EMBEDDED SYSTEM DESIGN

MEMORY

Doan Duy, Ph. D.

Email: duyd@uit.edu.vn

Objective and Content

- Overview of memory
- Memory Organization
- Example

Content

- **Overview of memory**
- Memory Organization
- Example

Role of memory in embedded systems

Traditional roles: Storage and Communication for Programs

In embedded systems:

- Communication with Sensors and Actuators
- Often much more constrained than in general-purpose computing: Size, power, reliability, etc.

→*Can be important for programmers to understand these constraints?*

Practical Issues

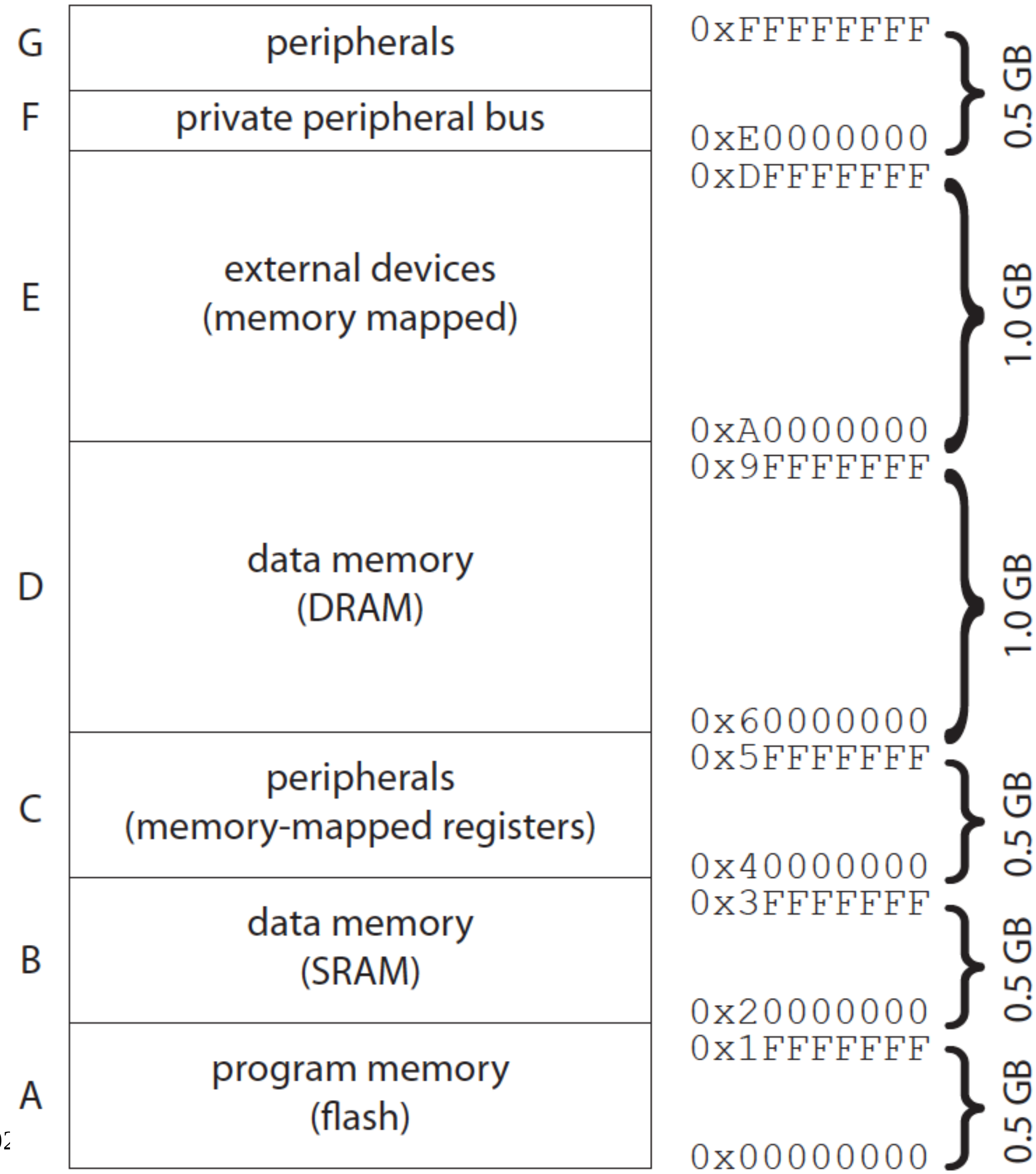
- Types of memory
  - volatile vs. non-volatile, SRAM vs. DRAM
- Memory maps
  - Harvard architecture
  - Memory-mapped I/O
- Memory organization
  - statically allocated
  - stacks
  - heaps (allocation, fragmentation, garbage collection)
- The memory model of C
- Memory hierarchies
  - scratchpads, caches, virtual memory)
- Memory protection
  - segmented spaces

Memory mapping

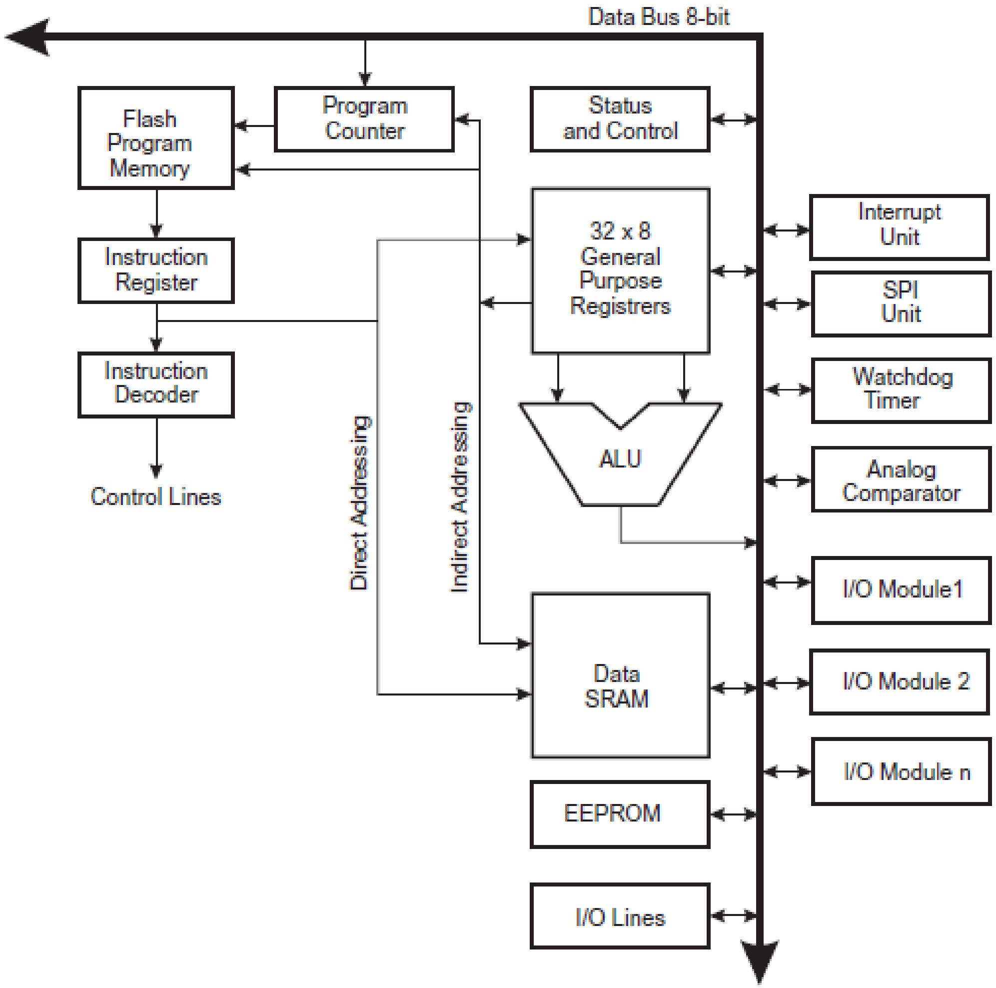
Memory Map of an ARM Cortex™ - M3 architecture

- Defines the mapping of addresses to physical memory.

- Note that this does not define how much physical memory there is!



Questions for understanding



- Why is it called an 8-bit microcontroller?
- What is the difference between an 8-bit microcontroller and a 32-bit microcontroller?
- Why use volatile memory? Why not always use non-volatile memory?

Objective and Content

- Overview of memory
- **Memory Organization**
- Example

Memory organization for Programs

- Statically-allocated memory
  - Compiler chooses the address at which to store a variable.
- Stack
  - Dynamically allocated memory with a Last-in, First-out (LIFO) strategy
- Heap
  - Dynamically allocated memory

Memory allocation (1)

- What is meant by the following C code:

```
char x;
void foo(void) {
  x = 0x20;
  …
}
```

An 8-bit quantity (hex 0x20) is stored at an address in statically allocated memory in internal RAM determined by the compiler.

Memory allocation (2)

- What is meant by the following C code:

```
char *x;
void foo(void) {
  x = 0x20;
  …
}
```

An 16-bit quantity (hex 0x0020) is stored at an address in statically allocated memory in internal RAM determined by the compiler.

Memory allocation (3)

- What is meant by the following C code:

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

The 8-bit quantity in the I/O register at location 0x20 is loaded into y, which is at a location in internal SRAM determined by the compiler.

Memory allocation (4)

Where are x, y, z in memory?

```
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.

Memory allocation (5)

- What is meant by the following C code:

```
void foo(void) {
  char *x, y;
  x = &y;
  *x = 0x20;
  …
}
```

16 bits for x and 8 bits for y are allocated on the stack, then x is loaded with the address of y, and then y is loaded with the 8-bit quantity 0x20.

Memory allocation (6)

What goes into z in the following program:

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

z is loaded with the 8-bit quantity in the I/O register at location 0x20.

Dynamically-Allocated Memory
The Heap

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

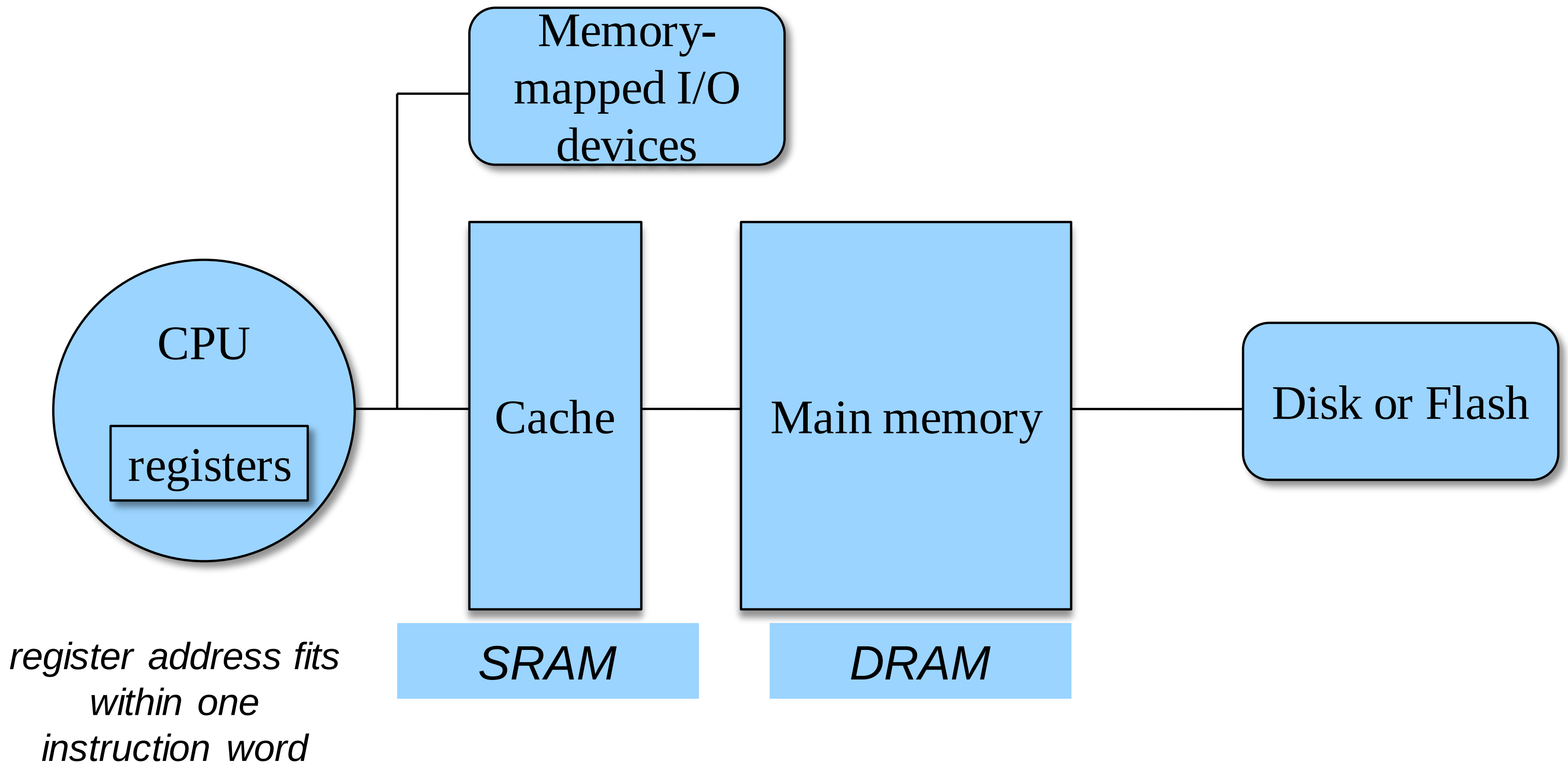
- Memory management (malloc() and free()) can lead to many problems with embedded systems:
  - Memory leaks (allocated memory is never freed)
  - Memory fragmentation (allocatable pieces get smaller)

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

Memory Hierarchies

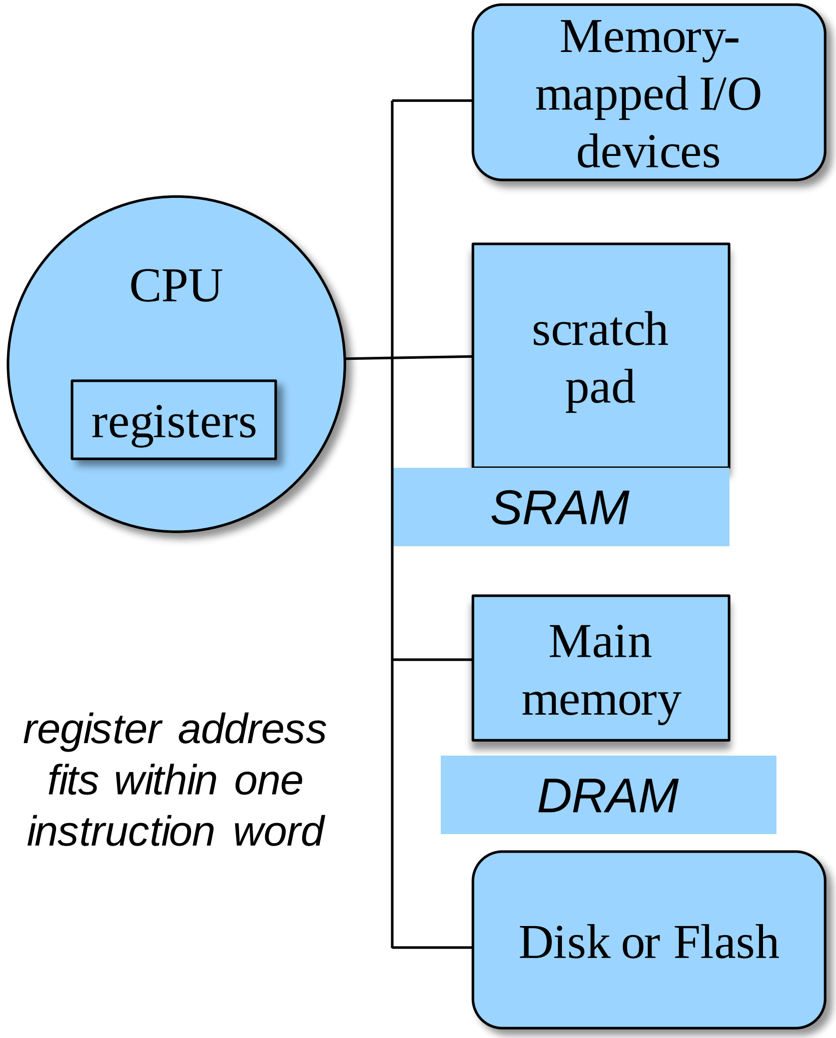
- Cache:
  - A subset of memory addresses is mapped to SRAM
  - Accessing an address not in SRAM results in *cache miss*
  - A miss is handled by copying contents of DRAM to SRAM
- Scratchpad:
  - SRAM and DRAM occupy disjoint regions of memory space
  - Software manages what is stored where
- Segmentation
  - Logical addresses are mapped to a subset of physical addresses
  - Permissions regulate which tasks can access which memory

Memory Hierarchies



Here, the cache or scratchpad, main memory, and disk or flash share the same address space.

Memory Hierarchies

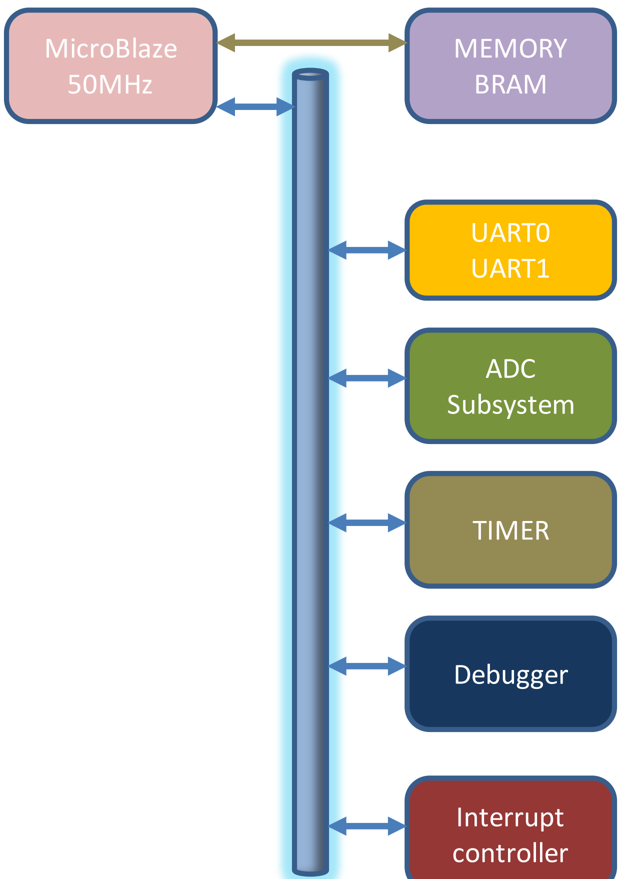


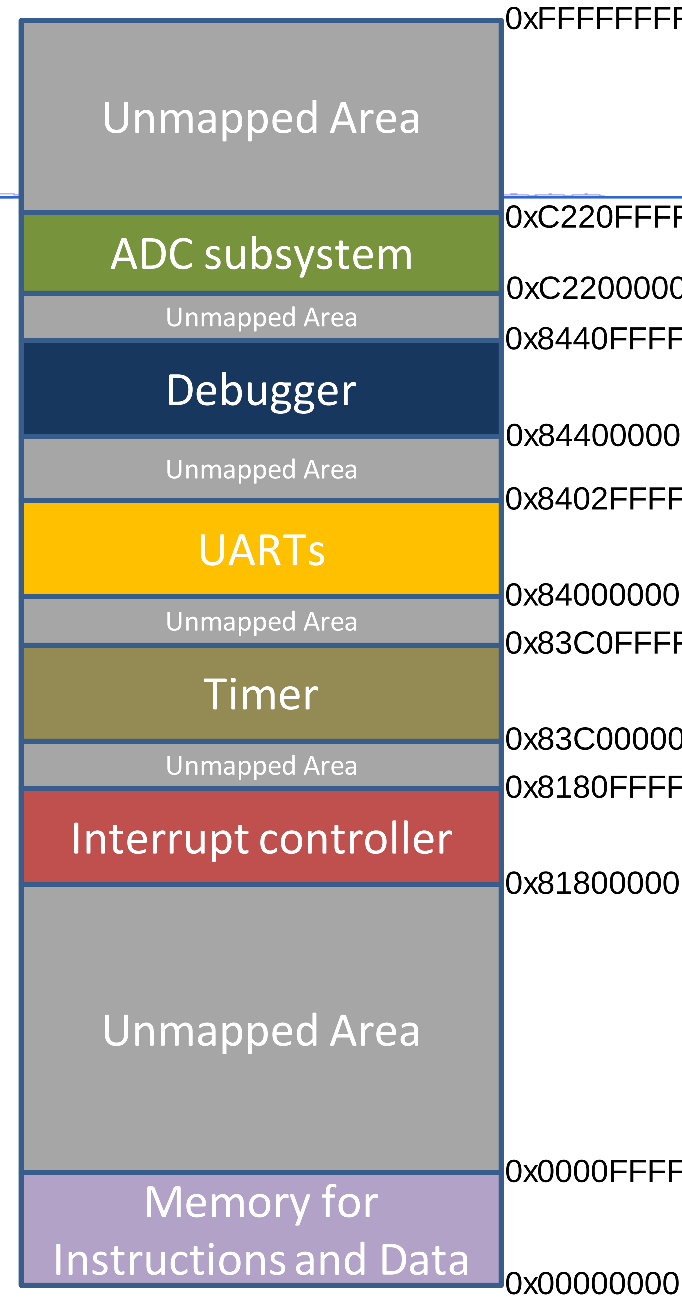
- Here, each distinct piece of memory hardware has its own segment of the address space.
- This requires more careful software design, but gives more direct control over timing.

Objective and Content

- Overview of memory
- Memory Organization
- **Example**

Berkeley Microblaze Personality Memory Map





Conclusion

Understanding memory architectures is essential to programming embedded systems!

Q&A