Computer organization and architecture

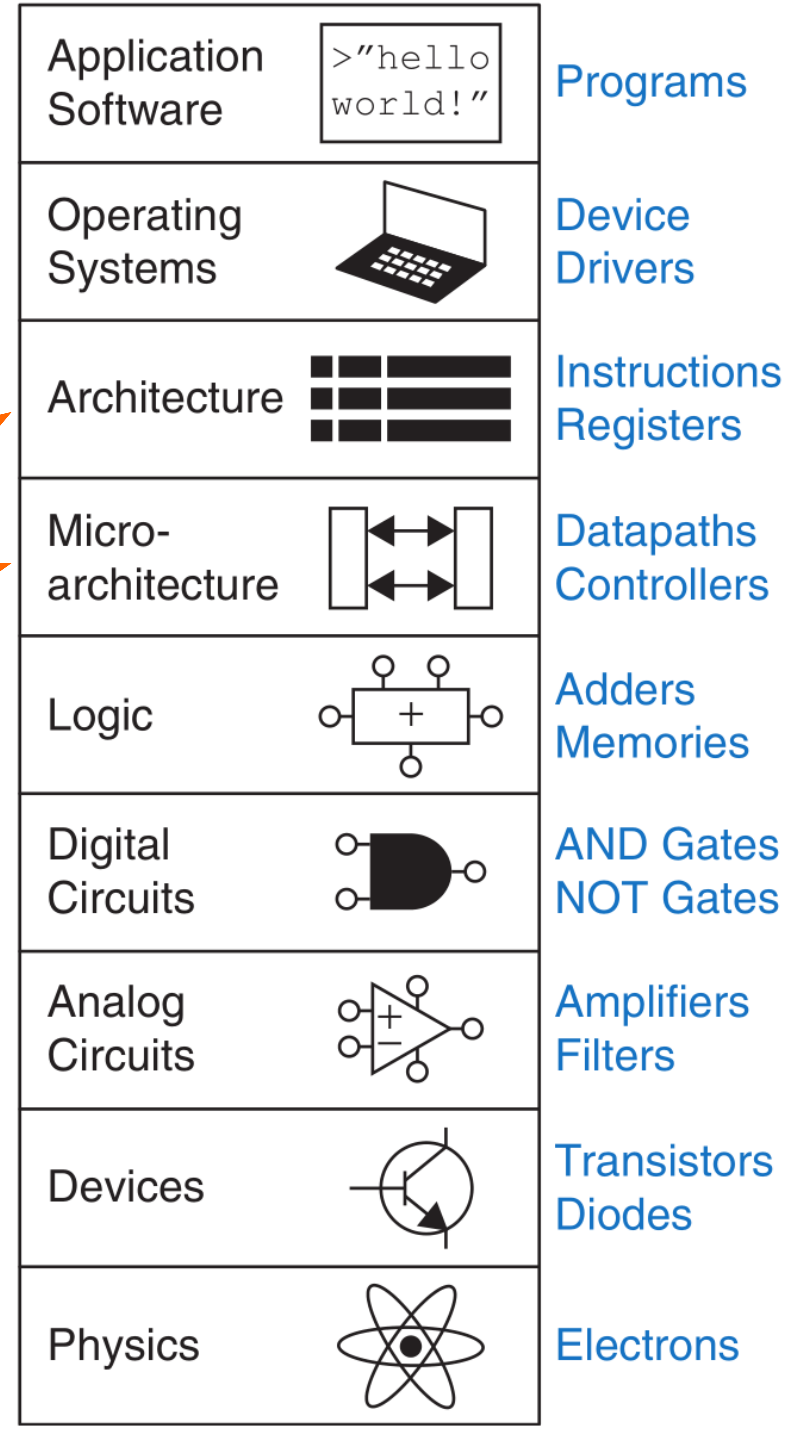
Lesson 1

Introduction

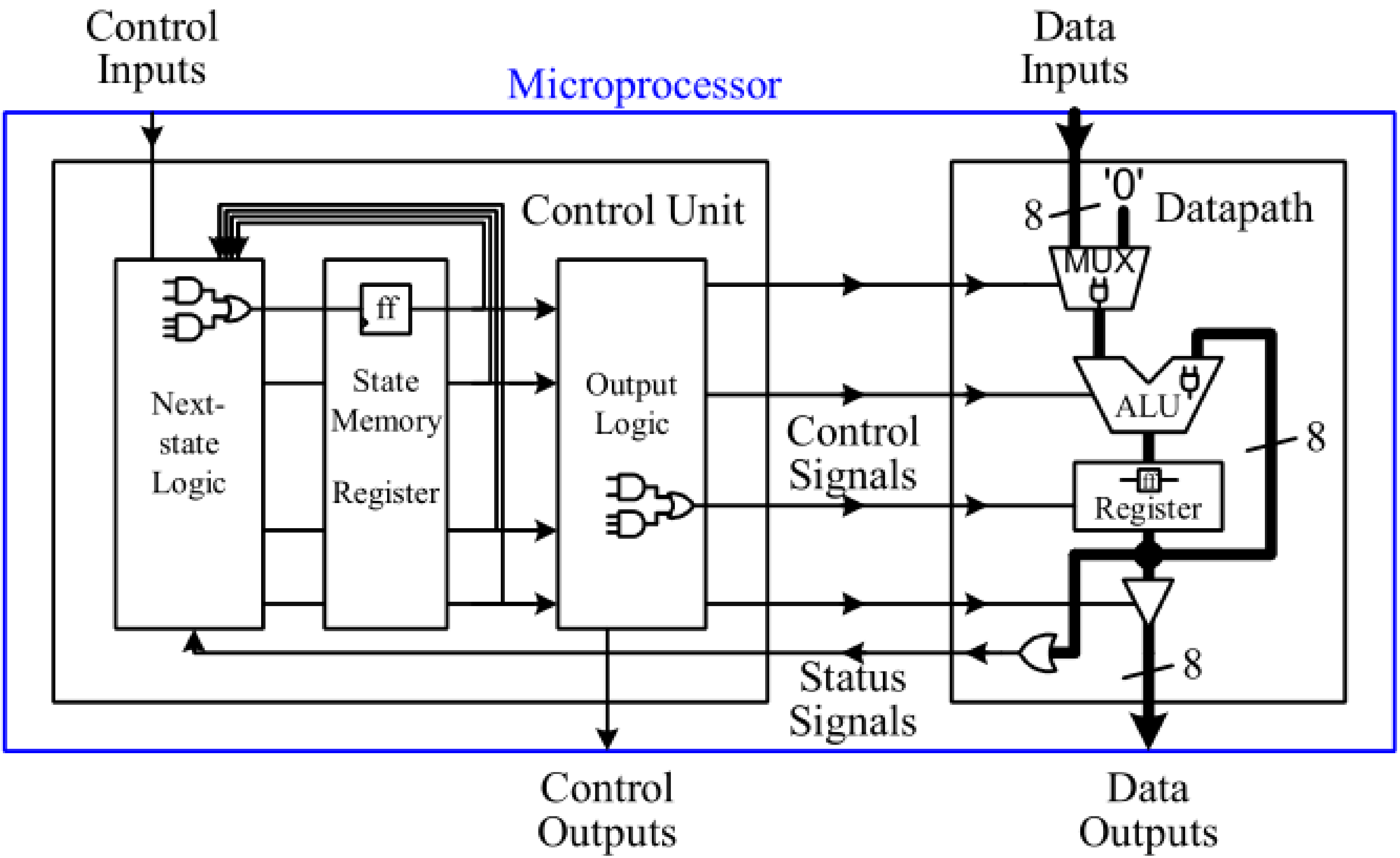
Levels of abstraction for electronic computing system

We are here

In this course, we will apply our knowledge from Digital Logic to learning internals of microprocessors and microcontrollers.

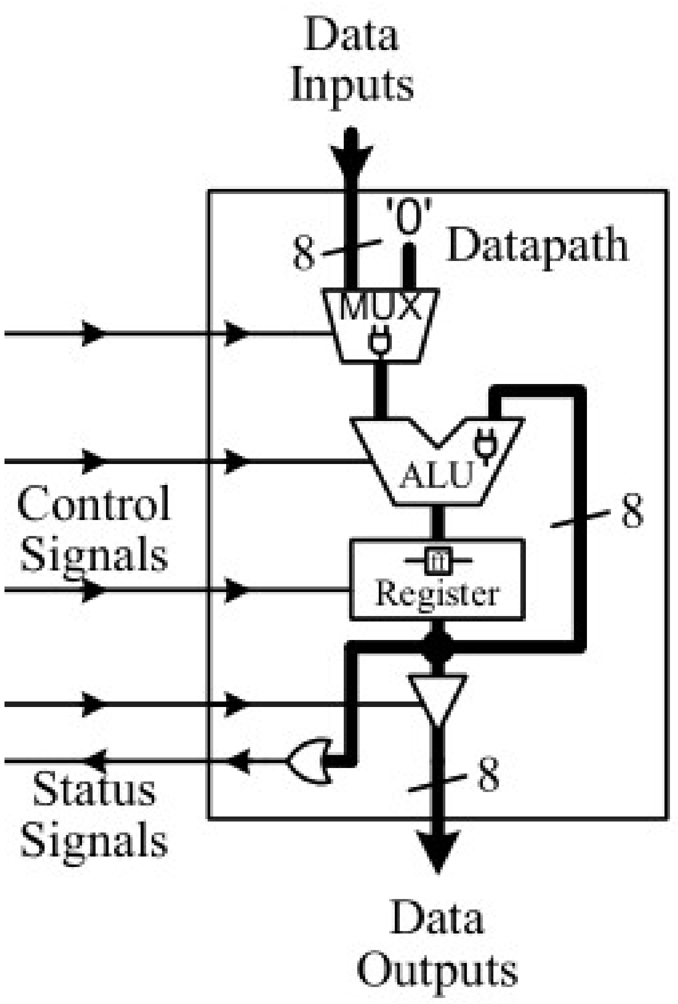


Internal parts of a microprocessor



Datapath

– responsible for the execution of data operations performed by the microprocessor, such as the addition of two numbers inside the arithmetic logic unit (ALU).

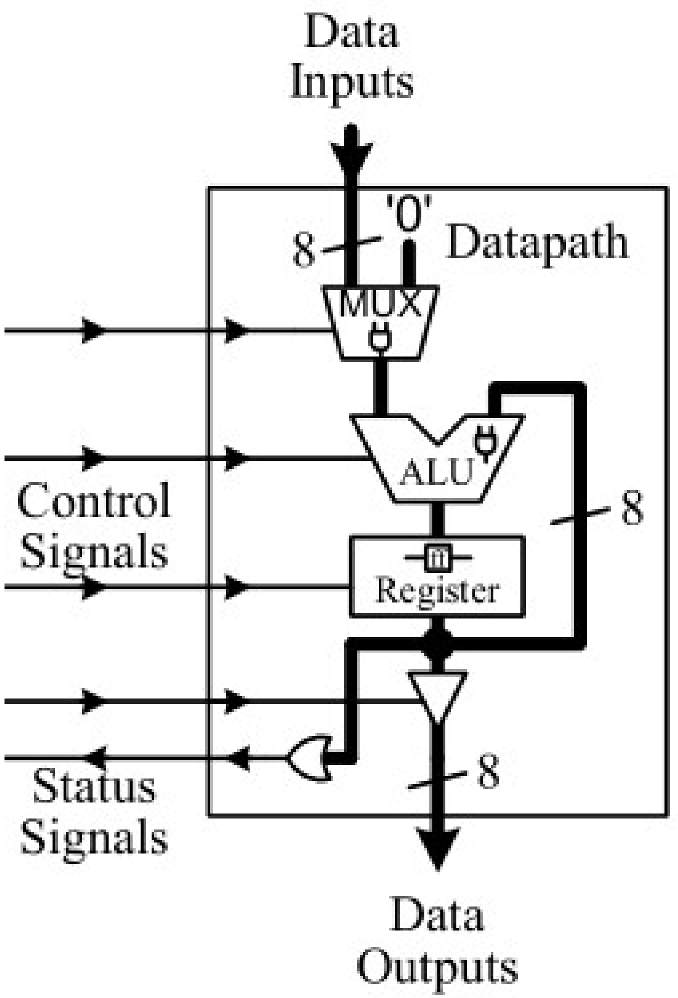


– includes registers for the temporary storage of data

Several data signal lines are grouped together to form a bus.

The width of the bus (the number of data signal lines in the group) is annotated next to the bus line.

Datapath

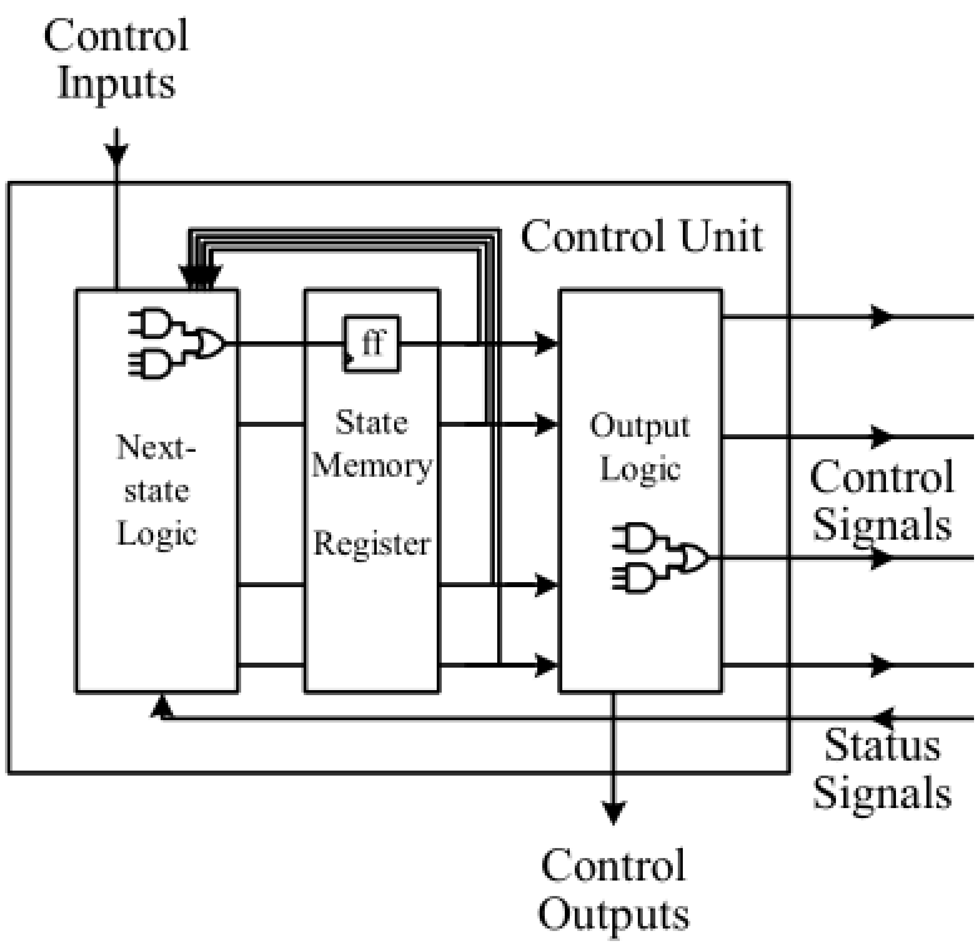


Multiplexers (MUXes) are for selecting data from two or more sources to go to one destination.

The tri-state buffer is used to control the output of the data from the register.

Control Unit (controller)

– controls the operations of the datapath, and therefore, the operations of the entire microprocessor.

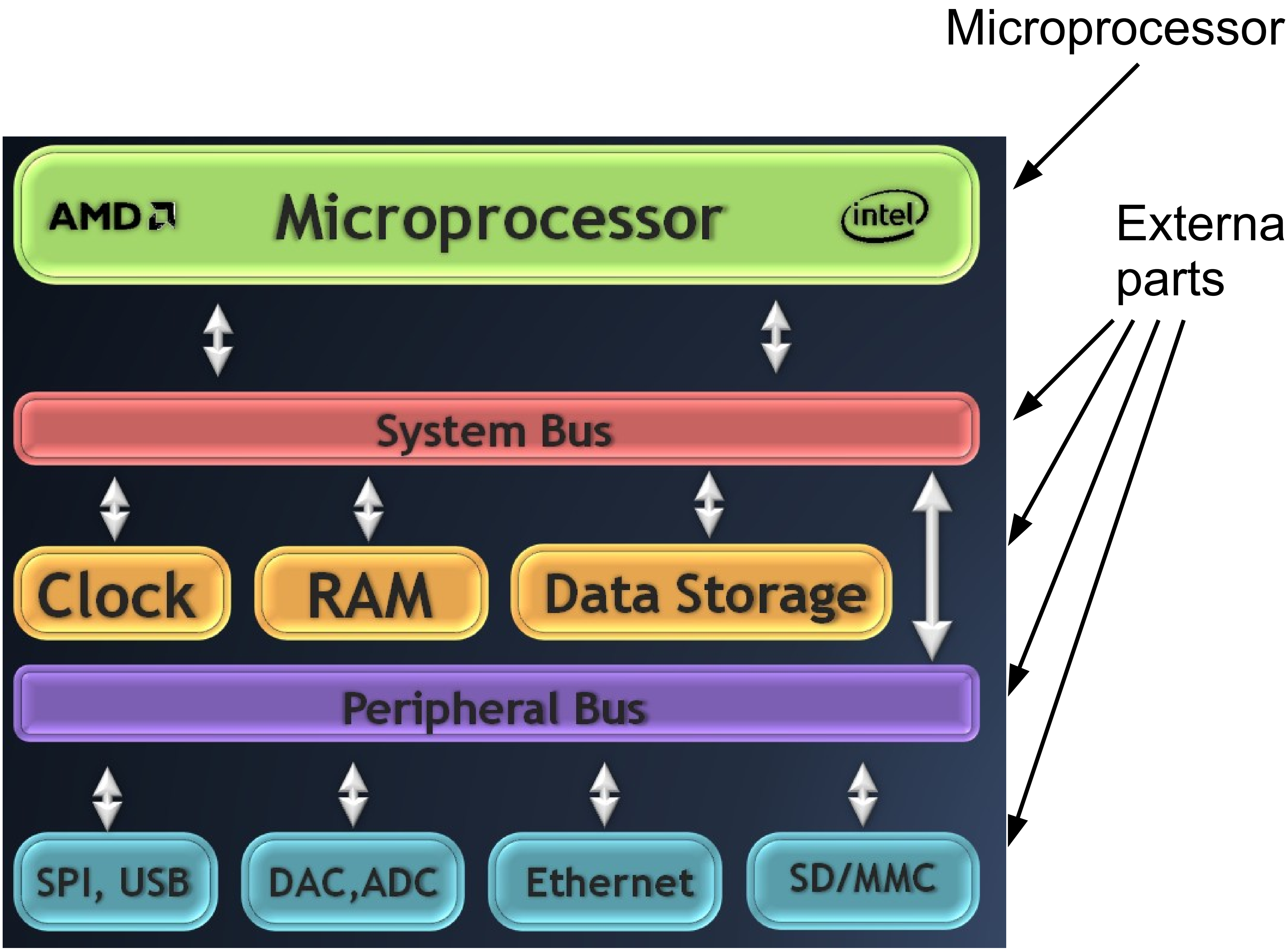


The control unit is a finite state machine (FSM)

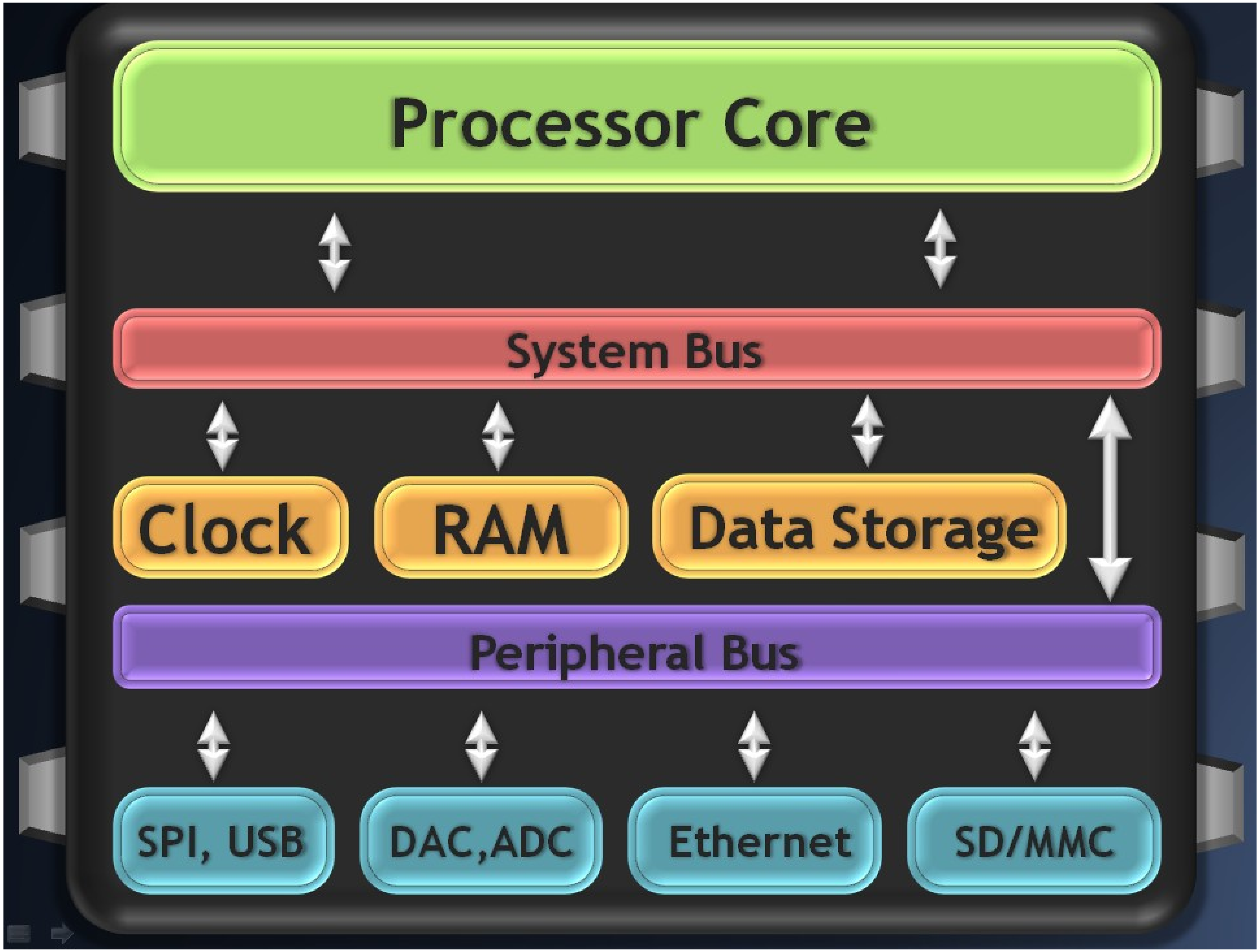
The output logic of this FSM generates the control signals for controlling the datapath.

Microcontroller versus Microprocessor

	Microprocessor	Microcontroller
Applications	General computing (i.e. Laptops, tablets)	Appliances, specialized devices
Speed	Fast (GHz)	Slow (MHz)
External Parts	Many	Few
Cost	High	Low
Energy Use	High	Low


Microprocessor
External parts
AMD
Microprocessor
intel
System Bus
Clock
RAM
Data Storage
Peripheral Bus
SPI, USB
DAC,ADC
Ethernet
SD/MMC

Microcontroller



All these parts are inside the microcontroller.

The **architecture** is defined by

- – the instruction set (language)
- – operand locations (registers and memory)

Instructions – the words in a computer's language

Instruction set – the computer's vocabulary

All programs running on a computer use the same instruction set.

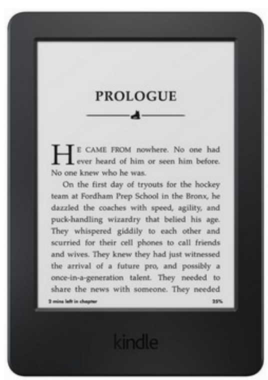
Even complex software applications are eventually compiled into a series of simple instructions such as add, subtract, and branch.

Many different architectures exist:

MIPS

x86

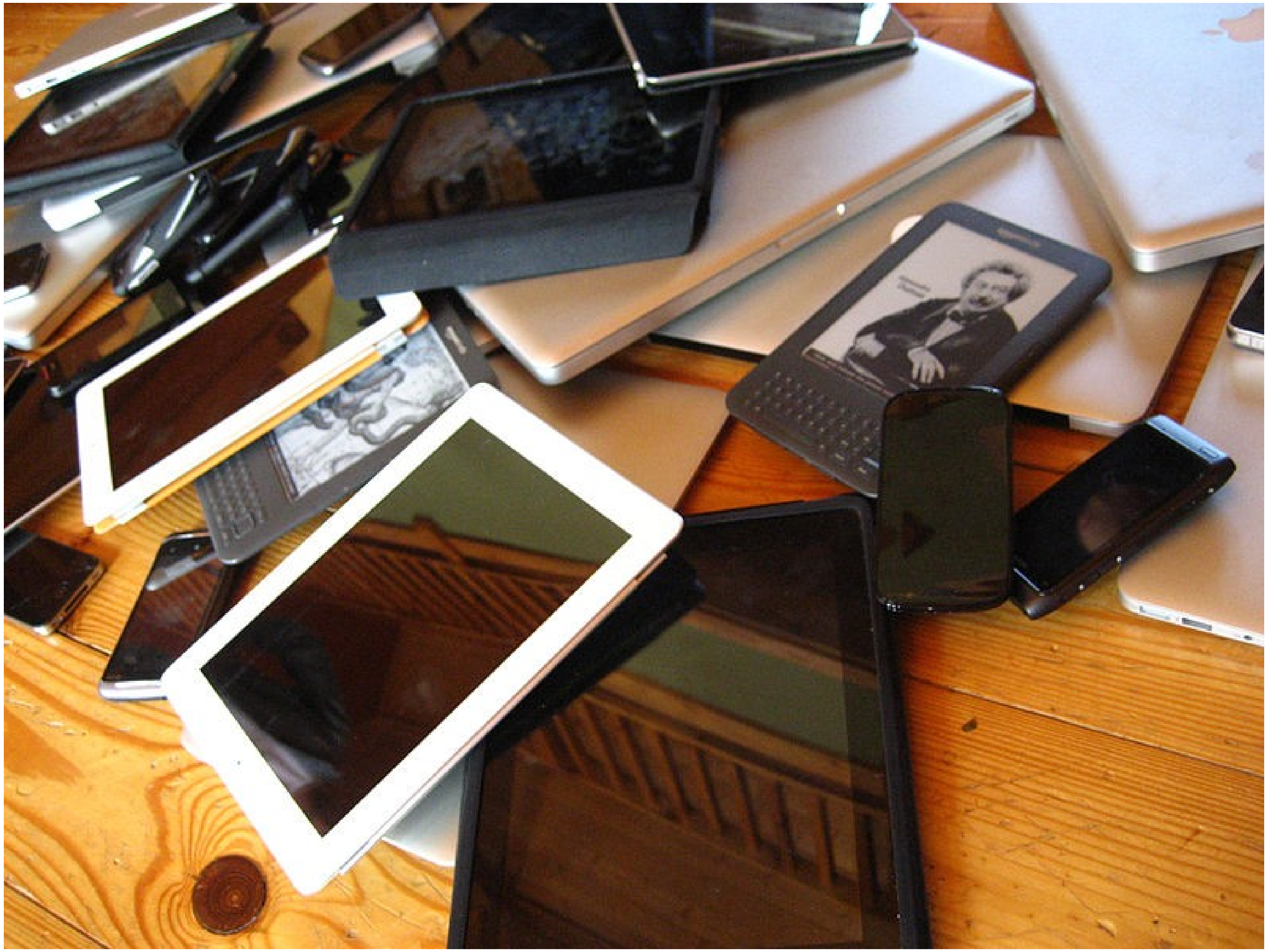
ARM



AVR

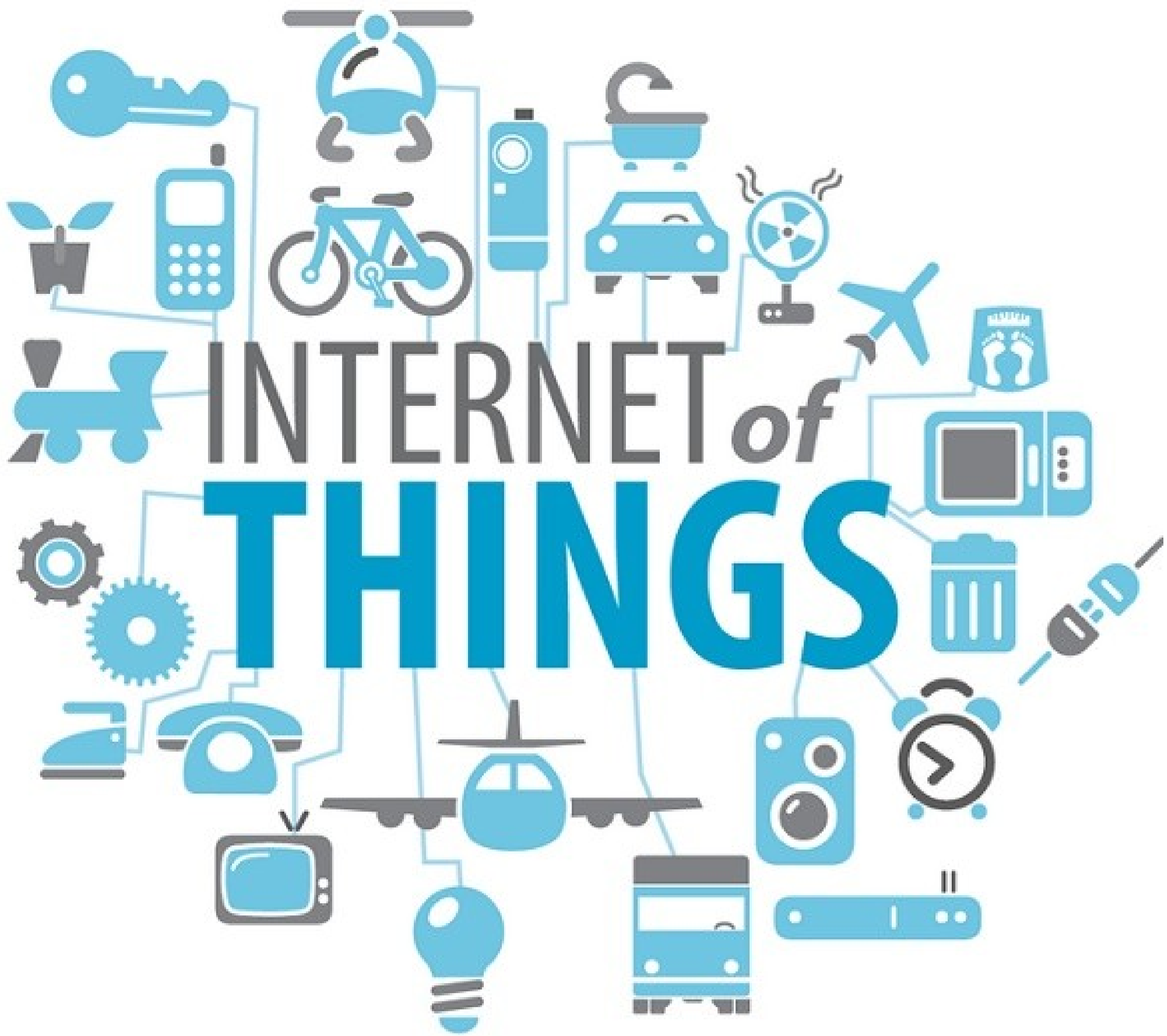
Also, SPARC, PowerPC, ...

More than 75% of humans on the planet use products with ARM processors.



Nearly every cell phone and tablet sold contains one or more ARM processors.

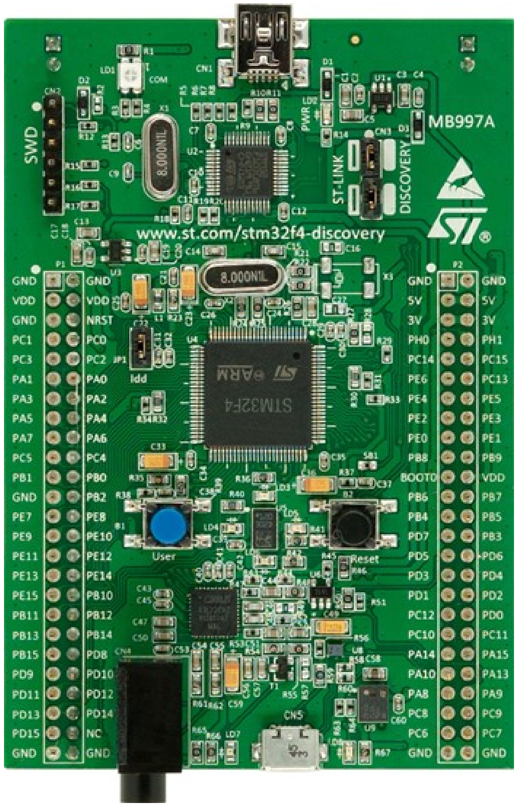
Forecasts predict tens of billions more ARM processors soon controlling the Internet of Things.



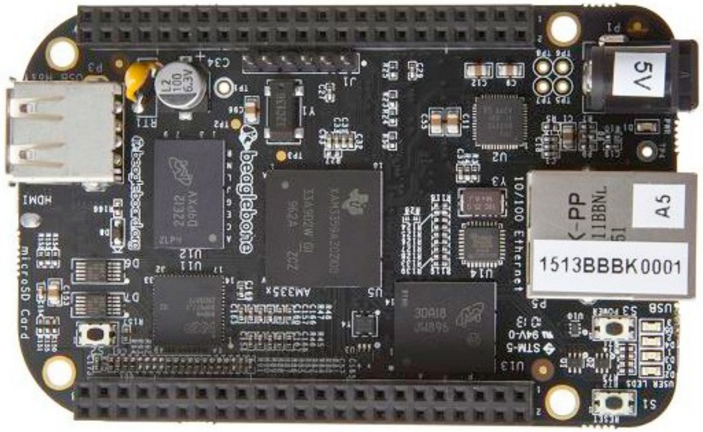
Raspberry Pi - a very popular ARM-based embedded Linux single board computer.

https://www.pjrc.com/teensy/

https://stm32f4-discovery.net/



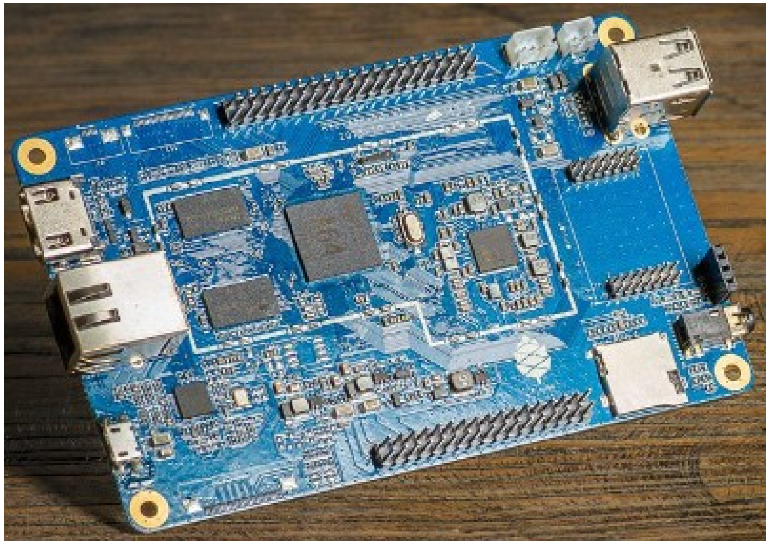
https://beagleboard.org/



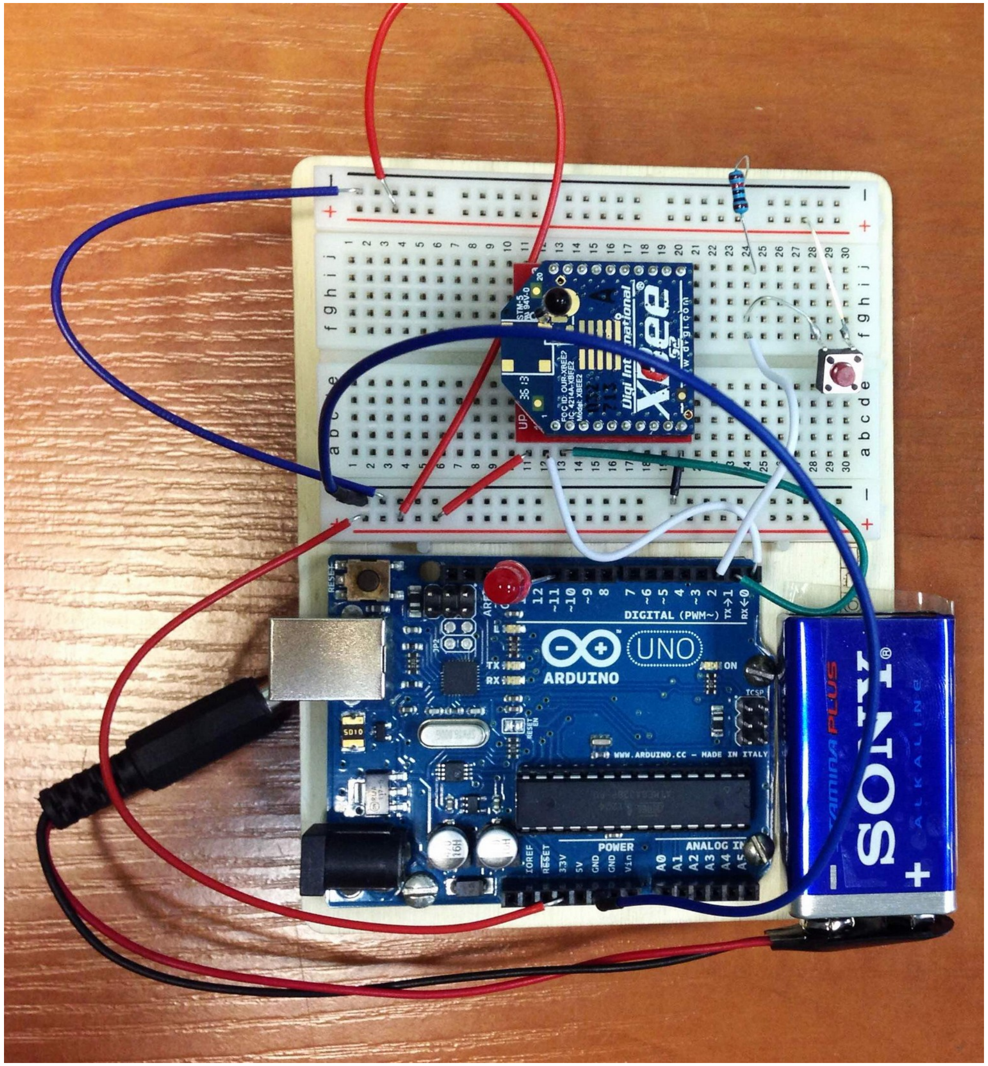
They are ARM

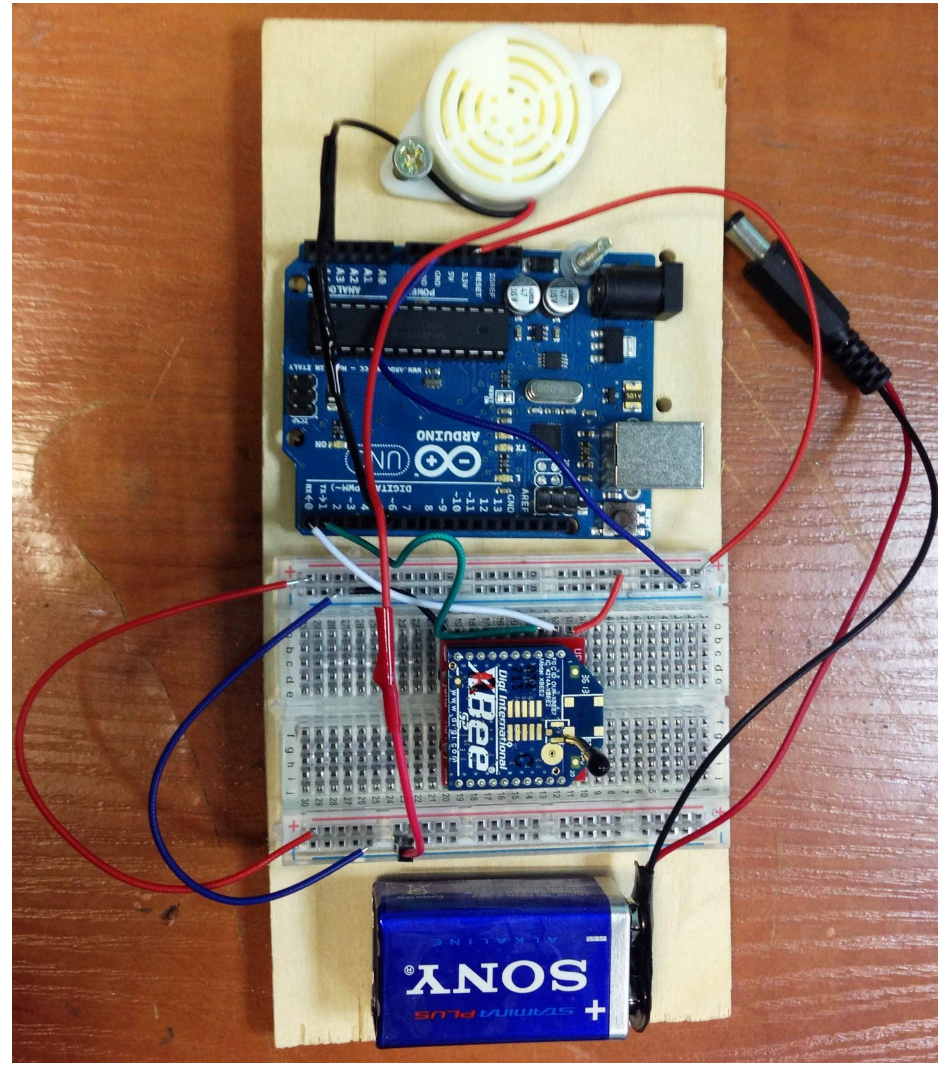
https://www.parallella.org/

http://pine64.com/



Wireless sensor networks





XBee has ARM Cortex M3

Computer hardware understands only 1's and 0's, so instructions are encoded as binary numbers in a format called **machine language**.

The ARM architecture represents each instruction as a 32-bit word.

Reading machine language is tedious for humans, so we represent the instructions in a symbolic format called **assembly language**.

The assembly code is converted into executable machine code by a utility program referred to as an **assembler**

Example for ARM

machine language instruction:

```
1110 0001 1010 0000 0011 0000 0000 1001
```

The meaning is:

copy the value from “register 9” into “register 3”

in assembly language

```
MOV R3, R9
```

http://www.toves.org/books/arm/

Adding the numbers from 1 to 10 in C

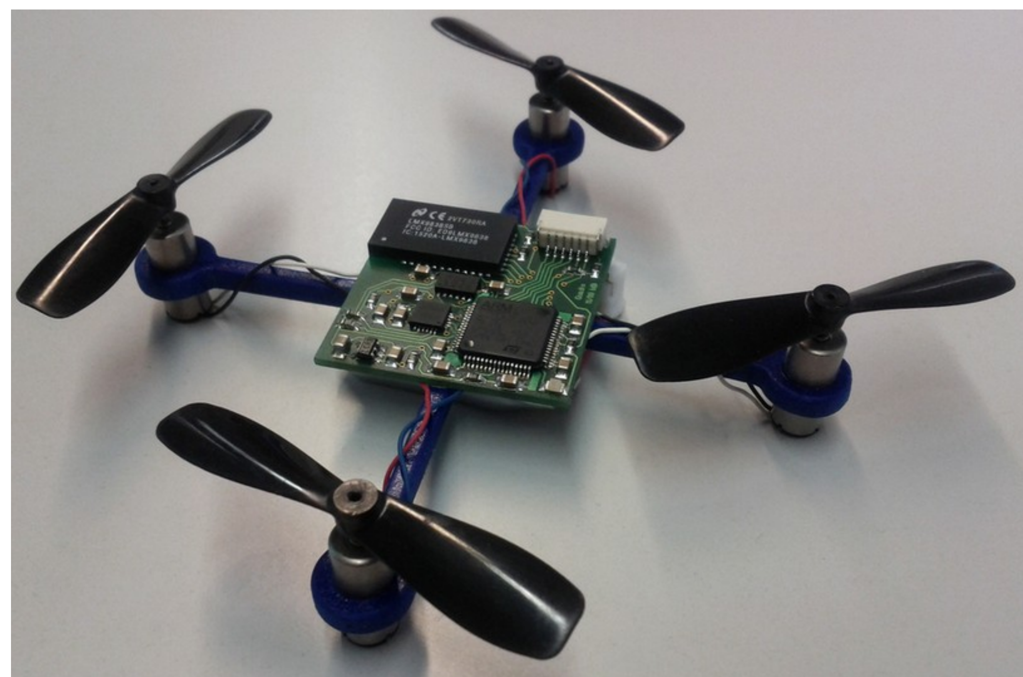
```c
int total;
int i;

total = 0;
for (i = 10; i > 0; i--) {
    total += i;
}
```

Adding the numbers from 1 to 10 in ARM assembly

```
        MOV  R0, #0        ; R0 accumulates total
        MOV  R1, #10       ; R1 counts from 10 down to 1
again   ADD  R0, R0, R1
        SUBS R1, R1, #1
        BNE  again
halt    B    halt      ; infinite loop to stop computation
```

Assembly language programming is important for small devices

Due to power and price constraints, the devices have very few resources, and developers can use assembly language to use these resources as efficiently as possible.



High-level:

```
int pow = 1;
int x = 0;

while (pow != 128) {
   pow = pow * 2;
   x = x + 1;
}
```

ARM assembly:

```
; R0 = pow, R1 = x
  MOV R0, #1 ; pow = 1
  MOV R1, #0 ; x = 0
WHILE
  CMP R0, #128 ; pow != 128 ?
  BEQ DONE ; if pow == 128, exit loop
  LSL R0, R0, #1 ; pow = pow * 2
  ADD R1, R1, #1 ; x = x + 1
  B WHILE ; repeat loop
DONE
```

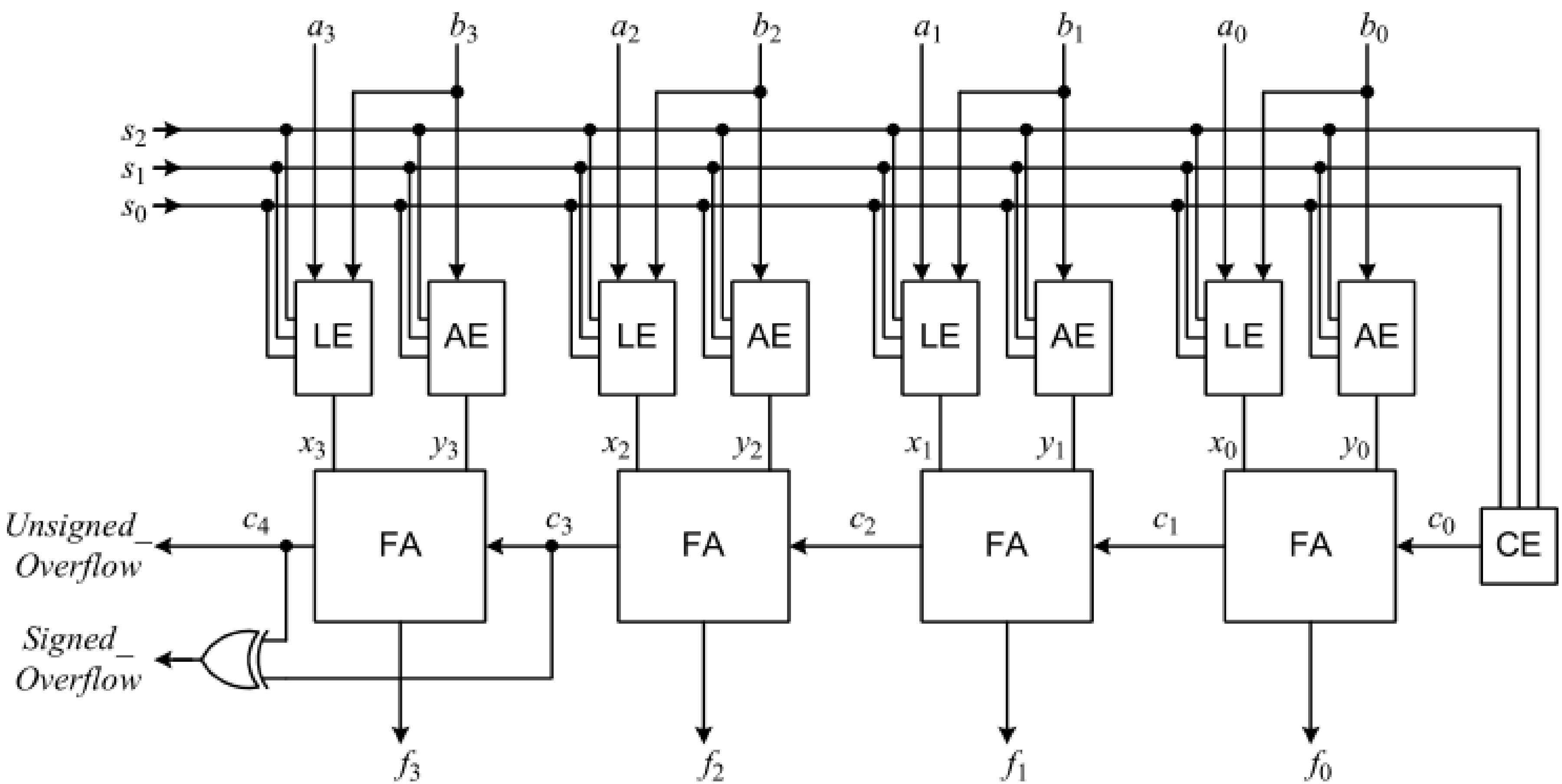
MIPS assembly:

```
# $s0 = pow, $s1 = x
  addi $s0, $0, 1 # pow = 1
  addi $s1, $0, 0 # x = 0
  addi $t0, $0, 128 # t0 = 128 for comparison
while:
  beq $s0, $t0, done # if pow == 128, exit while loop
  sll $s0, $s0, 1 # pow = pow * 2
  addi $s1, $s1, 1 # x = x + 1
  j while
done:
```

A computer architecture does not define the underlying hardware implementation.

Many different hardware implementations of a single architecture exist.

Example: different ALU for the same architecture



Example: Intel and Advanced Micro Devices (AMD) both sell various microprocessors belonging to the same x86 architecture.

They all can run the same programs, but they use different underlying hardware and therefore offer trade-offs in performance, price, and power.

Some microprocessors are optimized for high-performance servers, whereas others are optimized for long battery life in laptop computers.

The specific arrangement of registers, memories, ALUs, and other building blocks to form a microprocessor is called the **microarchitecture**.

Often, many different microarchitectures exist for a single architecture.

Books for this course

Sarah Harris, David Harris. Digital Design and Computer Architecture. ARM Edition.

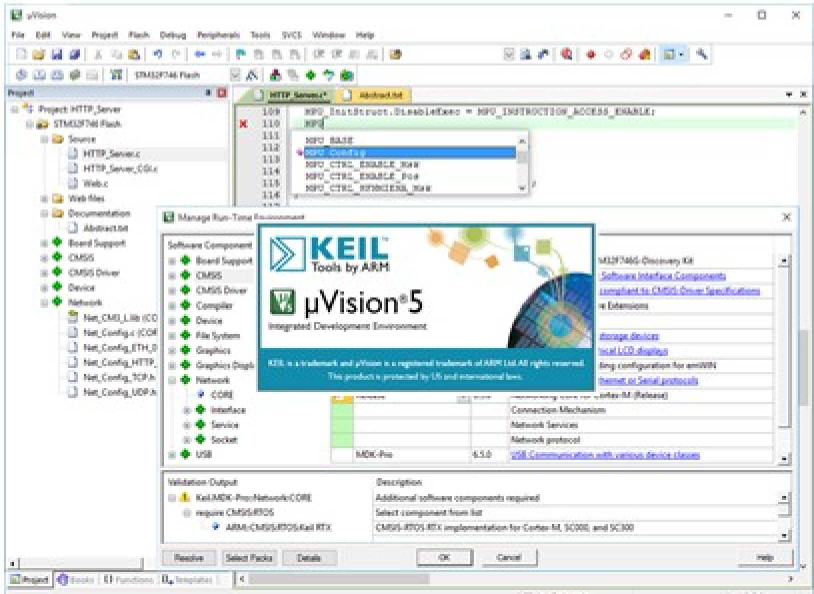
David A. Patterson, John L. Hennessy. Computer Organization and Design. The Hardware Software Interface. ARM Edition.

William Hohl, Christopher Hinds. ARM assembly language Fundamentals and Techniques. Second edition.

Ata Elahi, Trevor Arjeski. ARM Assembly Language with Hardware Experiments.

Andrew K. Dennis. Raspberry Pi computer architecture essentials.

MDK Microcontroller Development Kit



Download and install it

http://www2.keil.com/mdk5

Or get the file

MDK522.EXE

from me

Also install legacy support for ARM7, ARM9 & Cortex-R

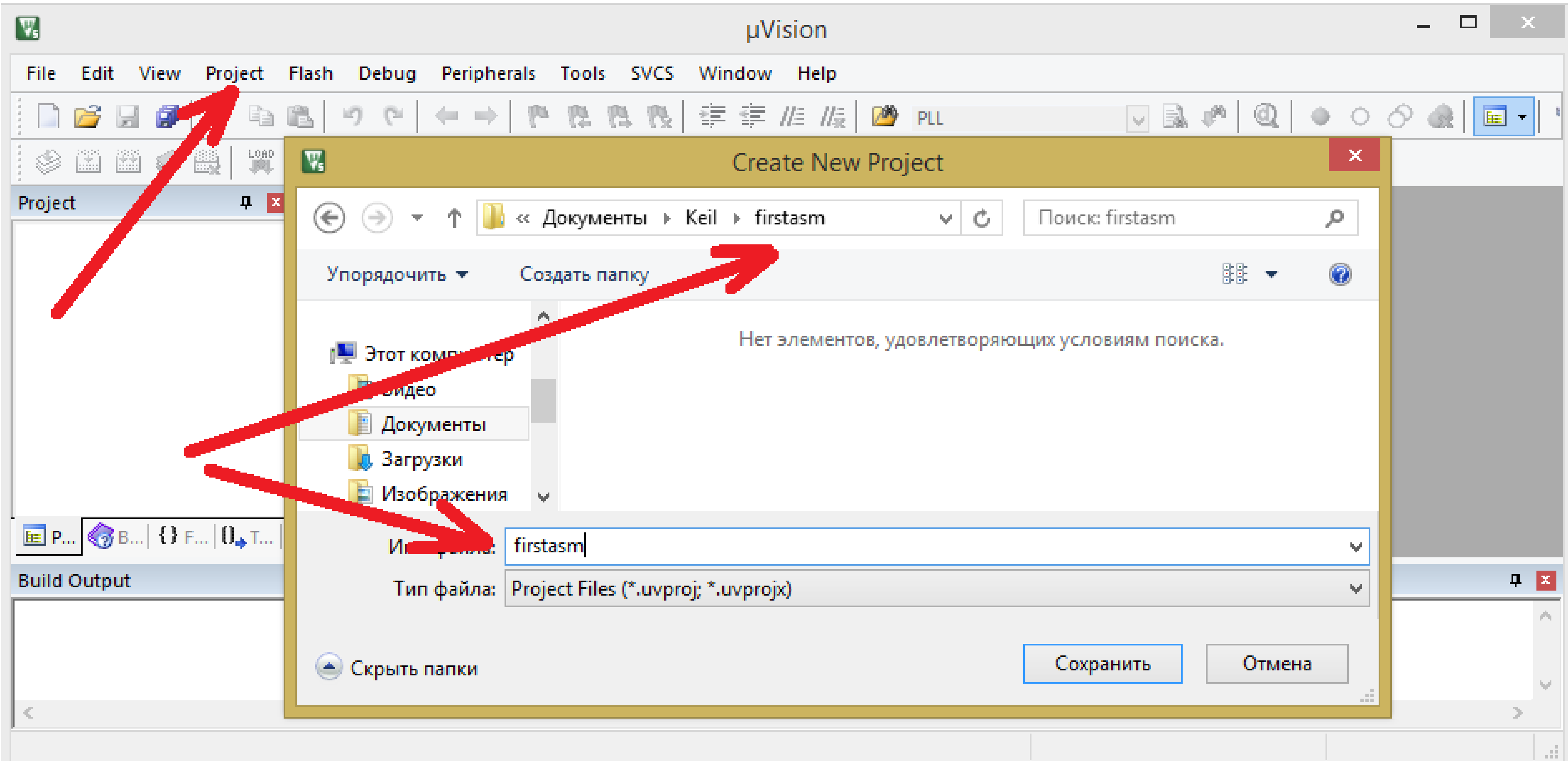
http://www2.keil.com/mdk5/legacy

Or get the file

MDK79522.EXE

from me

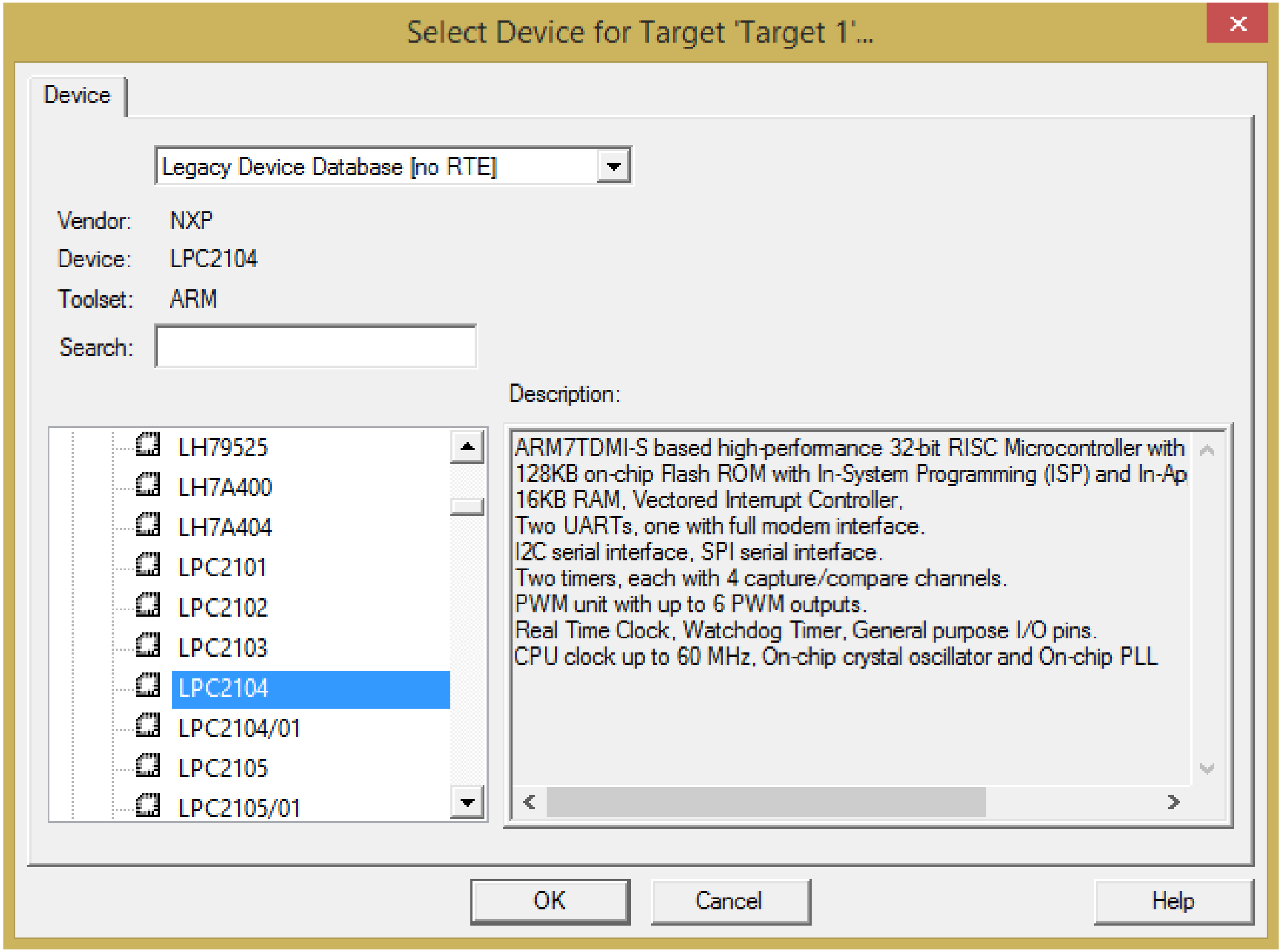
Start Keil and choose **New µVision Project** from the **Project** menu



Save each project in a separate folder.

Specify the following device to simulate:

LPC2104 from NXP



Scroll down until NXP and select LPC2104

LPC2104

ARM7TDMI-S based high-performance 32-bit RISC Microcontroller with Thumb extensions

128KB on-chip Flash ROM with In-System Programming (ISP) and In-Application Programming (IAP)

16KB RAM

Vectored Interrupt Controller

Two UARTs, one with full modem interface

I2C serial interface, SPI serial interface

Two timers, each with 4 capture/compare channels

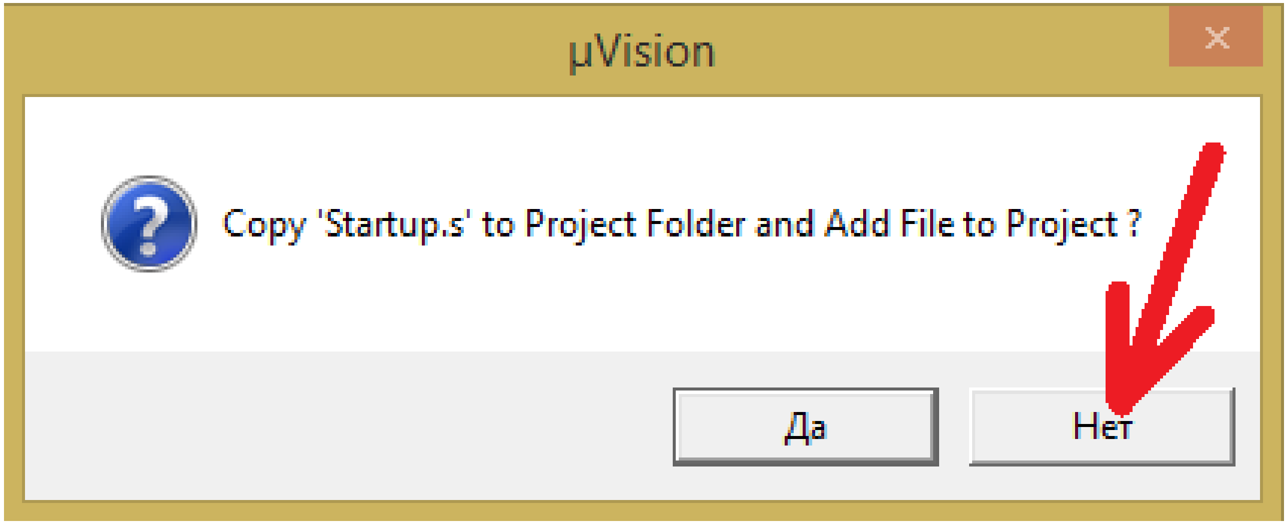
PWM unit with up to 6 PWM outputs

Real Time Clock, Watchdog Timer

CPU clk up to 60 MHz, on-chip x-tal oscillator, on-chip PLL

When you click OK, a dialog box will appear asking if you want to include startup code for this device.

Click **No**, since we are only making a small assembly program and will not need all of the initialization code.



Creating a source file

From the **File** menu, choose **New** to create your assembly file with the editor.

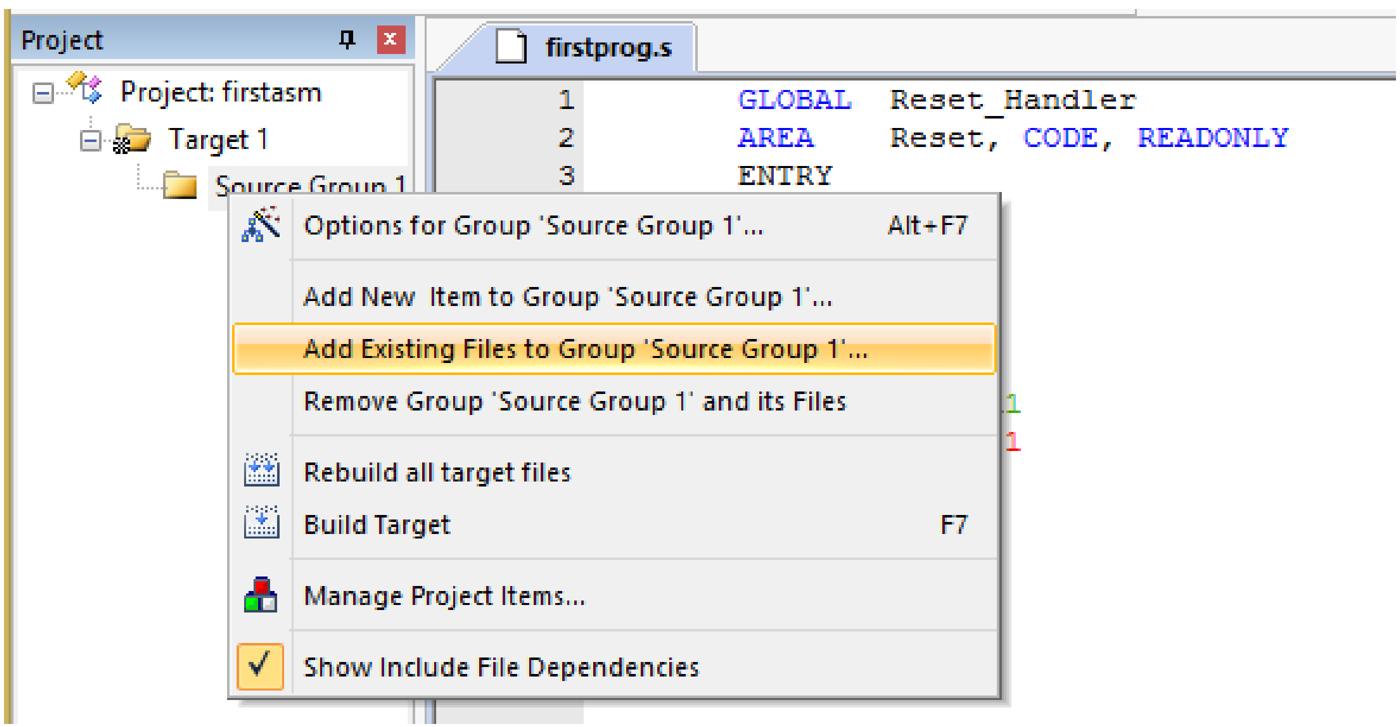
firstprog.s

```
            GLOBAL   Reset_Handler
            AREA     Reset, CODE, READONLY
            ENTRY

Reset_Handler

            MOV  R0, #0
            MOV  R1, #10
again       ADD  R0, R0, R1
            SUBS R1, R1, #1
            BNE  again

stop        B stop

            END
```

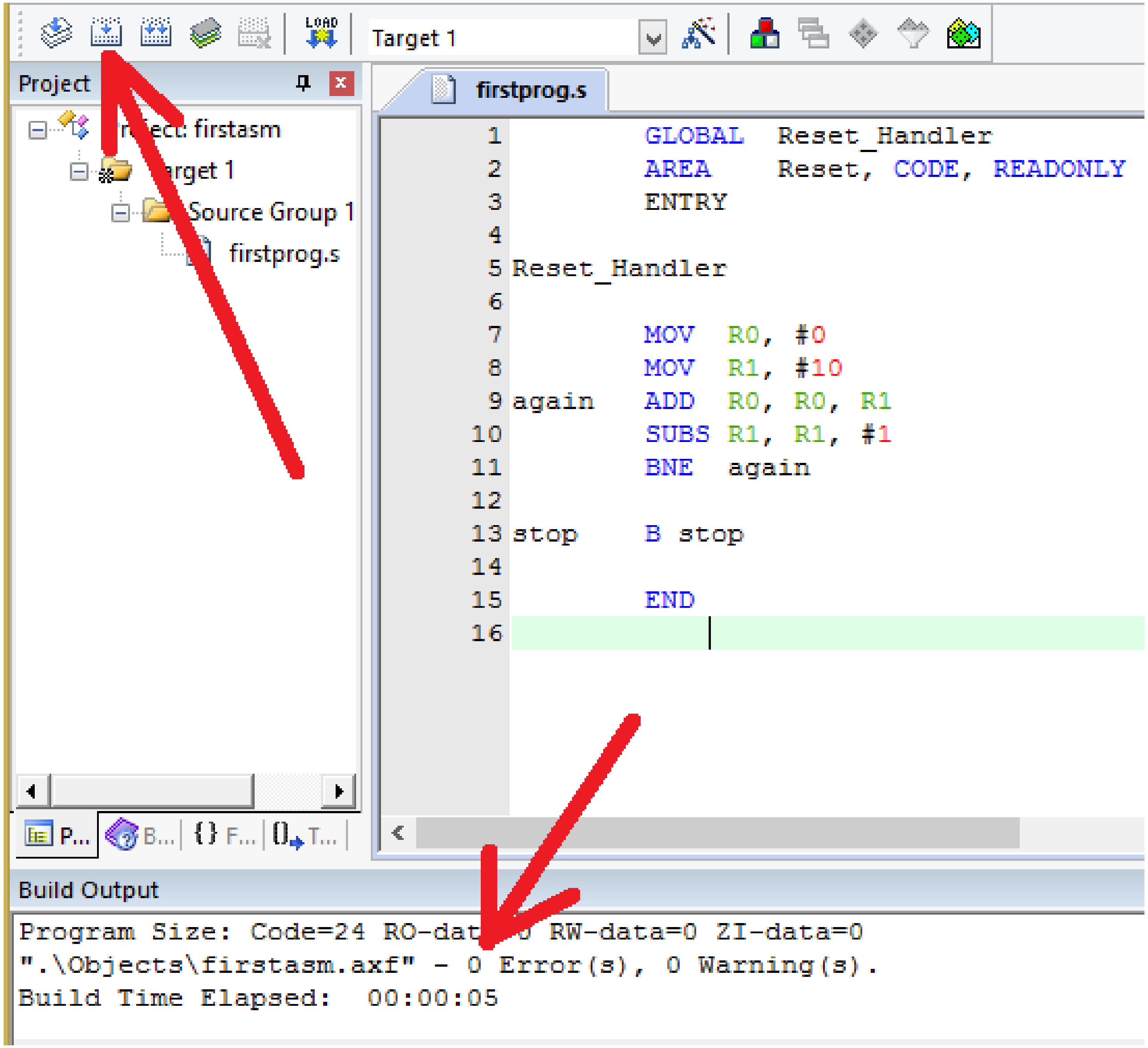
Save your file with the extension “.s”

The assembly file must be added to the project.

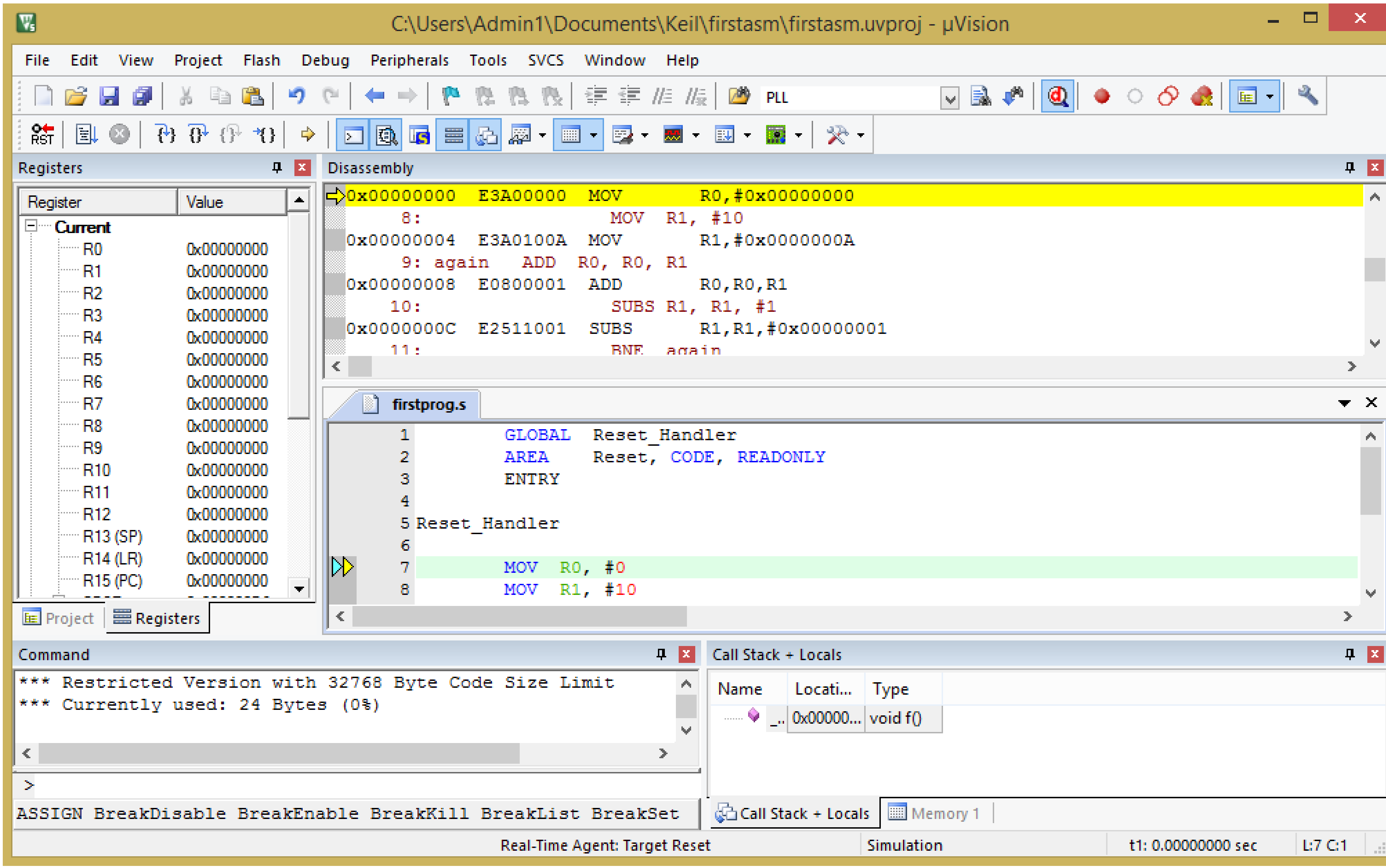
Right click on the **Source Group 1** folder, then choose **Add Files to Group "Source Group 1"**



To build the project, select **Build target** or **Rebuild all target files** from the **Project** menu.



From the **Debug** menu, choose **Start/Stop Debug Session**



You can single-step through the code, watching each instruction execute by clicking on the **Step Into** button on the toolbar or choosing **Step** from the **Debug** menu.

At this point, you can also view and change the contents of the register file, and view and change memory locations by typing in the address of interest.

When you are finished, choose **Start/Stop Debug Session** again from the **Debug** menu.

Exercise 1.1

What is the memory address of each instruction in your program?

Use View → Memory Windows to find each instruction in memory

Exercise 1.2

In this lecture, there was an example that

```
MOV R3, R9
```

is the machine language instruction

```
1110 0001 1010 0000 0011 0000 0000 1001
```

Use Keil to prove it.

Use Keil to find which machine instructions will correspond to

```
MOV R3, R8
```

and

```
MOV R4, R9
```