WHERE IS MY PROGRAM?

- Does my microcontroller run C language code?
- Where does it store its instructions?
- How does the microcontroller know where to begin?
- What happens during a reset?
- How does a computer *program* a microcontroller
- Are there going to be burgers in today’s class?

MICROCONTROLLER PROGRAMMING

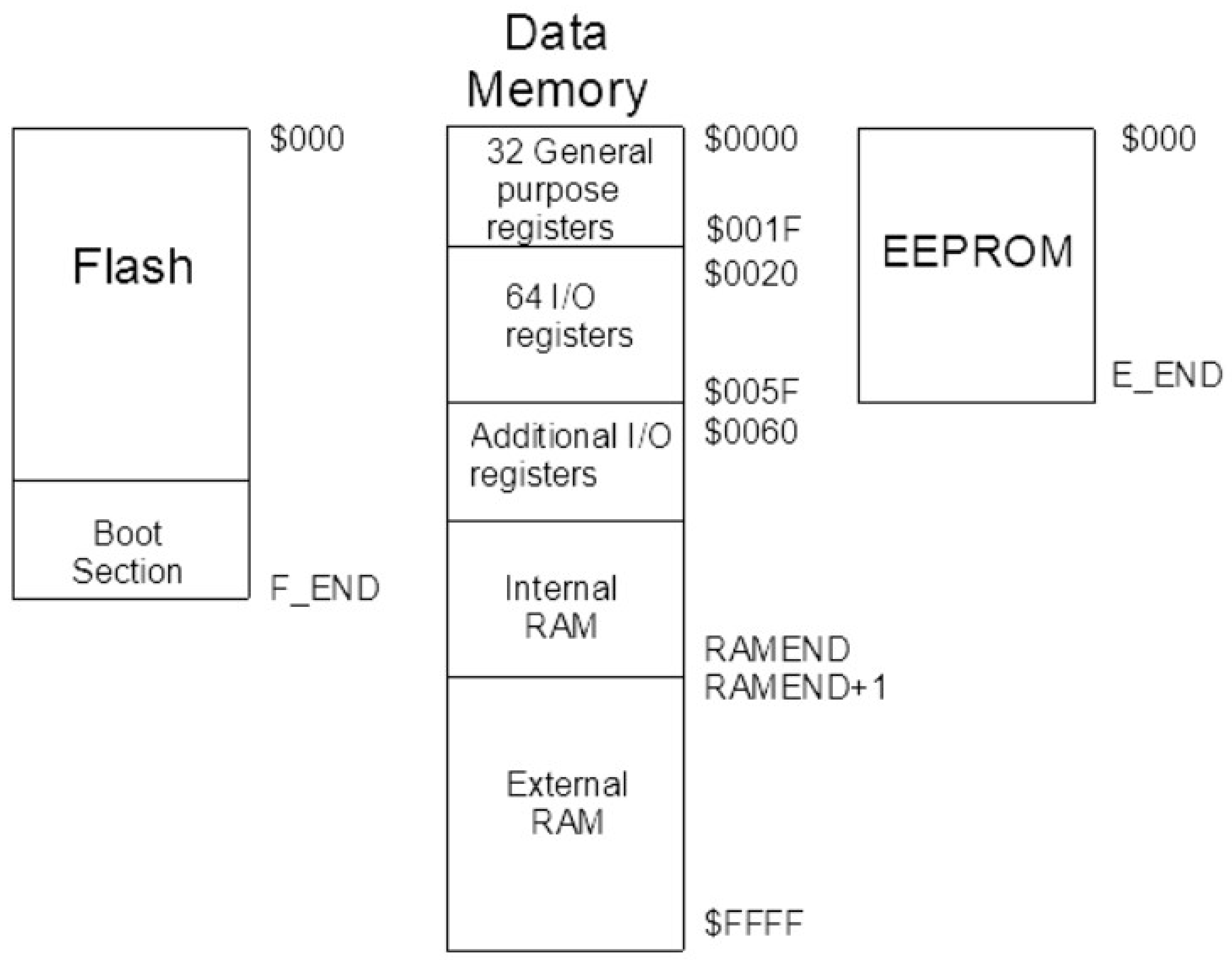
Daniel Martínez

I7266 - Programación de Sistemas Embebidos

CUCEI - UDG

COMPILER

AVR MEMORY MAP



Data Memory
Flash
Boot Section
$000
F_END
32 General purpose registers
$0000
$001F
64 I/O registers
$0020
$005F
Additional I/O registers
$0060
Internal RAM
RAMEND
RAMEND+1
External RAM
$FFFF
EEPROM
$000
E_END

I/O PORTS AND REGISTERS

HOW DO WE TALK TO THE EXTERIOR?

- Microcontrollers have input/output pins that can be used to receive and send signals with other devices.
- In AVR microcontrollers, we usually have *ports*, which are controlled by *i/o registers*, for example, in the Atmega328p there are four ports, namely PORTA, PORTB, PORTC and PORTD, and the behaviour and data in these ports is configured and read via the DDRx, PORTx and PINx registers.

I/O PORTS BASIC FUNCTIONALITIES

- We will employ the ports in the following manner:
- To set the direction of a port, we will use the Data Direction Register (DDRx).
  - If DDxn is written logic one, Pxn is configured as an output pin, if DDxn is written logic zero, Pxn is configured as an input pin.
- Remember, in this notation DDxn can be read as *the bit "n" in the Data Direction Register for port "x"*.
- This all means writing 00000001 binary to DDRx will enable bit 0 of port "x" as output.

I/O PORTS BASIC FUNCTIONALITIES

- Example 1:
- I look at the datasheet and I choose pin PC5 to output a digital signal.
- From the name of the pin I know that I need to search for the register DDRC and I need to write a logic one into the bit 5.
- In binary, a one in the bit 5 can be set in different equivalent ways. Let us explore some of them.

I/O PORTS BASIC FUNCTIONALITIES

- The objective is to send the binary value «00100000» to the DDRC register.
- We should not write that value directly to the register, this is, we cannot do *DDRC = 00100000* since that number would be interpreted as decimal.
- It is possible, however, to indicate that it is a binary number appending the suffix “0b”, e.g.: 0b00100000. Note that this is not recommended since it is out of the standard.

I/O PORTS BASIC FUNCTIONALITIES

- Since it is not recommended to use «0b00010000», we can use other methods to write that value to the DDRC register, the first one is using its decimal or hexadecimal value:
  - In decimal, that value represents 16. So, *DDRC = 16* is good.
  - In hex, that value represents 0x10. So, *DDRC = 0x10* is good too.
- Another way is doing a bit shift, if we want to put a *1* in the 5th bit of a byte, we can do *1 ≪ 5*.
- Indeed, DDRC = 1 ≪ 5 would work just fine.

I/O PORTS BASIC FUNCTIONALITIES

- The aforementioned methods would do the job, but those are not the safest methods.
  - First, it is a good practice that instead of assigning a logic one to a bit, we OR-operate it so the rest of the bits are not affected.
  - Second, while using operators in C, we make use of parenthesis to make clearer the intention of the operator employed.
  - Third, we can use macros and constants to make things clearer.
- Here are some real life examples of this.

I/O PORTS BASIC FUNCTIONALITIES

- DDRC |= (1 << PORTC5);
  - PORTC5 is a preprocessor constant equivalent to *5*.
- DDRC |= _BV(PORTC5);
  - The macro *_BV* is a preprocessor function that does the shift during preprocessing, making things faster.

I/O PORTS BASIC FUNCTIONALITIES

- Example 2:
- I look at the datasheet and I choose pin PB1 to read a digital signal.
- From the name of the pin I know that I need to search for the register DDRB and I need to write a logic one zero the bit 1.
  - If I want to set a zero, this time I need to AND-operate the value in order to preserve the other bits unaltered, the value that we will use will be referred as *mask* from now on.

I/O PORTS BASIC FUNCTIONALITIES

- Our mask needs to be the opposite of «00000001», this means «11111110», however, if we try to OR-operate that with a register, we would be setting all the other bits to 1. This is why we need to AND-operate with it.

I/O PORTS BASIC FUNCTIONALITIES

- `DDRB &= ~(1 << PORTB1);`
- `DDRB &= ~(_BV(PORTB1));`

EXERCISE 1

PORT MANIPULATION

- Gather in groups.
- Every group has to figure out how to make the following pattern:
- The pattern must repeat indefinitely.
- You can draw the algorithm, write it in pseudocode or try to implement it in C.

THE COMPILER

Est-ce que tu comprends ?

WHAT IS A COMPILER

WHAT IS A COMPILER?

- A compiler is a computer program that *translates* source code from a high level programming language into a low level programming language. It does its job in multiple steps.
- In this lecture we will analize tools related to the GNU Compiler Collection for AVR devices.
- We will use avr-gcc to compile our code, and it does almost everything we want with a single command!

WHAT IS A COMPILER?

- To understand what the command does we have to read the documentation, however, that will take a lot of time, so we are going to get our hands dirty and learn in the process.
- I have a program, the one that turns an LED on. I am going to compile it with the following command:
  - **avr-gcc main.c -mmcu=atmega328p**
- What is the output? What is next? Let us see…

AVR-GCC

AVR-GCC

- avr-gcc is the program in charge of compiling our source code, we need to provide options to compile and a file to compile.

  (show **avr-gcc –help** output)

- If we only provide a file, we will not be able to compile successfully

  (show **avr-gcc main.c** output)

- If we provide a file and the option for the microcontroller we are using, then we get a file called **a.out** as output.

  (show **avr-gcc -mmcu=atmega328p main.c** output)

AVR-GCC

- We can indicate where we want out output.
- (show **avr-gcc -mmcu=atmega328p main.c -o main.elf** output)
- That is the usual way to employ avr-gcc, but we can also run it so it shows us the intermediate steps. The first step is called *preprocessing*.

THE PREPROCESSOR

THE PREPROCESSOR

- Preprocessing is one of the first steps to compile software, it takes certain necessary steps for the compiler to understand what we want to compile.
- The C preprocessor performs these two main actions:
  - Remove comments
  - Substitue preprocessor directives:
    - Adding includes, substituting constants, expand macros and pragmas

THE PREPROCESSOR

- `avr-gcc -mmcu=atmega328p main.c -E -o main.i`
- `avr-gcc -mmcu=atmega328p main.c -g -fdump-tree-all-graph -O1`
- `xdot result.dot`

THE COMPILER

THE COMPILER

- Compiling requires the preprocessed code and three main analysis functions:
  - Lexical analysis
  - Sintax analysis
  - Semantic analysis
- The C compiler then:
  - Generates intermediate code
  - Optimizes it
  - Gives its output in assembly code

THE COMPILER

- `avr-gcc -mmcu=atmega328p main.c -S -o main.s`
- `vim main.s`

THE ASSEMBLER

THE ASSEMBLER

- Once the code is compiled into assembly, it is assembled into machine language:
- It translates, process directives and generates the object code.

THE ASSEMBLER

- `avr-gcc -mmcu=atmega328p main.c -o main.elf`
- `hexyl main.elf`

THE LINKER

THE LINKER

- The linker is in charge of linking (surprisingly) all the object files into a single file and bind the adresses according to the memory map provided.
- This is the last part of the process and this usualy ends with a .elf file.

THE LINKER

- avr-objdump main.elf -txS
- avr-objcopy -O ihex main.elf main.hex
- Hexyl main.hex

ROTABIT AND PATTERNS

ROTABIT AND PATTERNS

Check examples under

I7266/mega32A/002_Patterns/

- Once you get the hang of it, try to replicate the following patterns:
  - Binary count
  - Sweep
  - Rotabit
  - Wave
  - 2-1-toggle

SOFTWARE ARCHITECTURE

SOFTWARE ARCHITECTURE

- This term refers to the high-level design and organization of software components, modules, and subsystems that collectively form the software stack running on the embedded device.
- It encompasses the overall structure, relationships, and interactions between software elements, as well as the principles and patterns guiding their design and implementation.

THE C PROGRAM STRUCTURE

C PROGRAM STRUCTURE

- In embedded systems, a C program usually contains the following:
  - Configuration and Initialization
  - Main function and infinite loop
  - Utility functions and libraries
  - Hardware control functions
  - Boot or startup code

C PROGRAM STRUCTURE

- See the example from ArduinoClockv15

C PROGRAM STRUCTURE

- The parts of a program that we have used so far are two:
  - Configuration and initialization
  - Main loop
- The first refers to the code we use to set up the microcontroller to do what we want, for example, setting up outputs and inputs.
- The second refers to a software loop in which we put the code that we want the microcontroller to always execute.

INTERRUPTS

INTERRUPTS

- We are going back to the analogies: think about you in an evening having lots of stuff to do.
- Imagine you have no food in the fridge and you choose to cook some rice.
- The rice needs to cook for a few minutes, but you do not know the exact time, so you need to keep an eye on the pot so you do not have to eat burnt rice.
- We will call this: *Polling*.

INTERRUPTS

- The *event* that we are expecting is the rice being cooked, an event that we can recognize because the water in the pot has all evaporated.
- *Polling* refers to the action of taking a look at something every so often.
- An *event* is that something that we are expecting to happen.
- When we do this with the rice, if we do not check the rice at the right time, it might get burnt.

TOGGLE

- Let us try to read a button input.
- You will have to create a new program.
- In your new program, you will have to configure a pin as an input. Also, connect a pushbutton to that pin.
- You will also have to configure another pin as an output and connect an LED (with a resistor) to it.
- Your task is to toggle the LED each time you press the pushbutton.

TOGGLE

- To make things easier, let us review what the XOR bitwise operator does.
- The XOR operator outputs 1 if a single input (any of the two) is 1, but outputs 0 if both inputs are 1 or 0. In other words, outputs 1 when their inputs are different, and 0 when they are equal.
- If we apply an XOR operation to a byte with a mask, we will toggle the bits selected by the mask in the byte we are operating. Let us see some examples.

TOGGLE

- Check examples
  - I7266/mega32A/003_Toggle/001

INTERRUPTS

- Let us think of another analogy: what happens if you are doing your homework and then you feel the urge to go to the bathroom?
- In this case, the event is emptying your bladder, but this time you are not checking if your bladder is at a good level every so often. When nature calls, you have to run.
- This situation is the counterpart for *polling* an event, this is an event-driven routine, A.K.A. an *interrupt*.

INTERRUPTS

- There is another kind of interrupt. The one we just saw is an *internal interrupt*, since the event or the signal triggers the interrupt comes from inside our system (our body).
- Now think the same scenario of the rice, but this time imagine you say “alexa, set a timer for 12 minutes” when your rice starts cooking. This way we know we will have a signal coming from the exterior when the rice is done. You can think of some other cooking examples where this applies.

INTERRUPTS

- In these new examples, we are waiting for an *external interrupt*, since the signal that triggers the interrupt comes from the exterior.
- Internal interrupts are also called *software interrupts*.
- External interrupts are also called *hardware interrupts*.

- In short, interrupts make us stop whatever we are doing to switch to another task that needs our attention.

EXTERNAL INTERRUPTS

EXTERNAL INTERRUPTS

- Let us check the datasheed and look for information about external interrupts.

INTERNAL INTERRUPTS

NOTICES!

WhatsApp and classroom groups, missing data, class formality.

THANKS!

Please feel free to ask any related questions at any time.

C EMBEDDED PROGRAMMING

What is the difference?

EMBEDDED C

- It is a sort of variant of the C language. It keeps the rules, but it is also tailored in the way it is used to be more effective in embedded systems.
- Think of the constrains of embedded systems:
  - Fast response (real-time operation)
  - Limited storage space
  - Low possibilities of changing firmware after production
  - Among many others

FIRMWARE VS EMBEDDED SOFTWARE

- Although these may be concepts that are still evolving with the development of ever complex embedded technologies, my take on the question is the following:
  - Embedded software refers to any kind of software, (i.e., code), that runs in a micro-electronic system with a limited functional scope.
  - Firmware refers to the software in charge of the abstraction of hardware functionality as well as the basic behaviour of an embedded system.
- This is not the most important piece of information, but the topic leads to the question:
  - How complex and multi-purpose can an embedded system be?

EMBEDDED SYSTEMS SOFTWARE ARCHITECHTURE

SOFTWARE ARCHITECTURE

- With the improvements in integrated circuit manufacturing, microprocessors became microcontrollers, and microcontrollers became system-on-chips.
- As the hardware improves, the software has more room to grow too. This means that old programming structures or methods become obsolete.
- Let us use this time to explore how we will make a program