COMP 2121 Assignment One: Comparing the ISA of ARM and AVR

1. Overview

ARM microprocessors are known for being very small and cheap, simple compared to most other general purpose processors, while still retaining adequate performance. They have a very simple hardware design with a very small die. Some of the more salient features of ARM microprocessors are:

- Low power consumption. Power consumption can be controlled easily, thanks to the small die and simple pipeline design.
- High modularity. Components such as caches, floating point and other co-processors are all optional. This means that developers can make highly specialised microprocessors by only including the minimum of components needed.
- Conditional execution. After doing a compare, rather than branching there are instructions such as ADDNE (add if not equal) that only execute if the previous compare satisfies the required condition. It allows for very compact code.
- Powerful memory transfer instructions. Included in the instruction set are instructions allowing you to load/store up to 16 registers at once. This allows for faster, more compact code involving memory access.

There are a number of differences between the AVR and ARM architectures, which can be divided broadly into the categories of Memory models, Registers, Instruction set, and Data types.

2. Memory models

ARM processors have a single memory map, with both code and data sections stored in a single memory space. This is also known as a 'Von Neumann' memory architecture. In ARM processors this space has a maximum size of (2^32 -1) bytes, about 4GB. It is a little-endian memory and addressed in four byte long words. That is, it is aligned on four byte boundaries, with the two least significant bits being zero.

Registers for I/O devices are mapped into this memory space as well, while the general registers are usually implemented in a co-processor. Some ARM Cores, such as the Cortex-M0/1/3, do not have a co-processor and instead do in fact need to implement the general registers in memory.

Along with this memory space, there are also a number of caches on the chip, depending on the type of ARM Core and the developer's customisations. Cache memory is small, very fast memory used to retain copies of recently used memory values. This enables faster access than it would if the processor was accessing the memory every time.

3. Registers

AVR has 32 8-bit general registers. ARM also has 32 general-purpose registers, but they are 32-bit registers and only 16 are available for use in user mode (One of which is the program counter). The other 16 are used for system level programming and exception handling, and only become available when the processor is running in the corresponding operating mode.

ARM memory-maps the I/O registers of any input/output devices, which appear like a memory location at a particular address. These registers are read and written using the same instructions as any other memory locations.

Finally there is also the Current Program Status Register, similar in use to the Status register (SREG) in AVR.

a. N, Z, C and V flags

The Current Program Status Register (CPSR) is used to store the condition code bits. The bits 31 to 28 are the N, Z, C and V flags and they are used as follows:

- N – Negative. This flag is set to the value of the top bit of the 32 bit result of an ALU operation. If this bit is a one, it means that the result was negative.
- Z – Zero. Set to one if every bit of an ALU operation is 0, i.e. the result was zero.
- C – Carry. Set to one if an ALU operation generates a carry-out. This can result from an arithmetic operation or from a shift.
- V – Overflow. Set when an ALU operation causes an overflow into the sign bit. It is essentially the signed equivalent of the carry flag and is set when the most significant bit is set or cleared.

b. Operating modes

The bits from 0 to 4 control the operating mode of the processor. When these bits are set they allow the processor to operate in modes other than the default user mode. These privileged operating modes are used to handle supervisor calls and exceptions, and enable use of system registers and the ‘Saved Program Status Register. The saved program status register saves the state of the CPSR so that when the user process is resumed, it can be fully restored.

The operating modes are:

- 10000 – User mode. This is the default, which allows the use of just the user registers.
- 10001 – FIQ. Processing fast interrupts – entered when an interrupt is received from the designated fast interrupt source. Enables use of 7 FIQ registers. There are seven reserved registers (more than any other operating state), to reduce the amount that needs to be saved, and thus speed up handling of fast interrupts.
- 10010 – IRQ. Processing Standard Interrupts. Any other interrupt signal besides those from the fast interrupt source causes the processer to enter IRQ. Enables use of 2 IRQ registers.
- 10011 – SVC. Entered when an interrupt is encountered in the software, often used to invoke OS services. Enables use of 2 SVC registers.
- 10111 – Abort. Memory faults cause the processor to enter Abort mode. Enables use of 2 abort registers.
- 11011 – Undef. Handling undefined instruction traps, when an instruction is received that is not recognised by the main core or coprocessors. Enables use of 2 undef registers.
- 11111 – System. Used by the OS to run privileged operating system tasks.

c. Interrupt systems

Bits 5-7 are the T, I, F bits – Instruction set and interrupt enables.

- T – Thumb. This field is used to switch between the instruction sets used, ARM or Thumb.
- I – Interrupt, when set to one enables normal interrupts.
- F – Fast interrupt, when set to one enables fast interrupts.

In ARM, interrupts are all handled in the same general way. First the state is saved, by copying the PC and CPSR. Then the processor operating mode is changed appropriately and

the PC is changed, usually branching to the exception handler. Upon return, saved registers are restored and the program resumes.

ARM's interrupt system is very similar to AVR's, the major difference lies in software interrupts. Software interrupts are used in ARM and a number of other processors to generate an interrupt from an instruction in the code. AVR does not support software interrupts and so external interrupts must be used to implement a software interrupt. AVR also does not handle exceptions, while ARM does handle them, similarly to the way it handles interrupts.

4. Instruction Set
a. Instruction structure

Some structural features of the ARM Instruction Set are:

- ARM instructions are all of a 3-address instruction format. This means that all operand registers and result registers are specified separately. For example, 'ADD Rr, Rc, Rd' puts the result of (Rc + Rd) in register Rr. In comparison, AVR uses a 2 operand instruction format, with nearly all instructions using 1 or 2 operands. Instead of specifying the result register separately, it is either preset (e.g. the MUL instruction always stores the result in r0) or in one of the operand registers (e.g. 'ADD Rr, Rd' instruction uses Rr as both an operand and the result register).
- All instructions, operands and results are 32 bits wide. AVR however has an 8-bit RISC architecture, with 8-bit registers, and most instructions have a fixed 16-bit length. Operands are usually 8-bit but can be 16 or more bits by using multiple registers to store them.
- Operands come from registers or are specified in the instruction itself, while the result is always stored in a register. To operate on anything in memory it must first be loaded into a register, operated on, and then stored in memory again. This is called 'Load-Store' Architecture. AVR also uses load-store memory access architecture.
- ARM also supports the Thumb instruction set. Thumb Instruction set is comprised of a number of 32 bit ARM instructions, compressed down to 16-bit opcodes. While slower than ARM code, it allows developers to save on memory and power.

b. Addressing modes.

The ARM data transfer instructions are all based around register-indirect addressing. This is where one register holds the memory address and another is used to load or store values into/from. AVR can be used with both direct and indirect addressing. Direct addressing uses memory addresses as operands to instructions, while indirect functions similarly to ARM, by using the X/Y/Z registers to store address pointers, and then the other registers to store data.

There are several different forms of addressing used in ARM that build off this form:

- Base plus offset addressing: An offset of up to 4Kb can be added to the memory address held in register. This allows you to loop through loading/storing a number of values, by increasing or decreasing the base by an offset each time. This is done with either pre-indexing (accessing the address + offset) or post indexing (accessing the address, then increasing the address by the offset). E.g. LDR r0, [r1, #4]! loads the value at r1 into r0, then increments the address in r1 by 4). AVR has indirect with displacement with the use of e.g. X + k, to access the address k bytes from X. Pre-decrement and post-increment are also supported (with –X, X+ respectively).
- Multiple register data transfer: Allows you to specify a data location and a number of registers and then load/store multiple values from first that memory address, and then consecutive word addresses, into these registers. AVR does not support this at all.

- Stack addressing: Uses a stack pointer to point to the top of the stack and then grows or shrinks as needed. Suffix's map onto the instructions to specify how the stack is to be stored and read from memory. (Full/empty, ascending/descending). The stack is implemented in AVR with the stack pointer SP, which points to the top of the stack.
- Block Copy Addressing: Uses the load/store multiple register instructions to move a block of data from one memory address to another. Not supported in AVR, instead loops can be used to do this.

c. Conditional Execution

In AVR, conditional execution is implemented using branches, which jump to another section of code if a condition is met, or skips, which skip a single instruction if the condition is met.

ARM, however, can apply conditional execution to all ARM instructions through the use of a two letter condition after the three letter opcode. So instead of branching to avoid code, after doing a compare, you can simply add, for example, 'NE' to the end of an instruction so it will only be executed if the compare was not equal. Similar two letter codes exist for a number of conditions.

There is a drawback in this, in that instead of a single check for the condition when branching, the program rechecks for every instruction that has a condition, adding a clock cycle to execute. This slows down the program when there are more than three conditional instructions (it taking about three cycles to execute a branch). This slight disadvantage aside, it still allows for faster code every time there are three or less instructions in the conditional sequence, and can allow for very compact and efficient code when used well.

d. Stack operations.

AVR implements the stack as a block of bytes in SRAM, growing downwards from a high memory address. The stack pointer (SP) is used to point to the next available address and the instructions PUSH and POP are used to load/store bytes from the stack one at a time. E.g. 'push Rr' stores the value in Rr to the address in SP, then decrements SP by one byte to the next available memory location.

Comparatively, ARM offers a more flexible set of stack operations through the combination of multiple register transfers and two letter postfixes to specify the stack address mode. Four types of stack are supported: Full ascending, empty ascending, full descending and empty descending. 'Full' signifies that the stack pointer points to the last valid item, while for an 'Empty' stack it points to the next available address. Ascending/descending specifies the direction it grows in memory.

Register 13 is usually used as the stack pointer, and the load multiple/store multiple commands are used to 'push' and 'pop' a number of registers onto/from the stack at once, making for much more compact code than in AVR. For example, the instruction 'STMFD r13, {r2-r9}' stores registers 2-9 onto the stack (using stack pointer in register 13), in the 'Full Descending' address mode, signified by the FD postfix.

ARM offers a compact yet very versatile stack implementation, while AVR uses a much simpler, 2-instruction method.

e. Multiple bit-shift instructions

The ARM pipeline architecture includes a barrel shifter, which operands pass through and are shifted if necessary. The barrel shifter operates with a 32x32 cross-bar switch matrix in order to shift bits. This is illustrated below with a smaller 4x4 matrix.

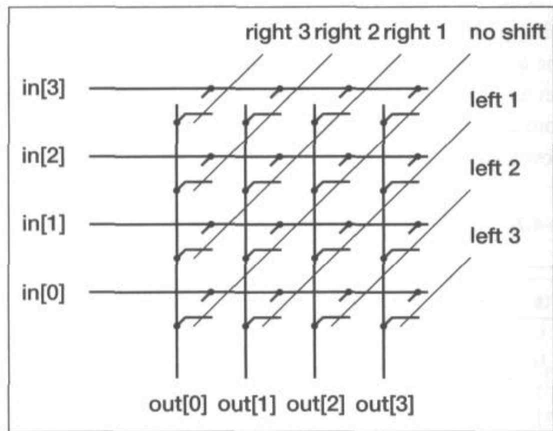

Figure 4.15, Steve Furber, ARM System-on-Chip Architecture, 2nd Edition, pp 92

- When implementing a left or right shift, the corresponding row of diagonals is turned on, in order to connect all the input bits to their destination output bit. Some bits are lost in these shifts, and just become a zero in the output as all outputs are pre-charged to a logic zero.
- For a rotate right, the right shift diagonal is enabled along with the complementary left (32 - right shift) diagonal.
- Arithmetic shift right uses zeros or ones for the unconnected bits depending on the sign of the operand. These are output correctly through separate logic to the rest of the shift.

f. I/O ports

AVR uses the *in/out* instructions to access I/O registers. This is because AVR usually implements these registers in a separate space to memory. This approach frees up memory space for other uses and is faster to access as there is a smaller address space to decode.

ARM, however, only implements a Memory Mapped I/O. This means that all registers in peripheral devices appear in the memory map. Rather than special instructions, these registers are read and written using the same instructions as any other memory location. This is a simpler approach for CPU design and allows for a smaller instruction set.

AVR is more flexible in this area than ARM, as it supports both separate and memory mapped implementation of I/O ports where required, while ARM supports only memory mapped I/O.

g. Power saving instructions

The command *sleep* in AVR is used to save on power by powering down the processor, effectively halting it altogether until the next interrupt. ARM processors have a similar instruction, 'Wait for Interrupt'. *WFI* is equivalent to AVR's *sleep*¸ it makes the processer suspend execution until it receives an IRQ or FIQ interrupt, or a debug entry request is made to the processor.

h. Watchdog timer

The watchdog timer in AVR can be reset with the *wdr* command. ARM provides no similar command and there is no easy way to reset the watchdog timer.

5. Data types

AVR uses only 8-bit registers, and supports 8-bit signed and unsigned integers. This means that any data types larger than 8-bits must be manipulated using multiple registers, which can be complicated to implement.

ARM processors support signed and unsigned 8-bit bytes, 16-bit half words, and 32-bit words. 64-bit integers can however be implemented by using two registers to contain the value. Operations such as addition can be performed using the carry bit, and there is a multiply-accumulate instruction to multiply two 32-bit numbers and get a 64-bit result. There is also support for floating point numbers, with the addition of a Floating Point coprocessor.

ARM processors support a much larger range of data types, however they also take up much more room in memory, so an AVR processor can be more memory-efficient when only small data types are needed.

6. Conclusion

The ARM processor is powerful, yet low cost and low power. It is quite versatile, as a 32-bit processor, and has a number of features, such as conditional branching and multiple memory transfer instructions, that allow for compact, fast code. ARM processor's low cost and respectable power means they are useful for small, mobile applications which require decent processing power while still saving on battery. Applications have included iPhones, iPods, and a large number of mobile phones and simple computers.

The AVR processor, in comparison, is an 8-bit processor and has less processing power than the ARM processor family. AVR has much less processing power, and is suited for very small and simple systems with a single purpose. AVR processors have been used in applications such as automotive systems in cars, simple motors, and USB based AVR's have been used in Microsoft Xbox controllers.

References

http://www.cs.umd.edu/~meesh/cmsc411/website/proj01/arm/home.html, Computer Science faculty, University of Maryland, Date of publish unknown.

http://en.wikipedia.org/wiki/ARM_architecture, and http://en.wikipedia.org/wiki/Atmel_AVR, Wikipedia, 2011.

www.ee.ic.ac.uk/t.clarke/arch/lectures/Part1.ppt, Department of Electrical and Electronic Engineering, Imperial College of London, Date of publish unknown.

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0360e/CHDEEIJB.html, ARM, 2010.

http://www.atmel.com/dyn/resources/prod_documents/doc0856.pdf, Atmel, 2010.

Leonid Ryzhyk, *'The ARM Architecture'*, June 5 2006

Steve Furber, *ARM System-on-Chip Architecture*, Addison-Wesley, 2nd edition, 2000.

Annie Guo, *COMP2121 Lecture notes: Weeks 1-5*, UNSW Computer Science and Engineering Faculty, 2011.