Comparing the Instruction Set Architectures of ARM and AVR

OVERVIEW

- *Pipelining:* Pipelines increase the speed at which a microprocessor executes operations by processing multiple instructions at the same time. Earlier ARM microprocessors, such as the ARM7, have 3-stage pipelines. The stages are fetch, decode and execute. Later ARM microprocessors tend to have more pipelines. In contrast, AVR has a 2-stage pipeline.
- *Architecture:* ARM is a Reduced Instruction Set Computer (RISC). AVR has the following typical RISC architecture features:
    - Has a large uniform register file.
    - Uses a load/store architecture (like AVR) where data operations only operate on register contents, rather than on memory contents.
    - Addresses are determined from register contents and instruction fields only.
    - Uniform and fixed-length instruction fields to simplify instruction decoding.[1]
- *Conditional execution:* Most instructions contain a 4-bit condition field, which contains flags that indicate whether to test for equality, non-equality or inequalities. The instruction only executes when the conditions are true.
- *Hardware support for power saving:* ARM processors have a number of hardware features that reduce power consumption.[2]
    - ARM processors use lower-speed transistors, which require less power.
    - Some ARM processors sleep the core until it receives an instruction.
    - The instruction set is minimal and extended using coprocessors, so ARM processors need fewer parts that drain power when unused.
- *Caching:* Caches are used in ARM memory to improve average performance. A cache is a block of high-speed memory locations whose addresses can be changed.
- *Hardware support for floating-point operations:* ARM processors provide hardware support for floating-point operations via the Vector Floating-Point (VFP) architecture. This is a co-processor extension that supports single- and double-precision floating-point arithmetic.

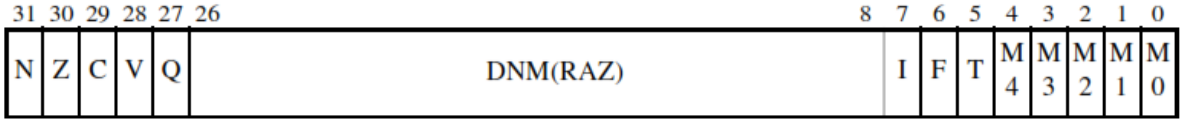
MEMORY MODELS

- *How many memory spaces are there in ARM?* ARM microprocessors see a single memory space of variable size depending on the particular chip. This may be divided into a number of different sections – for example, the Cortex-M3 has 8 sections in memory for code, SRAM, peripherals and so on.
- *Is there a separate memory space for code?* There is a dedicated section in memory for code, but it resides on the same flat memory space as all other data.

- *Are registers mapped to a memory space?* In contrast to AVR, ARM registers are not mapped to a memory space.
- *What is the maximum memory size for each memory space?* The maximum memory size for an ARM memory space is 4GB.[3]

REGISTERS

- *AVR has general and I/O registers. Does ARM have the same types of registers?* ARM has 16 general registers visible at any given time and 6 status registers but no I/O registers. Each exception handler has its own status register. These general registers are 32-bits wide (instead of 8-bit as in AVR) and are divided into banked and unbanked registers. Banked registers prevent program corruption when handling exceptions.
- *Describe the Current Program Status Register (CPSR).* The Current Program Status Register (CPSR) contains condition code flags, interrupt disable bits, the current processor mode, and other status and control information. It is accessible in all operating modes.



- *Explain how ARM uses the N, Z, C and V flags for overflow, signed comparison and unsigned comparison.* These flags are known as the condition flags. They store the result of arithmetic and logic instructions, which can be used to conditionally execute other instructions. See the section on conditional execution for more details.

Flag	Name	Description
N	Negative	Used in signed comparison, i.e. when the result is interpreted as a 2's complement signed integer. If the result is negative, this bit is set. If the result was positive or zero, this flag is cleared.
Z	Zero	Used in signed and unsigned comparison. If the result is zero, this bit is set.
C	Carry	Used in signed and unsigned comparison. For addition, this flag is set if the addition produced a carry (unsigned overflow). For subtraction, this flag is set
V	Overflow	Used in signed comparison. If signed overflow occurred, this flag is set.

- *The least significant 5 bits are reserved for operating modes. How do these modes work?* All the modes except for User are privileged modes that have full access to system resources and can change mode freely. All the modes except for User and System are exception modes that can only be entered when an exception occurs.

Mode	Description[4]
User	Normal program execution mode under which most tasks run. Unable to access protected system resources or change modes (unless an exception occurs).
FIQ	For a high-speed data transfer or channel process. Entered when a high priority (fast) interrupt is raised.
IRQ	For general-purpose interrupt handling. Entered when a low priority interrupt is raised.

Supervisor	A protected mode for the operating system. Entered on reset and when a software interrupt instruction is executed.
Abort	Implements virtual memory and/or memory protection. Used to handle memory access violations.
Undefined	Supports software emulation of hardware coprocessors. Used to handle undefined instructions.
System	Same registers available as User mode. Used when system tasks need access to system resources, but do not require additional registers associated with exception modes. This prevents task state corruption when an interrupt occurs.

- *Compare interrupt systems.*

AVR	ARM
No fast interrupts.	Fast interrupts mask normal interrupts.
Maskable interrupts can be disabled by setting I in Status Register (SREG).	Normal and fast interrupts can be disabled by setting the I or F bit respectively in the CPSR.
Interrupt Service Routine (ISR) must save and restore conflict registers and SREG.	Some registers and CPSR are saved and restored by an exception handler. Other registers are banked, that is, they are temporarily replaced with another register.
Interrupts are vectored, not polled.	Interrupts are vectored, not polled.

INSTRUCTION SET

- *Compare instruction structure including encoding scheme, number of operands, etc.* In ARM, each instruction is encoded into a 32-bit word. Almost all instructions have a 4-bit condition code field and most have two operands. Opcodes have variable lengths. Some ARM instructions allow you to shift or rotate operands before execution. Uniquely, immediate operands are represented as 8-bits with a 4-bit rotation.[5] Some ARM processors also use Thumb instructions, which may be 16- or 32-bits long.
- *Compare addressing modes.* Addressing modes in ARM are more complex than in AVR.

Addressing mode	AVR	ARM[6]
Register direct: register containing operand is in instruction.	Yes	Yes
Data direct: address of data containing operand is in instruction.	Yes	Yes
I/O direct: ARM uses memory-mapped I/O so I/O direct addressing is useless.	Yes	No
Immediate: operand is in instruction.	Yes	Yes
Register indirect: the address or location of operand is in a register.	No	Yes
– ***with displacement***: address is displaced.	No	Yes
– ***with pre-increment***: address is increased before memory access.	No	Yes
– ***with post-increment***: address is increased after memory access.	No	Yes
– ***with register indexed***: used for accessing specific indices of an array.	No	Yes
– ***indexed with scaling***: used for traversing data structures.	No	Yes
Data indirect: address of data is located in a pointer.	Yes	Yes
Indirect program addressing: program address is stored in a pointer.	Yes	No
Relative program addressing: program address found relative to current PC.	Yes	Yes

- *Describe how instructions are conditionally executed in ARM.* Conditional execution in AVR requires the use of branches (for example, `breq`) or bit testing (such as `sbic`). ARM

programmers can instead post-fix 'condition codes' to the end of their instructions (for example, add turns into addeq). The processor uses these condition codes to determine which condition flags in the CPSR to test (NZVC). If the condition was true, then the instruction is executed and vice versa.

- *Compare instructions for stack operations.*

AVR Stack	ARM Stack
Registers need to be pushed and popped individually.	Multiple registers can be pushed and popped in one instruction.
SP needs to be read, modified and then written back when you want to change it. This is because it is stored in an I/O register.	SP can be modified directly since it is stored in a general register (R13).

- *ARM provides multiple-bit shift instructions. How are they implemented in hardware?* In hardware, multiple-bit shifts are implemented using a barrel shifter. This allows the shifts to be performed in one clock cycle.[7]
- *AVR provides special instructions to access I/O registers. Does ARM also provide such instructions? If not, how do programmers access I/O ports in ARM?* AVR uses special instructions to access I/O registers since it utilises separate I/O. In contrast, ARM uses memory-mapped I/O so it does not provide I/O-specific instructions. Instead, programmers use normal load and store instructions with I/O memory addresses.
- *AVR provides a special instruction called sleep for saving power. Does ARM also have a similar instruction?* ARM processors can sleep using instructions from the Thumb-2 instruction set. This set is not available on all ARM processors.

Instruction	Name	Function
`wfi`	Wait for interrupt	Halts program until interrupt occurs
`wfe`	Wait for event	Halts program until a reset, exception or other event occurs
`sev`	Send event	Signals all CPUs of an event

- *AVR provides a watchdog timer to prevent software corruptions through the instruction* `wdr` *that resets the watchdog timer. Does ARM also provide a similar instruction?* Watchdog timers are usually not built into ARM processors, so ARM does not provide a similar instruction. Watchdogs can be attached to ARM processors as peripherals.

DATA TYPES

- *Compare all data types.*

Data Type	AVR Data Type Size	ARM Data Type Size
Words	16-bit	32-bit
Half-words	Not implemented	16-bit
Bytes	8-bit	8-bit

- *How would you implement data types such as 64-bit signed and unsigned integers that are not supported by ARM?* 64-bit signed and unsigned integers can be implemented using pairs of 32-bit words, which would represent the low and high word of the number. You

can then write functions such as addition and subtraction to operate on your new data type by using built-in ARM instructions.

REFLECTION

- *Reflect on similarities in each processor.*
  - Both architectures are RISC based so they have simplified instruction sets. Usually, this reduces the cost and power consumption of the microprocessor.
- *Reflect on differences in each processor.*
  - ARM processors have more pipelines than AVR, so it runs processing-intensive programs faster.
  - ARM is 32-bit while AVR is 8-bit. This allows ARM processors to provide greater performance for the same size and price. However, it requires more power.
  - AVR processors are cheap and ubiquitous. They do not require proprietary hardware, firmware or development tools, and benefit from free and standardised development environments.
  - AVR processors are easier to understand and use. They can be easily soldered compared to ARM processors. [8]

CONCLUSION

- *What kind of embedded systems benefit from AVR and ARM?* ARM has more powerful features and better development support, making it more suitable for computationally complex consumer electrical devices. Consequently, it is commonly found in smartphones, GPS devices and digital TVs. In addition to being very low-power, AVR is simpler to understand, cheaper and easier to solder. As a result, it is found in Arduinos as well as applications where the powerful features of ARM are unnecessary – for example, in cars, home appliances and lighting.

---

[1] https://www.scss.tcd.ie/~waldroj/3d1/arm_arm.pdf
[2] http://www.quora.com/What-makes-ARM-based-chips-relatively-power-efficient
[3] http://www.arm.linux.org.uk/developer/memory.txt
[4] http://simplemachines.it/doc/arm_inst.pdf
[5] http://alisdair.mcdiarmid.org/2014/01/12/arm-immediate-value-encoding.html
[6] ftp://www.cs.uregina.ca/pub/class/301/ARM-addressing/lecture.html
[7] http://stackoverflow.com/questions/9083743/is-bit-shifting-o1-or-on
[8] http://electronics.stackexchange.com/questions/1107/why-is-avr-used-in-arduino