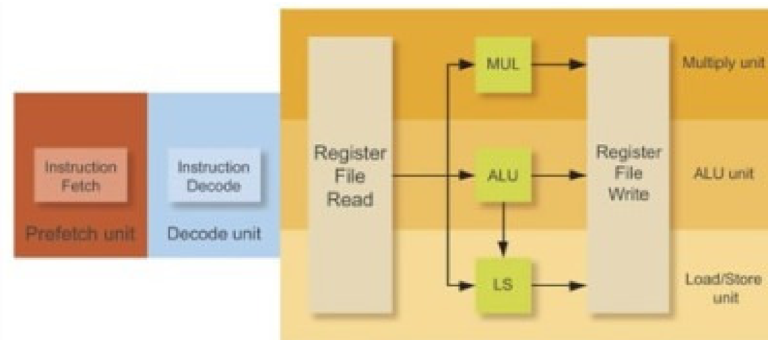
8-stage pipelines:



(Source: The ARM Architecture – http://www.eng.auburn.edu/~strouce/DaTseminar/UniPres07s.pdf)

- The ARM11 core introduced two main changes to the pipeline structure.
- Shift operation was separated into a separate pipeline stage.
- Instruction and data cache accesses were distributed across 2 pipeline stages.
- The instruction execution stage was thus split into three separate pipelines that could operate concurrently in some situations and could commit instructions out-of-order.
- The fetch and decode stages are executed in order.

AVR



AVR32 UC3 core pipeline structure (Reference: www.atmel.com)

The AVR32 UC core has a three-stage pipeline.

- Instruction Fetch stage
  The pipeline stage prefetches a 32-bit or two 16-bit instructions every clock cycle into an internal instruction buffer. The instruction fetch stage has been specially designed to optimize instruction fetch from on-chip Flash memory. The buffer ensures that the pipeline completely prevents pipeline stalls during sequential program execution. Execution from on-chip Flash can be sustained at the maximum CPU clock frequency without the CPU having to stall waiting for instructions from the Flash.
- Instruction Decode stage
  The second stage decodes instructions and generates necessary signals for instruction execution.
- Execution stage
  The third stage is made of three execution sub-units: the ALU, multiplication, and load/store units. The ALU performs arithmetical and logical operations, including hardware division. The multiply unit executes the numerous multiply and multiply-and-accumulate (MAC) operations available from the instruction set architecture (ISA), and the load/store unit performs single cycle memory accesses to SRAM or accesses on the high speed bus (HSB). There are no data hazards in the UC core so the register files can be updated during the same clock cycle as the instruction is executed. This makes assembly programming simpler compared to deeper pipelines as no code scheduling is needed.

MEMORY MAP AND MEMORY MANAGEMENT UNIT

ARM

- The *Memory Management Unit* (MMU) memory system architecture allows fine-grained control of a memory system.
- Most of the detailed control is provided through *translation tables* held in memory. Entries in these tables define the properties of memory areas
- The properties include:
  1. **Virtual to Physical Address Mapping:** An address generated by the ARM processor is called a *virtual address*. The MMU allows this address to be mapped to a different *physical address.* This physical address identifies which main memory location is being accessed. It can be used to allocate memory to different processes with potentially conflicting address maps, or to allow an application with a sparse address map to use a contiguous region of physical memory.
  2. **Memory access permissions:** These control whether a program has no access, read-only access or read/write access to the memory area. When an access is not permitted, a memory abort is signaled to the ARM processor.
  3. **Cachability and bufferability bits (C and B):** Transition tables are used to provide status information about memory aborts to the ARM. A full translation table lookup, known as a *translation table walk,* takes significant execution time cost. To reduce the average cost of a memory access, the results of translation table walks are cached in one or more structures known as *Translation Lookaside Buffers* (TLBs). Usually, there is a TLB for each memory interface of the ARM implementation:

     - a system with a single memory interface normally has a single unified TLB.

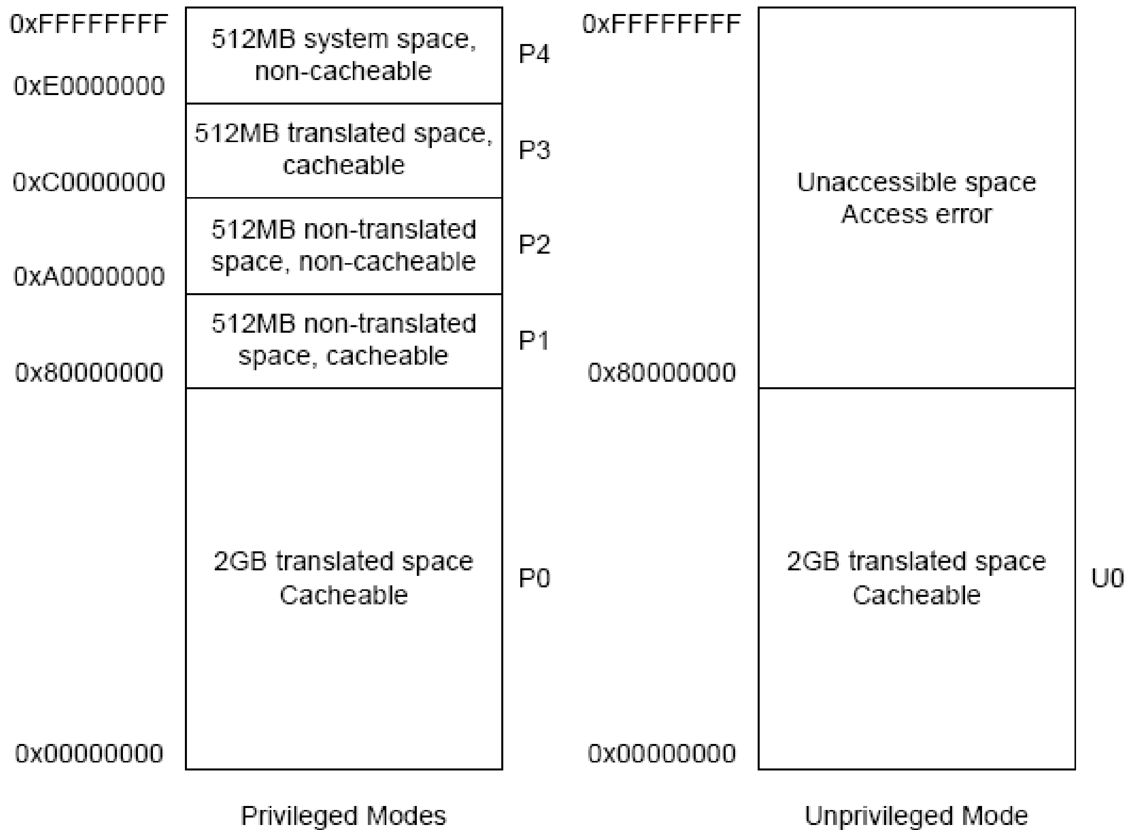
     - a system with separate instruction and data memory interfaces normally has separate instruction and data TLBs.

     In cached systems, there is normally one TLB per cache.
     When translation tables in memory are changed or a different translation table is selected, previously cached translation table walk results in TLBs cease to be valid. The MMU architecture thus supplies operations to flush TLBs. MMUs can be enabled and disabled.

AVR:

AVR32 Memory Map:

- The AVR32 architecture has a 32-bit virtual memory space.
- The virtual space is mapped into a 32-bit physical space by the MMU
  (Note: All implementations do not use caches)



(Fig. The AVR32 virtual memory space (Source: AVR datasheet)

- The memory map has six different segments, named P0 through P4, and U0.
- The P-segments are accessible in the privileged modes, while the U-segment is accessible in the unprivileged mode.
- Both P1 and P2 are segment translated by default to the physical address range by clearing the MSBs in the virtual address. Code for initialization of MMUs and exception vectors are located in these segments. (Note: P1 may be cached but P2 is always uncached).
- P3 space is also by default segment translated. But P3 becomes page translated after enabling and setting up the MMU. i.e. page translation overrides segment translation.
- P4 is intended for memory mapping the memory arrays in caches. This segment is non-cacheable and non-translated.
- U0 is accessible in unprivileged user mode. This segment is cacheable and translated depending on the configuration of the cache and MMU. If accesses to other memory addresses than the ones within U0 is made in application mode, an access error exception is issued.

The AVR32 architecture has two translations of addresses.

1. Segment translation
2. Page translation

Both these translations are performed by the MMU and they can be applied independent of each other, meaning you can enable:

1. No translation(virtual and physical addresses are the same)
2. Segment translation only
3. Page translation only
4. Both page and segment translations

The segment translation is turned on and the page translation is by default turned off after reset.

Memory Management Unit (MMU):

- The AVR32 architecture defines an optional Memory Management Unit. This allows efficient implementation of virtual memory and large memory spaces. Virtual memory simplifies execution of multiple processes and allows allocation of privileges to different sections of the memory space.
- The AVR32 MMU translates virtual address into physical addresses when memory access is performed. This information resides in a page table. The page table holds entries of each page, protection information and data needed for the translation process. The page table is accessed for every memory access, in order to read the mapping information for each page. A special page table cache called the Translation Lookaside Buffer (TLB) is used to speed up the translation process.
- The MMU also checks for access permissions.
- The MMU issues an exception, if an error occurs in the translation process or if Operating System intervention is needed for some reason, allowing the problem to be resolved by software.
- The MMU uses paging to map memory pages from the 32-bit virtual address space to a 32-bit physical address space. Page sizes of 1,4, 64KBytes and 1MByte are supported.
- Each page has individual access rights, providing fine protection granularity.

MODES OF OPERATION

ARM

There are seven modes of operation supported by ARM architecture

User	usr	Normal program execution mode
FIQ	fiq	Supports high speed data transfer or channel process
IRQ	irq	Used for general interrupt handling
Supervisor	svc	Protected mode for OS
Abort	abt	Implements virtual memory and/or memory protection
Undefined	und	Supports software emulation of hardware coprocessors
System	sys	Runs

Mode changes can be done by software or by external interrupts or exception processing. Most applications programs execute in User Mode. When in User mode, the program being executed is unable to access some protected system spaces or to change mode other than by causing an exception.

All modes except User (i.e. FIQ, IRQ, Supervisor, Abort, and Undefined exceptions) are privileged modes and hence have full access to system resources and can change system mode freely.

System mode on the other hand (present in architectures 4 and above) cannot be entered by an exception and has same registers available as the User mode. However it is a privileged mode and is not subject to restrictions of the User mode. It is intended for use by the Operating System tasks which access the system resources but wish to avoid using additional registers associated with the exception modes. Avoiding such use ensures that task state is not corrupted by occurrence of any exception.

AVR32

There are three modes of operations of ARM.

i. Normal RISC state

Under this mode, there are many execution contexts.

Priority	Mode	Security	Description
1	NMI	Privileged	NMI
2	Exception	Privileged	Execute exception
3	Interrupt 3	Privileged	General Purpose Interrupt
4	Interrupt 2	Privileged	General Purpose Interrupt
5	Interrupt 1	Privileged	General Purpose Interrupt
6	Interrupt 0	Privileged	General Purpose Interrupt
N/A	Supervisor	Privileged	Supervisor. Calls
N/A	Application	Unprivileged	Normal Execution

Mode changes can be made under software control or by external interrupts or exception processing. High priority modes have privilege of interrupting over low priority ones. When running an OS, running mode is ‘Application’. Upper halfwords of status registers and certain privileged instructions are curtailed from access. All other System modes have full privileges. Upon Reset, the processor enters Supervisor mode.

ii. Debug state

This state allows implementation of a software monitor routine that can allow changes to system information for development time debugging. Previlaged instructions are available. Interrupts are disabled but can be individually switched on by resetting their individual mask bits.

iii. Java State

Many AVR32 cores come with a Java Extension Module (JEM) which is capable of executing Java bytecodes. Java state is enabled when Normal RISC mode is suspended. Java state is not the same as the ‘system’ and ‘application’ modes. It can execute at different interrupt levels (although Java ISRs are known to be inefficient in terms of interrupt latency) without interfering with the mode settings. Performance is ensured using Java optimized RISC instructions.

Similarities and Differences:

- In terms of security, the AVR allows for more secure execution modes given that certain parts of the microcontroller are withheld from the user. It also allows for a more bit level access when debug mode is switched on.
- Java support in AVR is a welcome addition although Java usage in the Embedded Systems industry is low. It finds more acceptance in the PDA, Handhelds and mobiles related applications where the application gets much better support from Java enabled devices.

INSTRUCTION EXECUTION AND INTERRUPTS

ARM

ARM supports five levels of interrupts

a. Fast interrupt
b. Normal interrupt
c. Memory aborts (used for memory protection or virtual memory implementation)
d. Attempted execution of undefined instruction
e. Software interrupts which can make calls to an operating system.

When an exception occurs, CPU halts execution after completion of current instruction and starts execution of one of the exception vectors. An OS installs a handler for each exception. Privileged OS tasks execute in System mode within the OS to avoid state loss.

Exceptions are generated by internal and external sources to cause processor to handle an event. There are 7 types of exceptions in ARM

EXCEPTION	MODE
Reset	Supervisor
Undefined instructions	Undefined
Software interrupt (SWI)	Supervisor
Prefetch Abort	Abort
Data Abort	Abort

IRQ (interrupt)	IRQ
FIQ (fast interrupt)	FIQ

After Reset, the ARM processor begins execution from start in supervisor mode where interrupts are disabled. Undefined exception occurs when ARM executes a co-processor instruction and when no response from the co-processor is received. An SWI enters Supervisor (OS) function. A Prefetch Abort is initiated if the processor tries to execute an invalid instruction. Data Abort exception occurs before any instruction/exception have altered state of CPU.

Exception Priority

Priority	**No.**	**Exception**
Highest	1	Reset
	2	Data Abort
	3	FIQ
	4	IRQ
	5	Prefetch Abort
Lowest	6	Undefined instruction SWI

Undefined instruction and SWI cannot occur at the same time, as they each correspond to non-overlapping decoding and also have the lowest priority.

AVR

High priority internal MCU actions that halt the Normal RISC execution are called exceptions and that generated from external devices is called exceptions. The event processing system is designed in a way to handle multiple interrupts having different priorities like illegal opcode and external interrupt requests.

Event Handling in AVR32A

In the unmasked state, the pending event is accepted. All other events except non-maskable ones are masked. When a request is accepted, status register and program counter are stored on stack to ensure that core returns to the normal execution flow after interrupt completion. An exception handler of a dedicated address uniquely identifies the exception source. After the completion of the service routine, the program counter and status registers are popped off the stack and normal execution of the program continues.

Supervisor calls (privileged routines behave differently depending on the calling mode) also use the system stack. Debug requests are handled by the core by entering the debug mode. Upon entering the debug mode, hardware jumps to the Debug Exception Handler which executes in a debug context with dedicated return address register and return status register. These registers help improve the debugging because they don’t need to be stored on the stack.

Event Handling in AVR32B

In the AVR32B micro—architecture, the only difference in operation is that all the return information regarding the context of a routine is stored in dedicated return registers. Since registers are a lot faster than stack memory, it helps reduce interrupt latency. This micro-architecture has the biggest use in time critical applications where interrupt latency is an important factor.

Observations

In terms of exceptions according to the 'AVR32 Architecture' document, AVR32 has a better range of handlers for various exceptions. It also has a large number of priority levels which allows it to handle more events.

INSTRUCTION SET AND ADDRESSING MODES

ARM

ARM allows for two instruction sets, one which is ARM instruction set and the other is the Thumb instruction set.

ARM instructions can be classified into six classes

- Branch
- Data processing
- Status register transfer
- Load/Store
- Co-processor
- Exception Generating

Most data processing instructions update the 4 condition flags and are executed unconditionally. Branch instructions allow doe a 24-bit signed offset which allow for a forward/backward branch of up to 32MB.

Data processing instructions are of 4 types:

- Arithmetic/Logical instruction
- Comparison instructions
- Multiply instructions (of 32 bit Normal result or 64-bit Long result)
- Count leading zeros (CLZ) instructions which counts the number of zero bits to destination register of CLZ instruction

Co-processor information transfer is handled by Co-processor instructions and has three types, data processing, data transfer and register transfer.

Status register transfer instructions transfer contents of CPSR or SPSR to/from general purpose register. Writing to CPSR sets values of condition flags, interrupt bits and processor mode.

Load/Store instructions can load a 32 bit word, 16 bit half-word or an 8-bit byte from memory into a register. Load/Store multiple instructions perform a block transfer of any number of general purpose registers to/from memory.

Exception generating instructions are of two types:

- Software Interrupt (SWI) instructions cause software interrupt to occur and usually make calls to an OS defined service. It allows an unprivileged task to gain access to privileged resources by changing to privileged processor mode.
- Software Breakpoint instructions (BKPT) cause an abort exception to occur. With a suitable debugger attached to a abort vector, an abort exception is treated as a breakpoint preventing an exception from occurring.

The Thumb Instruction Set of ARM is half the size of regular instructions (16 bit as compared to 32), with a greater code density than usual. The only two limitations of the Thumb instruction set are:

- More instructions than usual are required which makes ARM Instructions better for time critical code.
- Exception handling code is not included.

Data types of ARM

ARM v4 and above support the following data types

Byte 8 bits
Halfword 16 bits
Word 32 bits

Addressing modes of ARM

5 modes of addressing are available in ARM:

- Data processing operands
- Load and Store word or unsigned byte
- Miscellaneous Loads and stores
- Load and store multiple
- Load and store co-processor

AVR:

The data format is as follows:

Format	Size	Extension
Byte	8 bit	.b
Halfword	16 bit	.h
Word	32 bit	.w

Double Word	64 bit	.d

Valid register pairs R1:R0, R3:R2, R5:R4, R7:R6, R9:R8, R11:R10 and R13:R12.

The AVR32 Instruction set has both compact and extended instructions. Compact instructions consist of 16bit length while extended instructions have an instruction length of 32 bits. Change of flow instructions such as branches, jumps, calls and returns may require the instruction buffer to be flushed

Addressing Modes in AVR

- Register Direct, with 1 and 2 registers
- I/O Direct
- Data Direct
- Data Indirect
- Code Memory Addressing

The Instruction set of AVR is roughly differentiated as:

- Math/Logical
- Branch
- Data transfer
- Bit and Bit test

The AVR32 has an orthogonal instruction set which allows registers in the register file to be used as pointers. The AVR32 has both compact and extended instructions. Compact instructions have a length of 16 bits whereas that extended instructions are of 32 bits. In normal instruction flow, instruction buffer is ensured to contain aligned and non-aligned extended/compact instructions that can be issued in a single cycle. Context change instructions like branches, jumps, calls and returns might require pipeline flushing

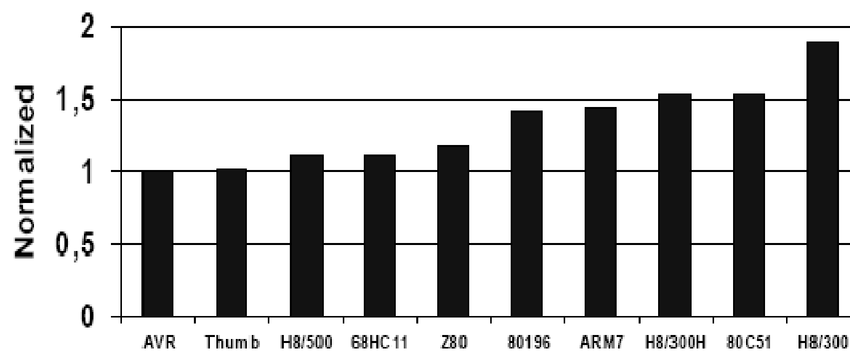
BENCHMARKS

Benchmark 1

- Atmel did a C code benchmarking to prove that they have a very High Level Language (C) suitable architecture. Real applications were collected from customers.
- Nine applications from different application areas were run.
- The AVR was compared with many of its competitors. Since not all code could be compiled for all compilers, all the indexes are relative to AVR. Only applications that could be compiled for both were used. The averaged ratios were taken
- Most of the compilers used were from IAR.
- Summary of Result:
    1. No single microcontroller was best for all the applications.
    2. AVR was in the top range for all applications

Normalized over all Benchmarks

- Averaged indexes from all applications
- All applications counts evenly



Benchmark2:

AVR32 was designed for extremely efficient code density and performance trial clock cycle. Atmel used the independent benchmark consortium EEMBC to benchmark the architecture with vaudeville compilers and consistently outperformed both ARMv5 16-bit (THUMB) code and ARMv5 32-bit (ARM) code by as much as 50% on code-size and **3X** on performance.

Performance Comparisons:

- More compact code increases the processor throughput and reduces power consumption by reducing the number of memory accesses.
- Atmel claims that their AVR32 UC core gives 20% better code density than ARM7(Thumb) and Cortex M3 (thumb2) – AVR32 UC code was consistently 5% to 20% smaller than code compiled for the ARM Thumb instruction set.
- They also say that when code was optimized for execution speed, AVR32 UC code was 30% to 50% more compact than the code compiled for the ARM instruction set.

Increasing Throughput through Architecture Enhancements:

Traditionally processor throughput was increased by increasing the processor clock frequency. But an increase in clock frequency leads to an increase in power consumption, which may be unacceptable for certain applications. An alternative is to try increase the amount of computation done per cycle. CPU

efficiency is affected by the number of cycles used to move data in and out of the processor, execution efficiency of individual instructions in the pipeline, branches and program code density.

Some of the steps that could improve computational throughput of the CPU include:

- Reducing the number of load store cycles.
- For Repetitive operations, such as multimedia, performing these operations on multiple data simultaneously (Single Instruction Multiple Data-SIMD) results in linear reduction in the number of cycles required to process the data stream.
- Allow unused pipeline resources to be used for non dependent calculations (out of order execution). Ex. During a divide operation which may take 32 cycles some non dependent calculations could be done.
- Improve the code density, especially for processors that rely on instruction cache for fast performance. Smaller the code, more the number of instructions that can be stored in cache. Also fewer will be the cache "misses" and fewer the fetches needed from external memory.
- Reducing the traffic on the main system bus can significantly reduce power consumption.

ABBREVIATIONS:

ARM	Acorn RISC Machine, later Advanced RISC Machine
ALU	Arithmetic Logic Unit
CISC	Complex Instruction Set Computer
CLZ	Count Leading Zeros
CPSR	Current Program Status Register
DSP	Digital Signal Processing
EEMBC	EDN Embedded Microprocessor Benchmarking Consortium
EX	Execute
GPR	General Purpose Register
HSB	High Speed Bus
ID	Instruction Decode
IF	Instruction Fetch
ISA	Instruction Set Architecture
ISR	Interrupt Service Routine
LDM	Load Multiple
MMU	Memory Management Unit
MCU	Microcontroller unit
MAC	Multiply and Accumulate
OS	Operating System
PC	Program Counter
PSR	Program Status Register
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
SPSR	Saved Program Status Register
SIMD	Single Instruction Multiple Data
SWI	Software Interrupt
SRAM	Static Random Access Memory
STM	Store Multiple
TLB	Translation Lookaside buffer

REFERENCES:

1. 'ARM Architecture Reference Manual' by ARM Limited, ©1996 – 2000
2. 'AVR32 Architecture' Document by Atmel Corporation
3. Wikipedia articles on ARM - http://en.wikipedia.org/wiki/ARM_architecture
4. Wikipedia article on AVR - http://en.wikipedia.org/wiki/Atmel_AVR
5. The Atmel Corporation website - http://www.atmel.com
6. www.embeddedrelated.com for user reviews on ARM and AVR
7. www.linuxdevices.com
8. 'Project Proposal document' for , Comparison between ARM and AVR architectures' by Rahul Sharma, Aby Kuruvilla, Chandrima Sarkar.
9. The AVR32UC Technical Reference Manual
10. The AVR32 Architecture Document
11. Programming and Customizing the AVR Microcontroller by Dhananjay V. Gadre (2001)
12. AT90S2313 datasheet
13. ARM System on a Chip Architecture by Steve Furber
14. http://en.wikipedia.org/wiki/Virtual_memory
15. Memory Management: http://pdos.csail.mit.edu/6.828/2006/readings/i386/c05.htm
16. Pipelining and Benchmarkshttp://www.atmel.com/products/avr32/uc3/UC_tech_brief.asp
17. Benchmark 1 http://www.atmel.com/dyn/resources/prod_documents/DOC1292.PDF
18. Benchmark 2: http://hup.hu/node/44892
19. The ARM Architecture presentation - http://www.eng.auburn.edu/~strouce/DaTseminar/UniPres07s.pdf
20. The ARM Architecture by Leonid Ryzhyk - http://www.cse.unsw.edu.au/~cs9244/06/seminars/08-leonidr.pdf
21. Atmel White Paper – MCU Architectures for Compute-Intensive Embedded Applications by Jo Uthus, Oyvind Strom- http://atmel.com/dyn/resources/prod_documents/doc4092.pdf
22. 'The ARM Architecture and Systems' by Dave Jaggar.