Proceedings,
The 2014 ASEE North Midwest Section Conference, ASEE-NWMSC2014-1C3
October 16-17, 2014, Iowa City, IA.

First Experiences with the AVR ATmega32 Microcontroller

Christopher R. Carroll
University of Minnesota Duluth
ccarroll@d.umn.edu

Abstract

In the fall of 2013, the Electrical Engineering department at the University of Minnesota Duluth reconfigured its microcontroller instructional laboratory to use the AVR ATmega32 microcontroller from Atmel for the first time. This change was prompted by pressure from students, who displayed considerable interest in the AVR family of processors due to its use in popular Arduino microcomputer systems. In response to this expressed interest from students, the microcontroller lab was redesigned around this new processor.

Atmel's ATmega32 is an 8-bit RISC processor, based on Harvard architecture. This new hardware replaced systems using Freescale's S12 processor, which is a 16-bit CISC processor, based on Princeton architecture. Thus, this change was a fundamental shift on at least three axes of computer characteristics. Was this change a gain or a loss on each of those three axes? Experience this year with teaching the lab using the ATmega32 has been generally positive, and this paper reports that experience in making the switch from the S12 to the ATmega32.

Despite the fundamental architectural differences between the former processor (S12) and the new processor (ATmega32), there are many similarities between the processors as well. The hardware features of the ATmega32 mimic the S12's features almost exactly, although the ATmega32 generally contains less of each feature, such as memory, I/O ports, timing features, etc. Many of the resources of the S12 were unused and wasted in lab exercises in the past. The scaled-back ATmega32, with basically the same features but in less abundance, is a better match to the instructional needs of this lab. Students feel they are getting a more complete exposure to the ATmega32, since nearly all the resources of the processor have been used in lab exercises, whereas many of the resources of the S12 were ignored in lab assignments because they exceeded the needs of the lab.

This paper will address the adaptations that were made in the structure and pedagogy of the microcontroller course to implement the change in processors from the S12 to the ATmega32. It will document some of the troubles and surprises encountered along the way, and should provide some guidance for others contemplating such a change in their microcontroller labs.

Setting

The "Microprocessor Systems" course at the University of Minnesota Duluth is a sophomore-level required course in the Electrical Engineering program intended to teach students to program in assembly language. It uses as prerequisite a first course in digital circuit design, basically just to acquaint students with digital terminology and with the logical functions AND, OR, NOT, and their derivatives. Over the years the course has used a variety of microprocessors and microcontrollers as foundation for the laboratory, to provide specific examples of the topics

discussed in class. Most recently, the course has used the S12 microcontroller from Freescale as the vehicle for lab experiments. Due to the popularity of the Arduino microcomputer systems for senior projects, students asked to change the processor used to the ATmega32, a member of the Atmel AVR family of processors, as used in the Arduino. In response to that student request, the course did adopt the new processor in the fall of 2013, replacing the S12.

The ATmega32 is architecturally different from the S12 in at least three important ways. Firstly, the ATmega32 is an 8-bit processor, meaning that its instructions operate primarily on byte-sized data, whereas the S12 is a 16-bit processor whose instructions operate primarily on 16 bits at a time. Secondly, the ATmega32 uses a Harvard architecture, meaning that it requires separate memory for storing program instructions, distinct from the memory used to store data, whereas the S12 uses a Princeton architecture processor that combines instructions and data in a single memory. Finally, The ATmega32 is clearly a Reduced Instruction Set Computer (RISC), meaning that its instructions each perform streamlined, minimal tasks, whereas the S12 is clearly a Complex Instruction Set Computer (CISC) executing instructions that individually perform complicated tasks. Despite these three fundamental differences, however, the ATmega32 and the S12 are remarkably similar in their hardware resources available to the programmer.

8-bit vs. 16-bit

Switching to the ATmega32 from the S12 involved changing from the S12's 16-bit architecture to the ATmega32's 8-bit architecture. At first glance, this seems to be a step backwards, since in considering computer performance, the more bits operated upon at once, the better. Shouldn't the change in processor have moved in the direction of 32-bit or even larger operand sizes, rather than changing from 16 bits to 8 bits? However, consider the goal of this "Microprocessor Systems" course. The goal is not to maximize processing power, or achieve maximum resolution in some mathematical calculation. The goal is to learn to program in assembly language, and most applications that these students will face will involve small scale situations. Variables will rarely need to exceed values that can be stored in just 8 bits, and using more bits just wastes hardware and causes additional confusion. Using an 8-bit processor is actually a pedagogical advantage in this class because high performance is not a goal of the course.

In the past when using the 16-bit S12 processor, probably the most confusing task that students confronted was keeping track of which registers in the S12 were 16-bit registers, and which were 8-bit registers. The difference is crucial, of course, but students did not appreciate the importance of distinguishing them, probably due to students' experience in programming using high-level languages where the storage size of operands is rarely considered. By contrast, the 8-bit ATmega32 has only 8-bit registers used for data, avoiding the confusion altogether. A few specific instructions do use 16-bit operands for special applications, but those instructions are easily distinguished so that the level of confusion on that front is markedly reduced.

Harvard vs. Princeton Architecture

The ATmega32 microcontroller uses Harvard architecture, meaning that storage for instructions that make up the program and storage for data operated upon by that program use physically separate memory structures. The "program memory" in the ATmega32 is 32K bytes in size,

actually organized as 16K 16-bit words. The "data memory" in the ATmega32 is 2K bytes of RAM plus 32 8-bit general-purpose registers plus 64 8-bit input/output registers, all byte-organized. Harvard architecture proponents tout improved performance because the program memory and data memory are separate and can be accessed independently, allowing for parallel simultaneous operations on the two memories. However, using a separate memory to store the program being executed restricts certain types of applications. In particular, "self-modifying" programs where the operation of the program actually changes the instructions in the program being executed, are generally not possible with Harvard architecture computers. Such "tricky" programming techniques are generally not a good idea anyway, but related situations are awkward to implement. For example, writing a program that itself allows development of and debugging a different program in memory, generally is not easy. Instead, one must rely on software tools provided by the microcontroller manufacturer to download programs into program memory and manipulate them there. Fortunately, Atmel does provide good software tools for program development and downloading into program memory, which avoids this problem.

By contrast, the S12 uses a Princeton architecture in which a single memory structure holds both the instructions that constitute the program and the data being manipulated. The S12 as previously used in this class contained 12K bytes of storage, and it was up to the programmer to keep data in a separate area of memory from the program. The fact that there is no difference in memory used to store instructions and that used to store data means that programs can freely manipulate instructions as needed, so that program development environments can be created easily. However, it is also easy to unintentionally modify instructions when a program misbehaves, or when some event such as a stack overflow occurs, so in Princeton architectures the program is often accidently corrupted by inexpert programmers.

RISC vs. CISC

The question of whether RISC or CISC architecture is the better choice for computer design is an ongoing and unresolved debate. The ATmega32 is a clear example of RISC design. The S12 is a clear example of the CISC approach.

In a RISC design, instructions executed by the processor are streamlined to the point that each instruction actually does very little. Large tasks are accomplished by using many individual instructions to get the job done. Programs in RISC processors therefore tend to be longer and use more program memory because more of these minimal instructions are needed to accomplish the task. RISC proponents acknowledge that their programs are longer, but they argue that since each instruction does very little, the instructions can be blazingly fast, so that even though more instructions are needed, the overall performance is better because each instruction is so fast.

In a CISC design, instructions are specialized to perform specific tasks well. Each instruction may perform several steps of processing, so that fewer instructions are needed to complete a given task. However, because the instructions tend to be complicated, they must execute more slowly. CISC proponents acknowledge that their instructions execute more slowly, but argue that because they need fewer instructions (and thus less memory) in the program for a given task, the overall performance is better.

Who wins, RISC or CISC? There is no clear answer. In an environment such as this class, RISC has the advantage that the instruction set of the processor is simpler and easier to explain, with fewer special cases to address. However, programs tend to be shorter with a CISC processor.

Similarities

Despite the architectural differences between the ATmega32 and the S12 detailed above, the two processors are actually remarkably similar in capabilities. Both include virtually the same collection of input/output devices and I/O resources. Both processors are members of large families of processors with varying resources, so to be specific, comparisons are made here with the particular family members that were used in the lab, the MC9S12DP256 (S12 for short) and the ATmega32. Generally the MC9S12DP256 and the ATmega32 have identical resources, but the ATmega32 has fewer of each type of resource. For example, compare internal memory:

Feature	MC9S12DP256	ATmega32
Static RAM	12K bytes	2K bytes
EEPROM	4K bytes	1K bytes
Flash memory	256K bytes	32K bytes
Internal input/output	1K bytes	64 bytes

As can be seen above, the two processors contain the same types of memory and I/O access. The S12 has much more than needed in instructional lab exercises. The ATmega32 is a better fit.

Comparing input/output resources, again the two processors include virtually the same types of devices. The S12 again includes much more than needed in instructional lab exercises:

Feature	MC9S12DP256	ATmega32
Parallel ports	ten: 4-, 7-, and 8-bit	four: all 8-bit
Timer	one	three
Input Capture	eight channels	one channel
Output Compare	eight channels	four channels
Pulse Width Modulation	eight channels	three channels
Asynchronous Serial I/O	two systems	one system
Serial Peripheral Interface	three systems	one system
Inter-Integrated Circuit I/O	one system	one system
Controller Area Network	five systems	--
Analog to Digital Conversion	sixteen analog inputs	eight analog inputs

Overall, the ATmega32 provides students with examples of the same resources that the S12 provides, so that lab exercises that were developed on the S12 were easily adapted to use the ATmega32 processor. Many of the resources of the S12 were ignored and wasted in lab.

Surprises

Using the ATmega32 for the first time as the foundation for this class revealed two surprising defects in its design that should have been corrected during the processor's design review process. Nevertheless, somehow these bugs survived the review, and must be tolerated.

The ATmega32 is a "little endian" processor. This means that when a mult-byte value is stored in consecutive locations in memory, the least significant byte of the value is stored at the lowest address. This is the sensible byte order for those who grew up with Intel processors, because Intel processors are all "little endian." The S12 is a "big endian" processor, meaning that multi-byte values are stored in the opposite order in memory. This is the sensible byte order for those who grew up with Motorola processors. Either byte order works fine, and programmers can learn to live with whichever order is supported by their processor. The ATmega32 is "little endian" … MOSTLY! For some unexplained reason, return addresses that are pushed on the stack by procedure calls and interrupt events are pushed in "big endian" order! Usually the programmer does not care, and the order is unimportant, BUT in those rare situations where the program must access and/or manipulate the return address on the stack, the programmer must remember this weird anomaly in the ATmega32 and realize that those return addresses are on the stack in "big endian" order.

The other defect in the ATmega32 design also has to do with stack addressing. When data is "pushed" onto the stack, the data is stored at the location identified by the stack pointer register, and then the stack pointer is decremented. This is known as "post-decrement" addressing because the register value changes after it is used. When data is "popped" off the stack the stack pointer is first incremented and then data is read from the location identified by the stack pointer. This is known as "pre-increment" addressing because the register value changes before it is used. So far, all is fine, and the stack works without a problem using this strategy of "post-decrement" and "pre-increment" addressing. However, it is also possible to address data memory using other registers that can be incremented or decremented as part of the access. Unfortunately, the choices available are "pre-decrement" and "post-increment" only, the opposite of how accesses with the stack pointer work! This is a stupid defect in the ATmega32 design, and means that if the programmer wants to, for example, implement a second stack using one of these other registers as the "stack pointer," the "regular" stack and the "created" stack will work differently.

Results

The switch from the S12 processor to the ATmega32 processor in the "Microprocessor Systems" course has been completed successfully. Despite fundamental differences in the architectures of the two processors, the lab provides the same experiences to students that were provided before the switch. Because the two processors share nearly identical input/output resources, existing S12 lab experiments required only minor changes to work with the ATmega32. Several advantages to using the ATmega32 have been noted:

- 8-bit data size is a better match to application needs, and less confusing
- RISC instruction set is easier to describe, with fewer special cases to consider
- Harvard architecture minimizes program corruption when programs misbehave

Also, several disadvantages have been noted:

- 8-bit data size seems outdated to students
- CISC instruction set provides more opportunities for creative programming
- Princeton architecture makes system program development easier

Has this been a productive change? Some say yes, some say no. Regardless, it is a change, and change helps keep the course content current and relevant. Change is usually for the good!

Bibliography

1. ATmega32 User Manual, Atmel Corporation document 2503Q-AVR-02/11, 2011.

2. Carroll, C. R., "Converting a Microcontroller Lab From The Freescale S12 to the Atmel ATmega32 Processor," *Proceedings of the 2013 ASEE North Midwest Section Meeting*, Fargo, ND, 2013.

3. Margush, Timothy S., Some Assembly Required: Assembly Language Programming with the AVR Microcontroller, CRC Press, New York, NY, 2012.

4. Pack, Daniel J. and Steven F. Barrett, Microcontroller Theory and Applications: HC12 & S12, Pearson/Prentice Hall, Upper Saddle River, NJ, 2008.