Department of Computer Science
National Tsing Hua University

CS 5244: Introduction to Cyber Physical Systems

Unit 5: Memory Architectures (Ch. 8)

Instructor: Cheng-Hsin Hsu

Acknowledgement: The instructor thanks Profs. Edward A. Lee & Sanjit A. Seshia at UC Berkeley for sharing their course materials

Memory Architecture: Issues

- Types of memory
- Stack
- Caches
- Scratchpad memories
- Absolute and relative addresses
- Virtual memory
- Heaps
  - allocation/deallocation
  - fragmentation
  - garbage collection
- Segmented memory spaces
- …

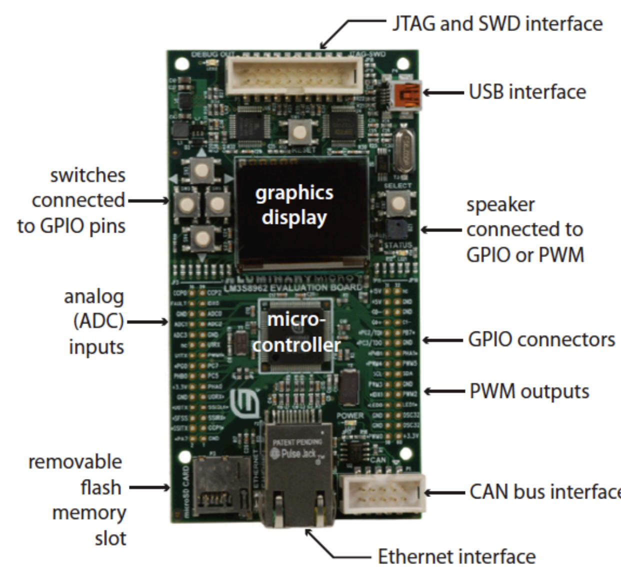
These issues loom larger in embedded systems than in general-purpose computing.

Specific Examples

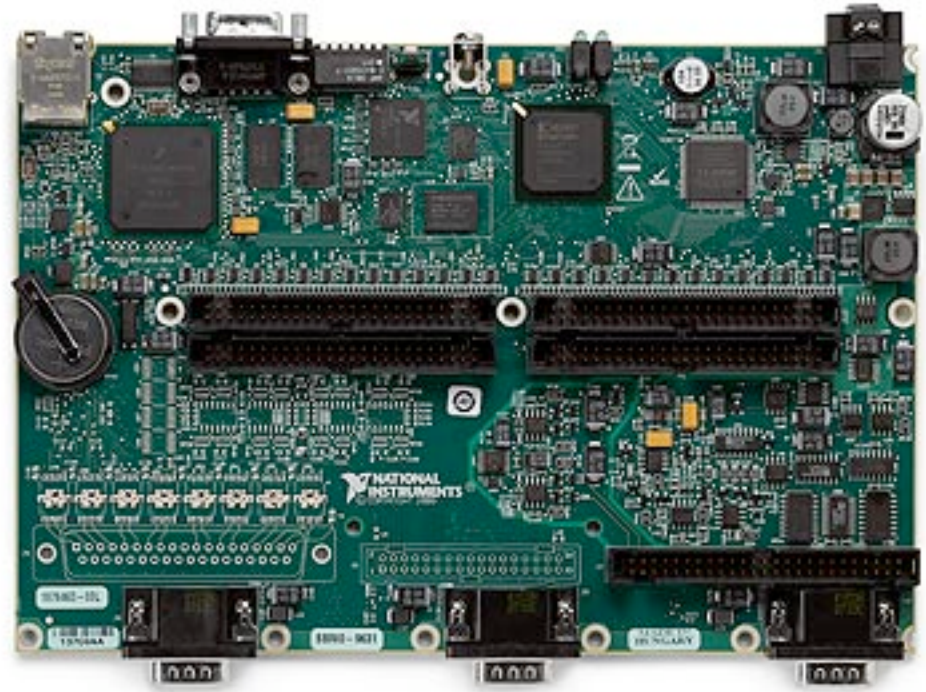
To be concrete:

- A low-end example:
  8-bit microcomputer

- A medium-end example:
  32-bit microcomputer



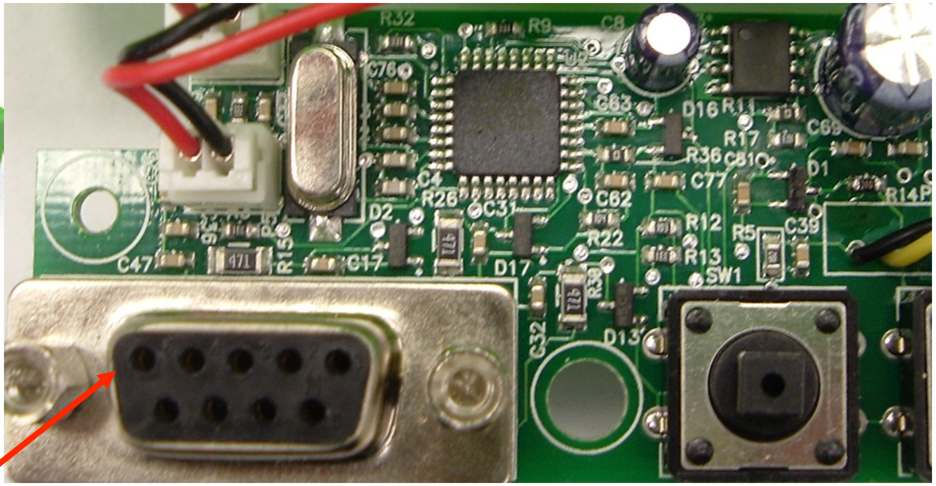
Specific Examples (Continued)

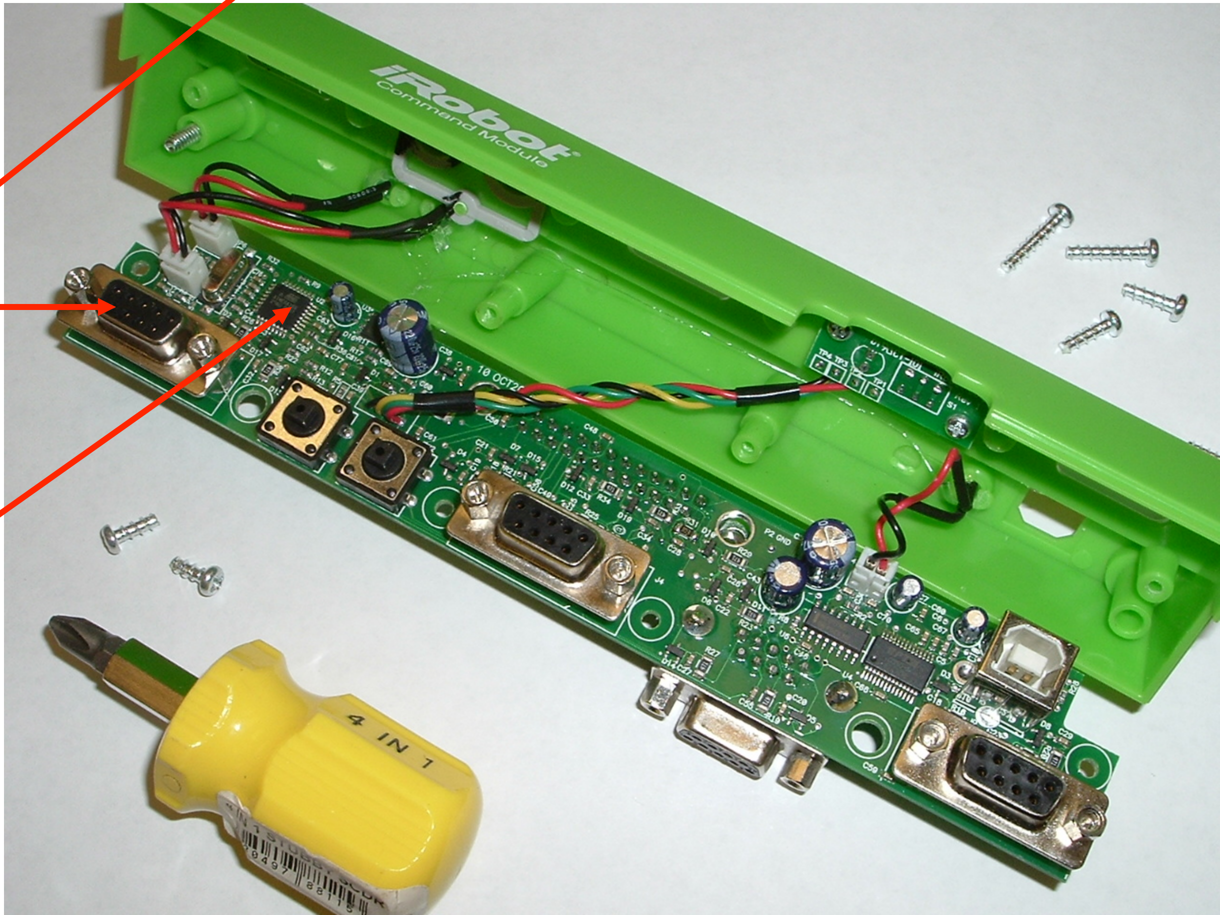


Single-Board RIO (National Instruments)

- Xilinx FPGA
  - In our lab: preconfigured with a 32-bit MicroBlaze microprocessor running without an operating system (“bare iron”).
- PowerPC processor (Freescale MPC5200)
  - In our lab: running VxWorks RTOS (real-time operating system) or LabVIEW Embedded.

The *iRobot Create* Command Module



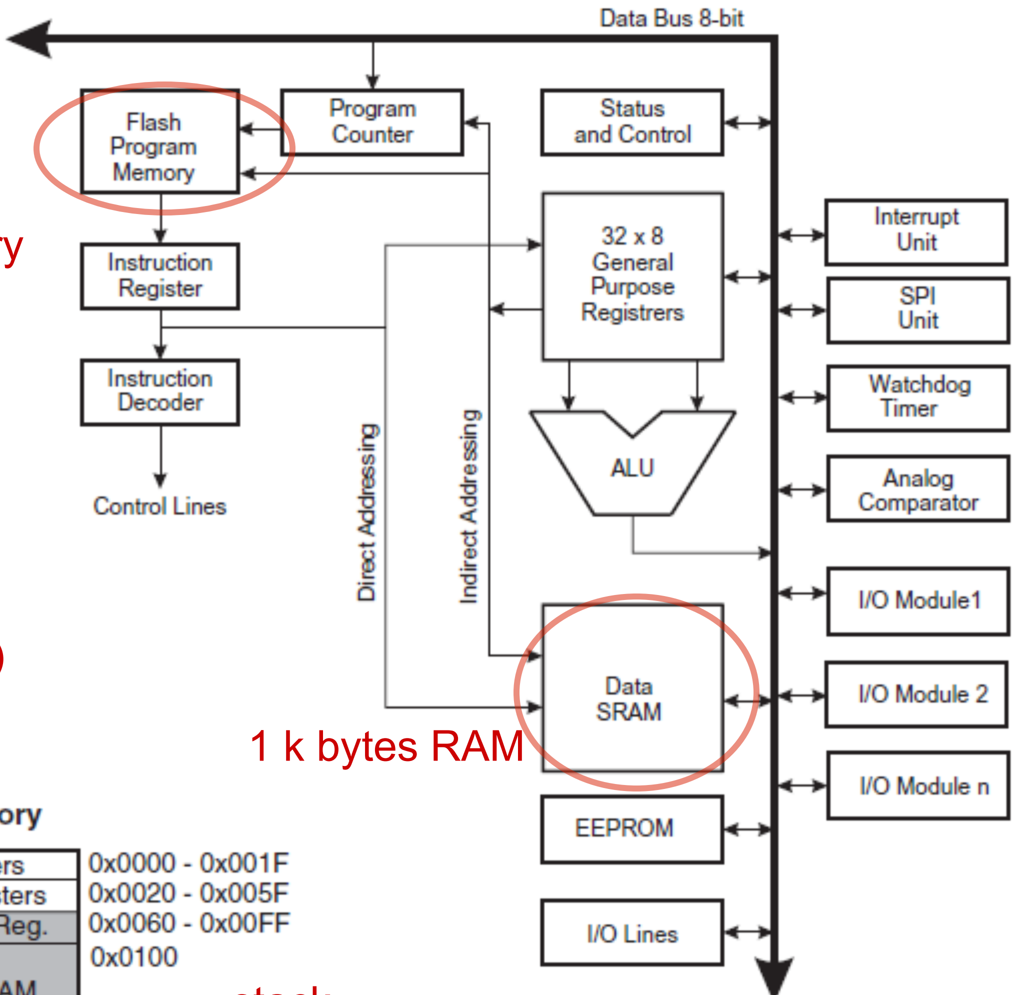


9-pin I/O port

Atmel ATMega 168
Microcontroller

ATMega168 Memory Architecture
An 8-bit microcontroller with 16-bit addresses

iRobot command module has 16K bytes flash memory (14,336 available for the user program. Includes interrupt vectors and boot loader.)



1 k bytes RAM

Example of a microcontroller architecture. Used in iRobot command module.

Additional I/O on the command module:
- •Two 8-bit timer/counters
- •One 16-bit timer/counter
- •6 PWM channels
- •8-channel, 10-bit ADC
- •One serial UART
- •2-wire serial interface

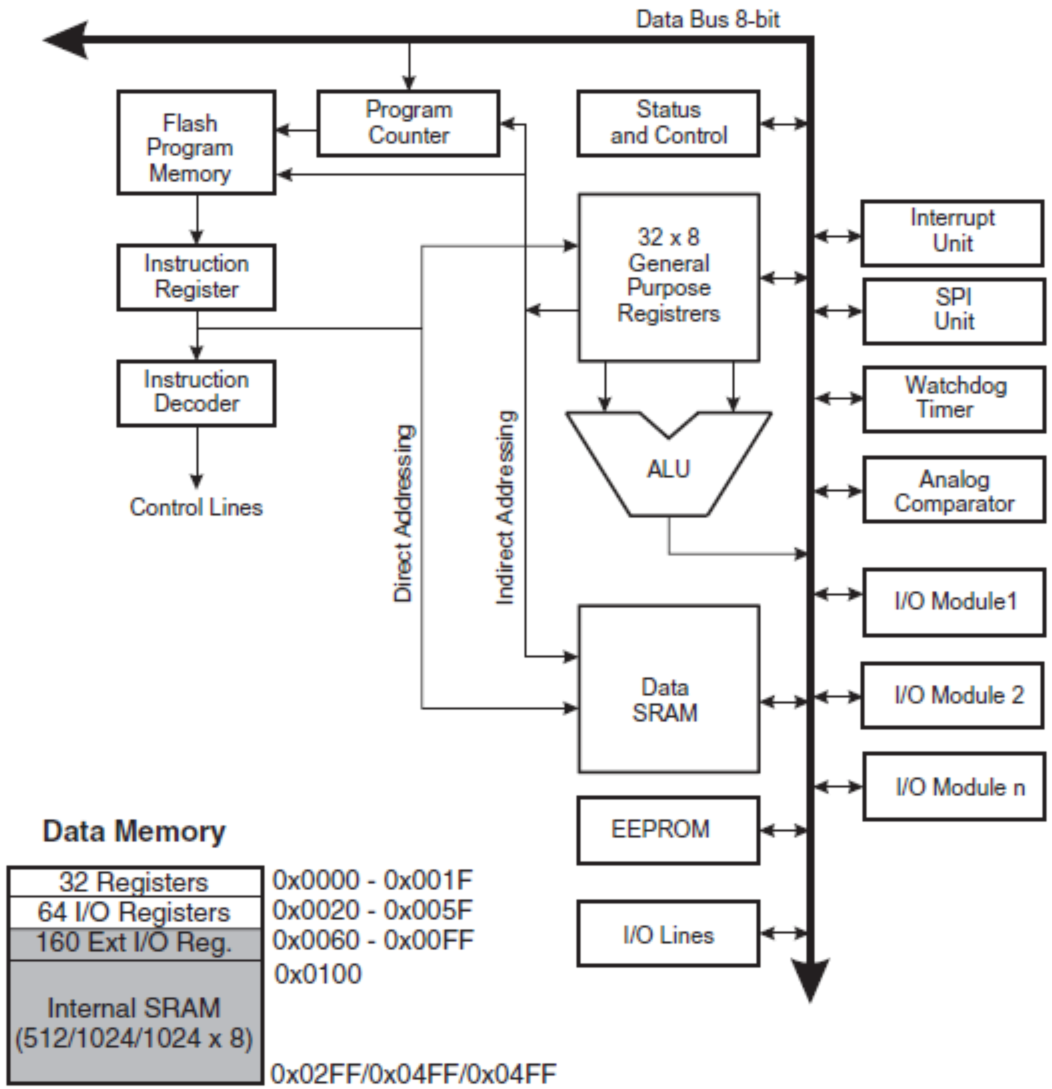
Data Memory

32 Registers	0x0000 - 0x001F
64 I/O Registers	0x0020 - 0x005F
160 Ext I/O Reg.	0x0060 - 0x00FF
Internal SRAM (512/1024/1024 x 8)	0x0100
	0x02FF/0x04FF/0x04FF

stack

Source: ATmega168 Reference Manual

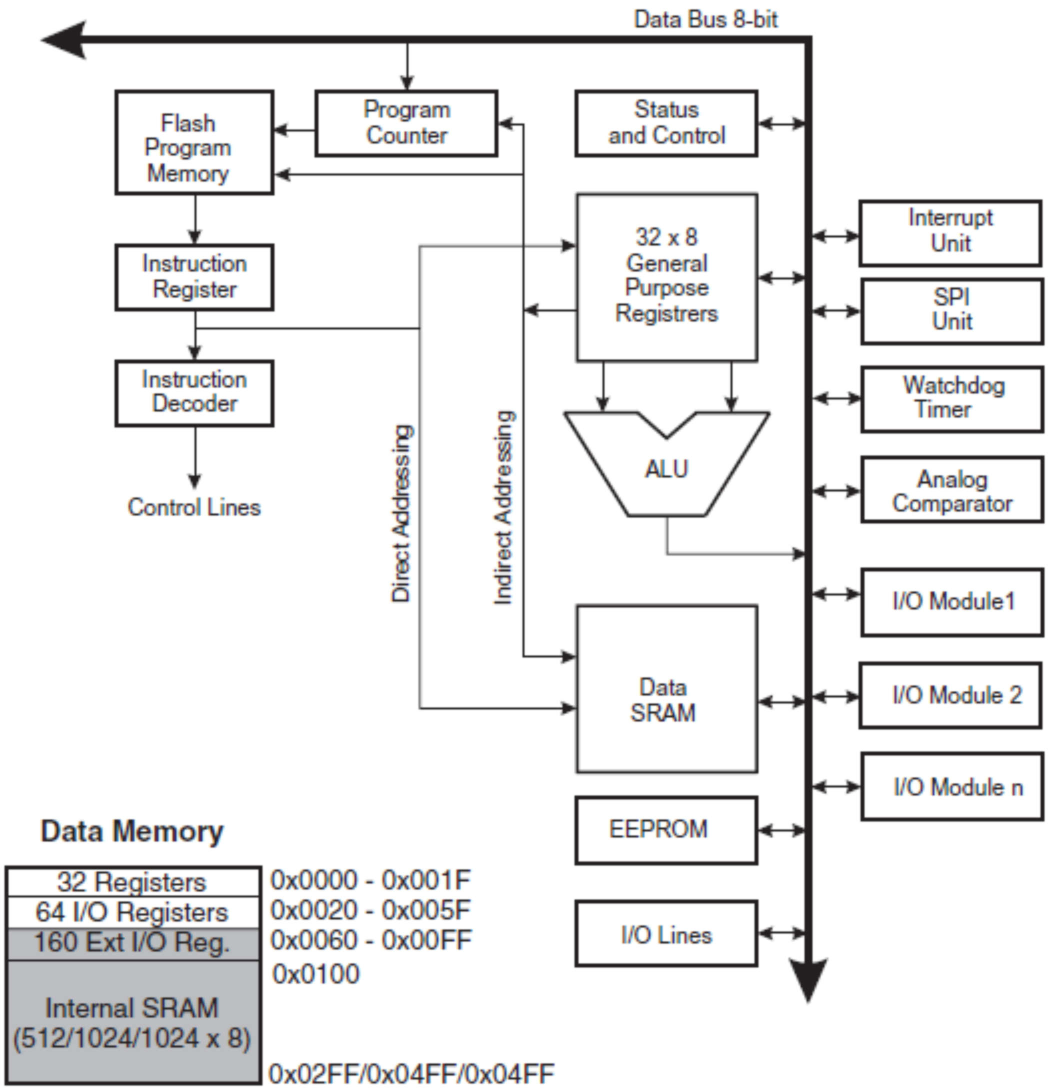
Understanding the memory architecture



What is meant by the following C code:

```
char x;
x = 0x20;
```

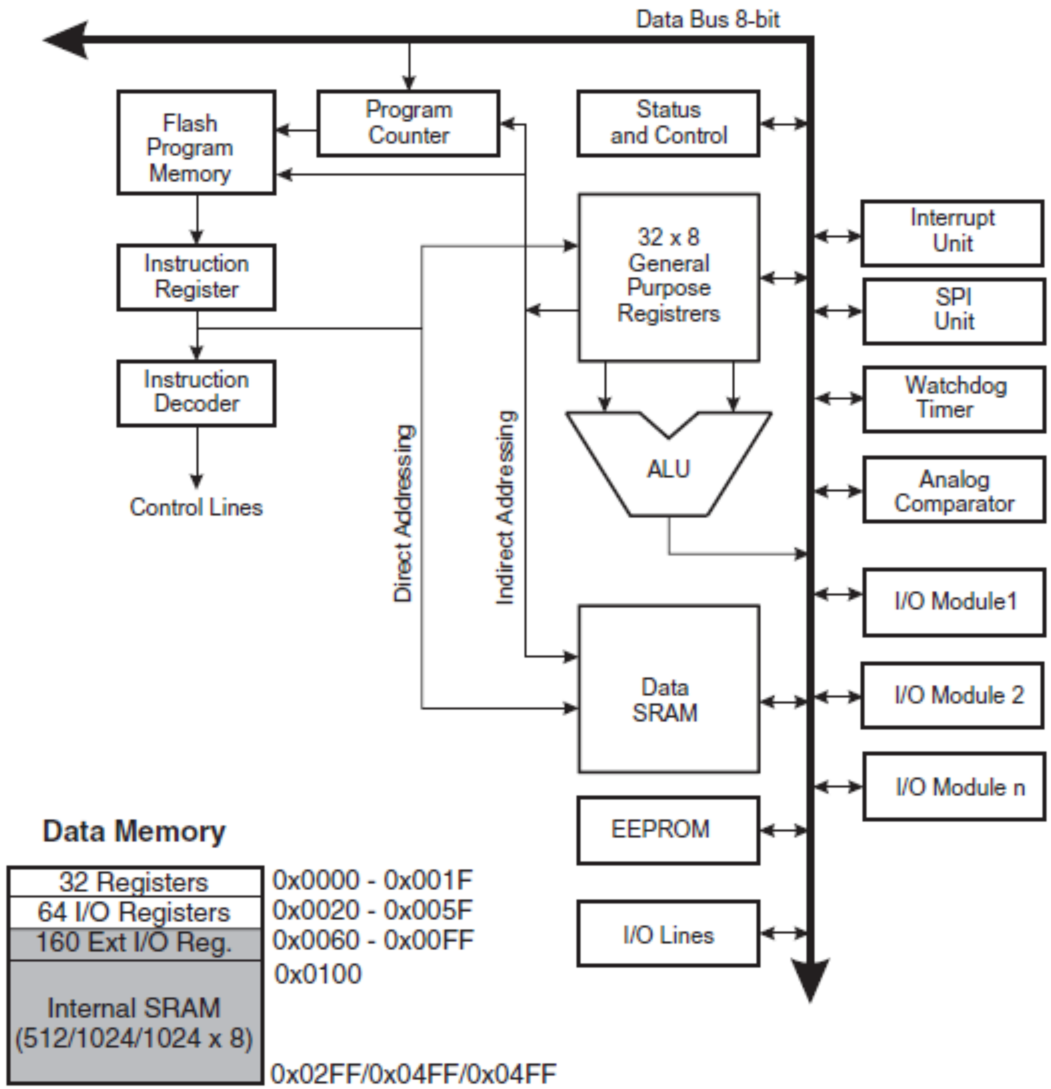
Understanding the memory architecture



What is meant by the following C code:

```
char *x;
x = 0x20;
```

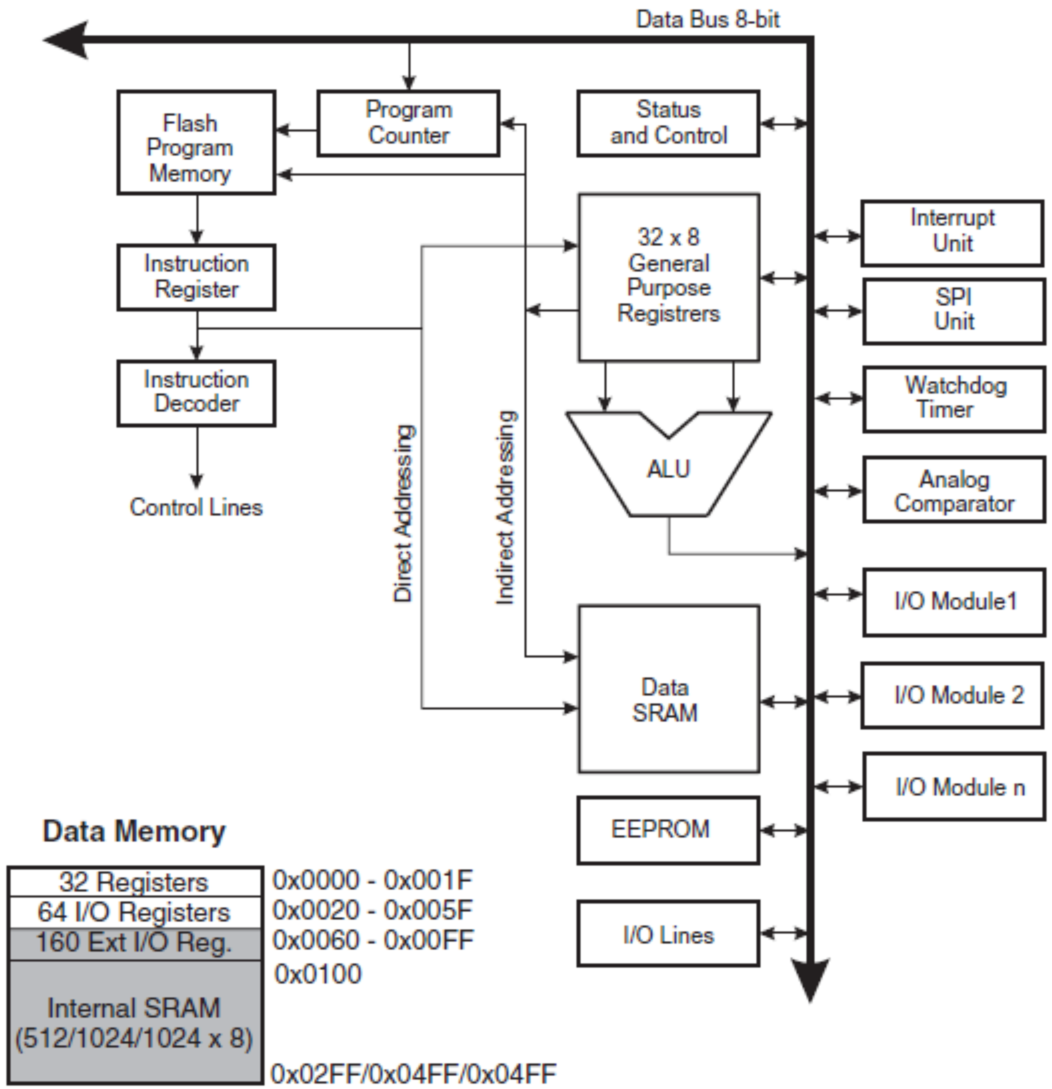
Understanding the memory architecture



What is meant by the following C code:

```
char *x, y;
x = 0x20;
y = *x;
```

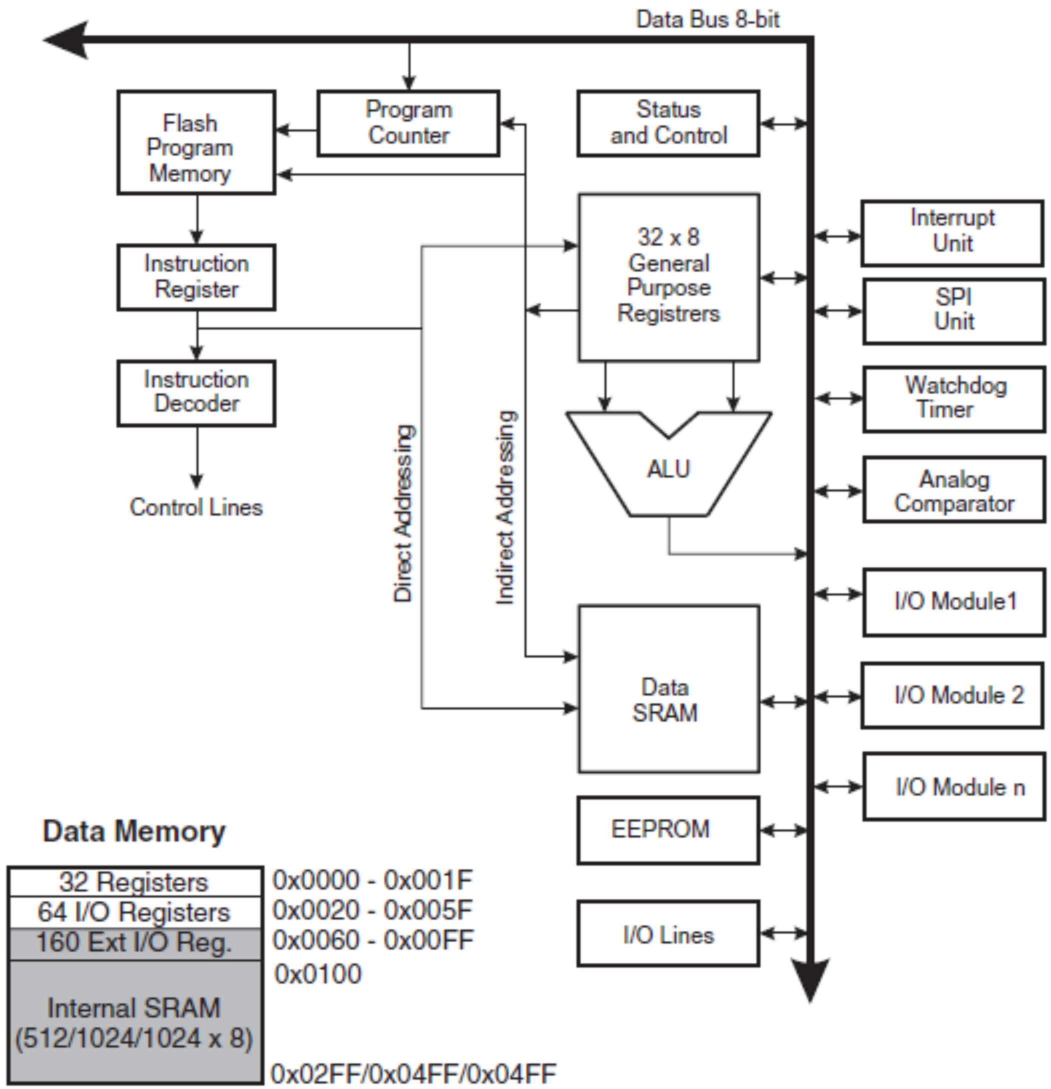
Understanding the memory architecture



What is meant by the following C code:

```
char *x, y;
x = &y;
*x = 0x20;
```

Understanding the memory architecture

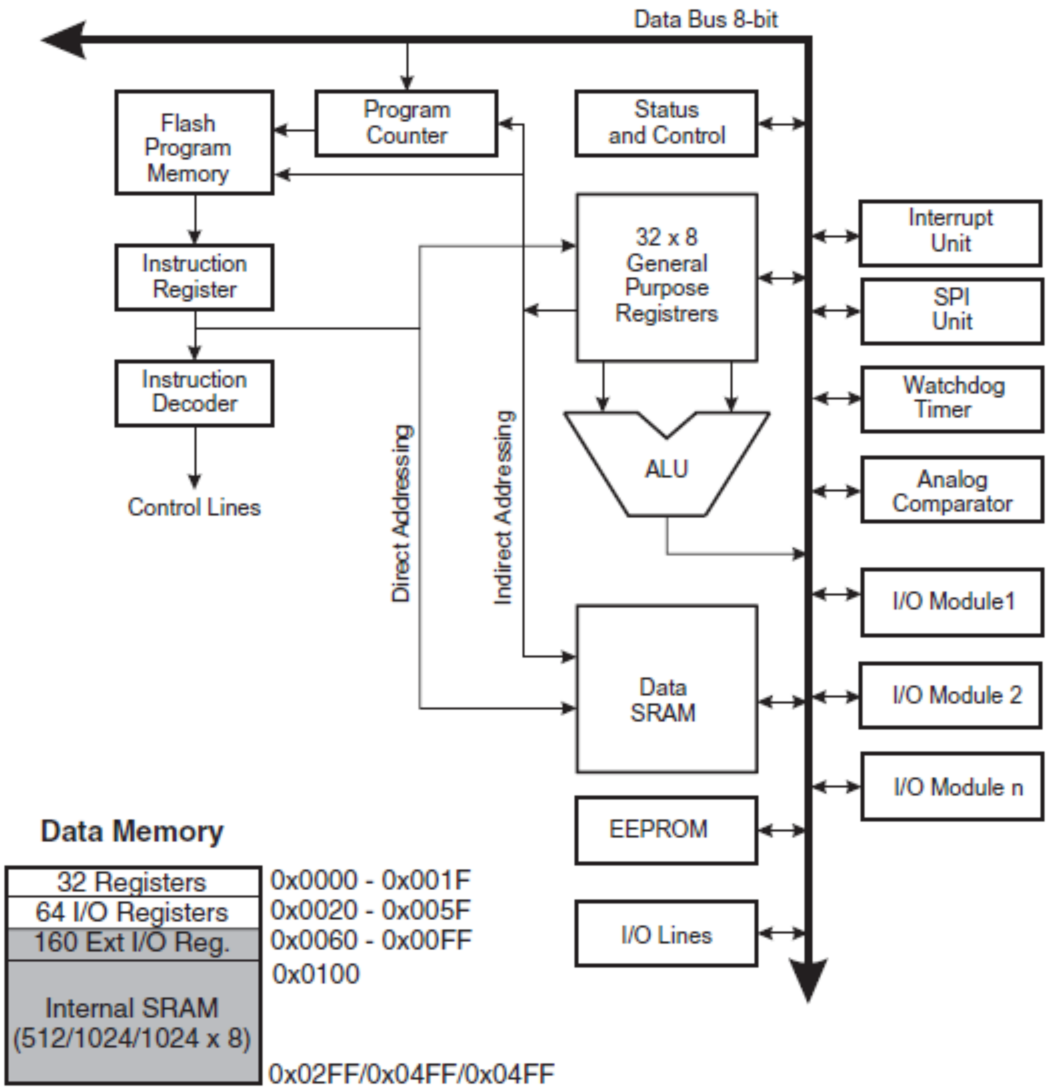


What is meant by the following C code:

```
char foo() {
      char *x, y;
      x = 0x20;
      y = *x;
      return y;
}
char z;
int main(void) {
      z = foo();
      …
}
```

Where are x, y, z in memory?

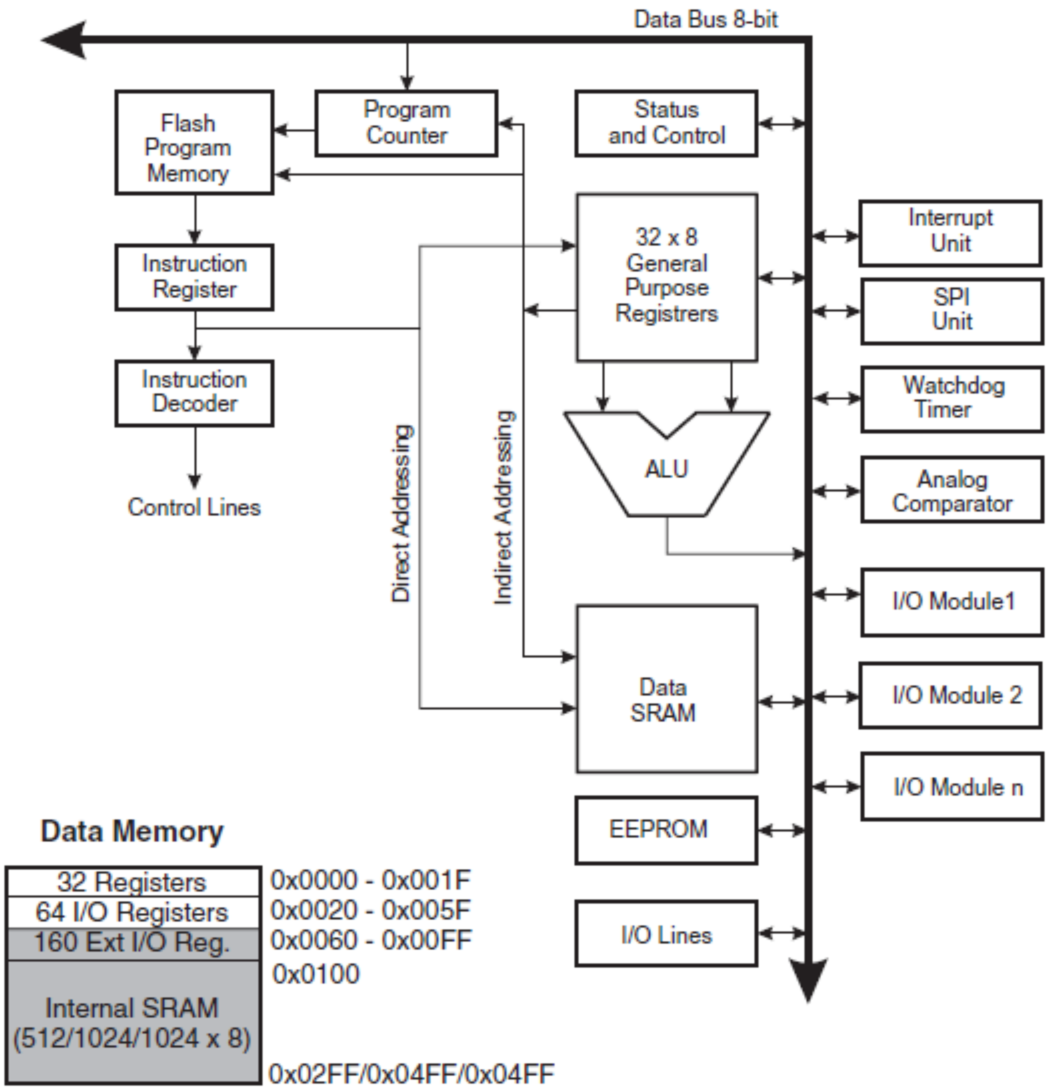
Understanding the memory architecture




```
char foo() {
    char *x, y;
    x = 0x20;
    y = *x;
    return y;
}
char z;
int main(void) {
    z = foo();
    ...
```

x occupies 2 bytes on the stack, y occupies 1 byte on the stack, and z occupies 1 byte in static memory.

Understanding the memory architecture



What is meant by the following C code:

```
char foo() {
    char y;
    uint16_t x;
    x = 0x20;
    y = *x;
    return y;
}
char z;
int main(void) {
    z = foo();
    …
}
```

Memory usage: Understanding the stack. Find the flaw in this program

```
int x = 2;

int* foo(int y) {
  int z;
  z = y * x;
  return &z;
}

int main(void) {
  int* result = foo(10);
  ...
}
```

statically allocated: compiler assigns a memory location.

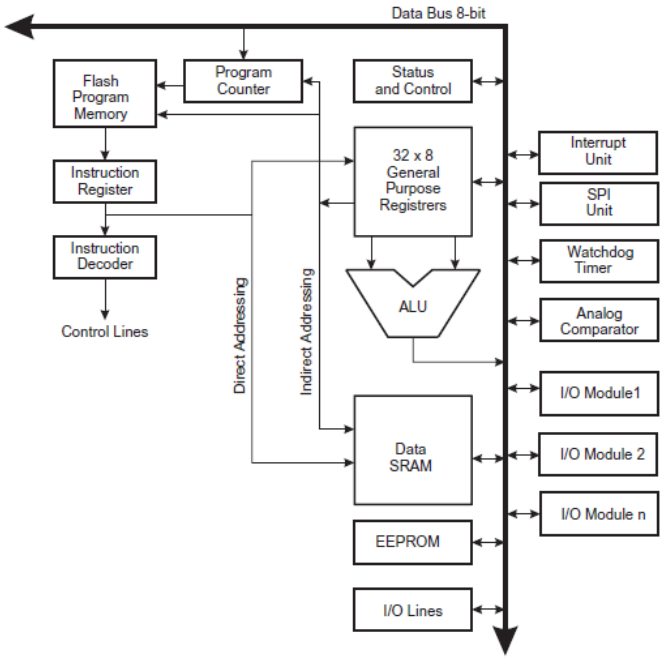
arguments on the stack

automatic variables on the stack

program counter and copies of all registers on the stack

This program returns a pointer to a variable on the stack. What if another procedure call occurs before the returned pointer is de-referenced?

Understanding the memory architecture




```
void foo(uint16_t x) {
    char y;
    y = *x;
    if (x > 0x100) {
        foo(x - 1);
    }
}
char z;
void main(…) {
    z = 0x10;
    foo(0x04FF);
    …
}
```

What is the value of z?

Data Memory

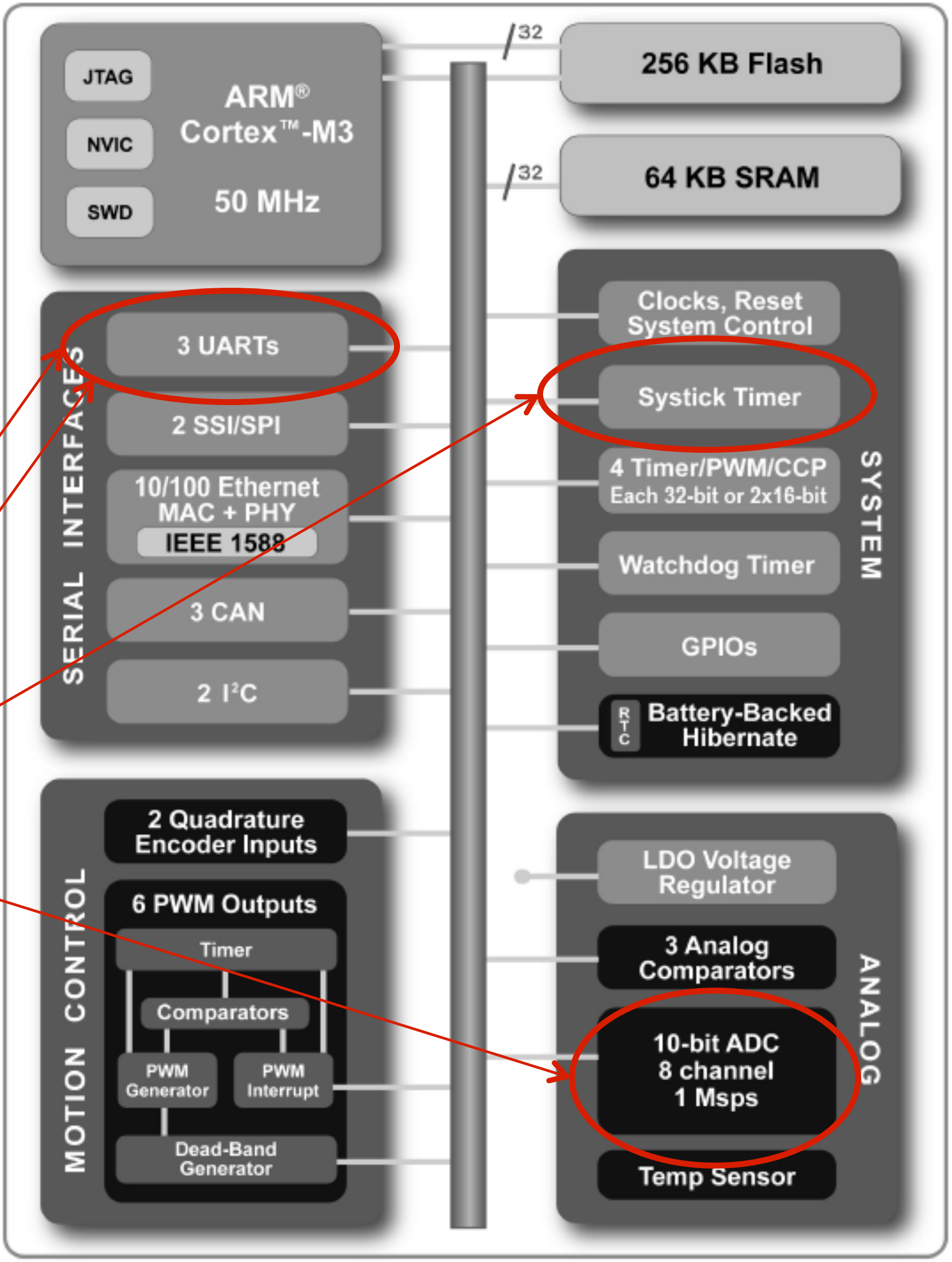
32 Registers	0x0000 - 0x001F
64 I/O Registers	0x0020 - 0x005F
160 Ext I/O Reg.	0x0060 - 0x00FF
Internal SRAM (512/1024/1024 x 8)	0x0100
	0x02FF/0x04FF/0x04FF

Luminary LM3S8962 I/O Architecture

32-bit microcomputer.

Relevant features:

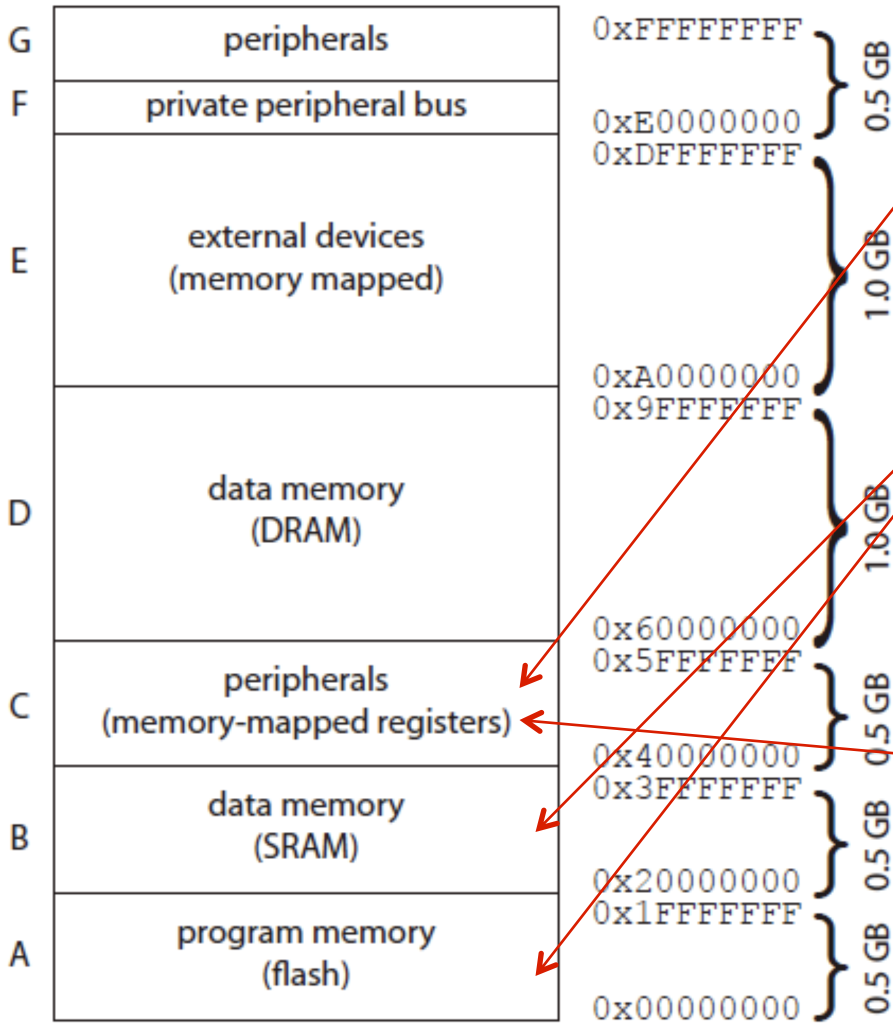
- UART interface
- ADC to accelerometer
- SysTick/other Timers
- UART to Bluetooth
- JTAG interface for programming
- *No floating point!*



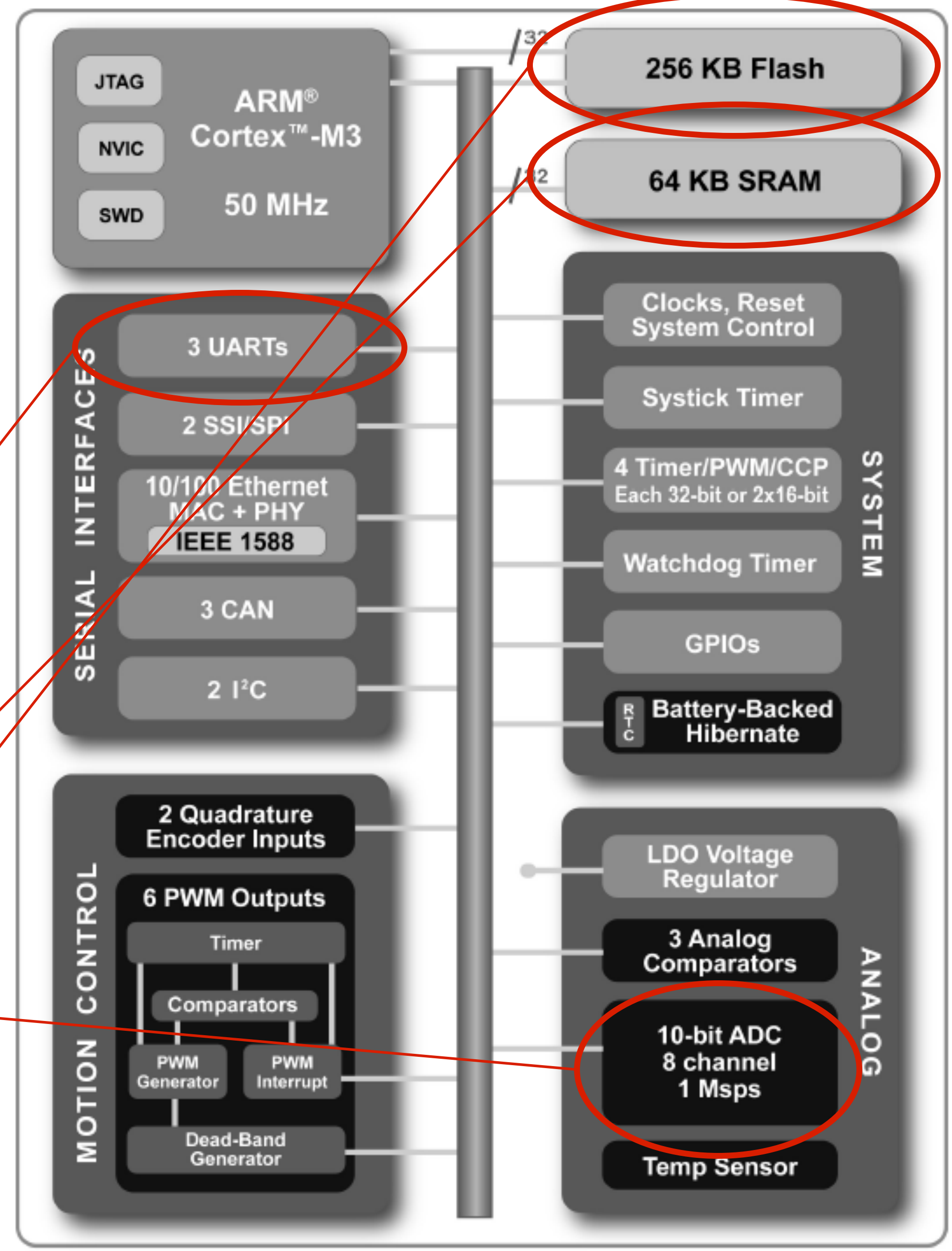
Source: Luminary Micro Datasheet

Memory Architecture

ARM Cortex M3



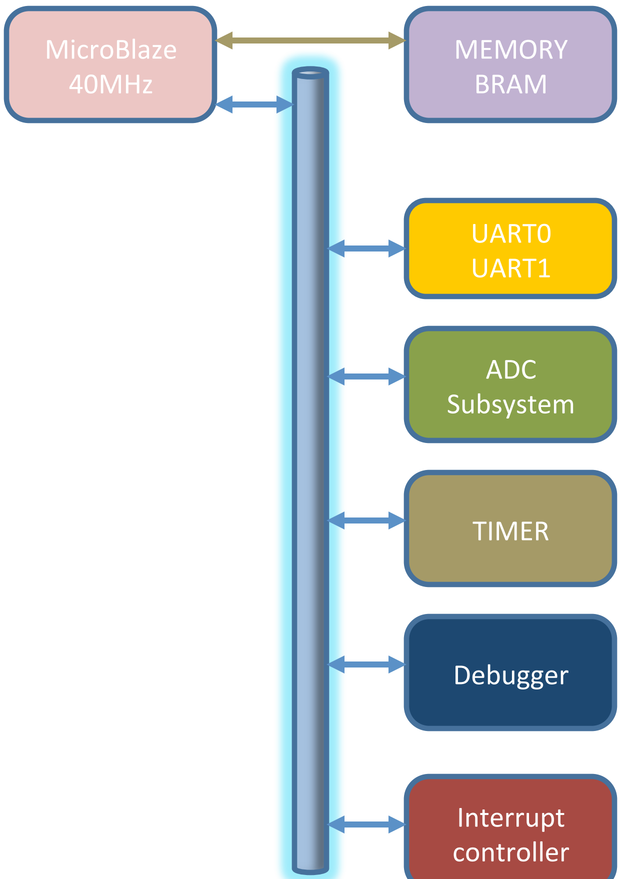
Luminary LM3S8962

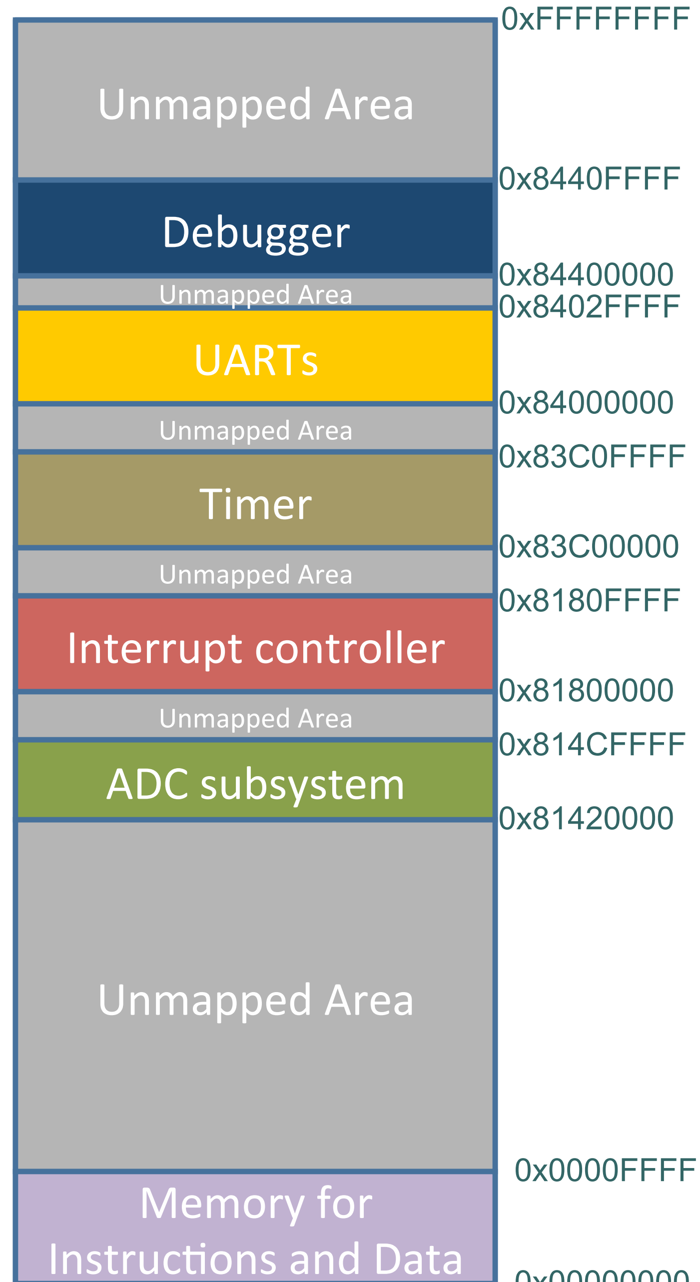


Source: Luminary Micro Datasheet

Microblaze memory map

Xilinx soft core





Thanks to Trung Tran

Heaps

An operating system typically offers a way to dynamically allocate memory on a "heap".

Memory management (malloc() and free()) can lead to many problems with embedded systems:

- Memory leaks (allocated memory is never freed)
- Memory fragmentation (allocatable pieces get smaller)

Automatic techniques ("garbage collection") typically require stopping everything and reorganizing the allocated memory. This is deadly for real-time programs.

Summary

Understanding memory architectures is essential to programming embedded systems.