Memory Architecture

CS 447– Wireless Embedded Systems
Outline

• Overview
• Memory types
• Memory hierarchy
• Registers
• Memory models
• Memory protection
• Dynamic memory

What is this all about?
Overview

Memory systems have huge impact on system performance

Three sources of complexity in memory:
• Commonly necessary to mix variety of memory technologies
• Often need memory hierarchy
• Address space often divided up for access
Overview

Source of complexity #1: *mixing memory technologies*

- Have a mix of both volatile and non-volatile memory
- Embedded systems need both
- Several choices within each category (with consequences)
Overview

Source of complexity #2: *often need memory hierarchy*

- Memory w/ high capacity and low power consumption => slow
- To get good performance *at low cost*, must mix fast and slow
Overview

Source of complexity #3: *address space divided up to provide access to*-

- Various kinds of memory
- Designated addresses for interaction with other devices
- Support for common programming tasks (e.g., scratch space)
Outline

• Overview
• **Memory types**
• Memory hierarchy
• Registers
• Memory models
• Memory protection
• Dynamic memory
Memory Types

Memory is a **big deal** in embedded systems
- Choice of memory has implications: e.g.,
- Will data persist after losing power?
- How long does it take to read/write non-volatile memory?
- Are there any memory bottlenecks?
Memory Types

RAM

- μC has register file and some amount of RAM
- Individual bytes and words can be written or read quickly
- Volatile- lose content when power lost

- Arduino Uno has 2 kB of SRAM...
Memory Types

**SRAM – Static RAM**
- Faster than DRAM
- Takes up more silicon
- Holds data as long as power is maintained
- More common in μC

**DRAM – Dynamic RAM**
- Holds data for only short time
- Must be refreshed
- Timing required to refresh DRAM must be considered
- (Often used in graphic cards)
Memory Types

Non-volatile memory
• μC may need to store data when power is off

Several options:
• Battery backups to never lose power (not ideal)
• Use non-volatile memory
Memory Types

ROM
• Most basic form of non-volatile memory
• Context fixed at factory
• Useful for products that need constant program or data stored
• Data never changes
• Programs known as firmware (not as “soft” or flexible)

• Rarely used today.
Memory Types

EEPROM – Electrically-Erasable Programmable ROM
• Can write
• Write takes longer than read
• EEPROM has limited number of writes in lifetime
• Useful form of EEPROM: flash memory
• Used to store firmware and user data that must persist

• Arduino has 1 kB of EEPROM available to use
  • EEPROM.write( ... )
  • EEPROM.read( ... )
Memory Types

Flash Memory

- Convenient form of non-volatile memory

Presents challenges for embedded systems programmers
- Fast read times (though slower than RAM)
- Slow write times
- Limited number of writes over lifetime
- Not a substitute for working memory (use RAM)

- Arduino has 32 kB of flash memory available
  - `const char msg[] PROGMEM = {"hello world"};`
Memory Types

Flash Memory – two types

**NOR**
- Longer erase and write times
- Can be accessed like RAM

**NAND**
- Less expensive
- Faster erase and write times
- Block device (data must be read in chunks)
- Behaves more like secondary device
Memory Types

Disk Memory
• Non-volatile
• Store large amounts of data
• Relatively slow access times (vs RAM, EEPROM, Flash)
• Vulnerable to vibration and temperature extremes
• Difficult to put in some embedded systems
Outline

• Overview
• Memory types
• **Memory hierarchy**
• Registers
• Memory models
• Memory protection
• Dynamic memory
Memory Hierarchy

Systems often require more memory than is available on-chip.

Many processors use memory hierarchy:
- Combines different memory technologies to increase overall memory capacity.
- Must consider tradeoffs between cost, latency, and power consumption.
Memory Hierarchy

Typical:
- Small amount of on-chip SRAM
- Larger amount of off-chip DRAM
- Third level w/ larger capacity but lack random access (slow)
Memory Hierarchy

Virtual memory can be used to manage hierarchy

• But this can cause serious problems for embedded system programmers

• Need to know exactly how long memory access will take

• Need to understand memory more deeply
Outline

• Overview
• Memory types
• Memory hierarchy
• Registers
• Memory models
• Memory protection
• Dynamic memory
Registers

• Tightly integrated with memory in processor
• Each register stores a **word**
• Size of **word** is key property: e.g., 8-bit, 32-bit, 64-bit
• E.g., Arduino Uno uses an 8-bit ATmega328 processor

• Can be implemented using flip-flop gates in processor circuitry
• Or as single memory bank in SRAM
Registers

Number of registers usually small
• Not due to cost of register hardware
• But cost of bits in each instruction

Instruction set architecture only access one, two, or three registers at a time
• Reduce # of bits used to identify registers
• E.g., 16 registers => 4 bits needed, leaves only 4 bits for instr.
Outline

• Overview
• Memory types
• Memory hierarchy
• Registers
• Memory models
• Memory protection
• Dynamic memory
Memory Models

• Defines how memory is used by programs

• Hardware, operating system (if any), programming language and compiler all contribute to memory model
Memory Models

• At minimum, memory model must define range of memory addresses available to program

• In C, addresses stored in pointers (E.g., 32-bit unsigned int)
  • ~4,000,000,000 addresses
Memory Models

Memory addresses refer to bytes (duh)

When writing program that directly modifies bytes at memory addresses, must consider:

• **Alignment** of data: four-byte ints start at multiples of 4
• **Byte order**: big vs. little endian (Arduino is little endian)
Memory Models

Stacks

• Dynamically allocated region of memory for running program
• LIFO - last in, first out
• **Stack pointer** (register) contains address to top of stack
  • Items pushed to stack (e.g., function call), stack pointer++
  • Items popped from stack (e.g., return) stack pointer--
Memory Models

Stacks

Often used to implement function calls:
• Push: address of next instruction on stack
• Push: current values of some or all registers
• Push: arguments to procedures
• Then change program counter to function code

• Data pushed to stack know as stack frame
Memory Models

Stacks

In embedded systems, disastrous if / when stack pointer gets incremented beyond memory allocated

• Stack overflow- overwrite memory used for something else
• Unpredictable results
• Need to limit use of stack
• E.g., NO RECURSION!!!
Memory Models

What’s wrong with this code?

```c
int* foo(int a) {
    int b;
    b = a * 10;
    return &b;
}

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

• Overview
• Memory types
• Memory hierarchy
• Registers
• Memory models
• Memory protection
• Dynamic memory
Memory Protection

PCs have hardware to protect memory space
• Trying to read / write inaccessible memory
• Cause of segmentation faults

µCs do NOT have such hardware (typically)
• Seg fault on Arduino? Ha!
• Device will behave unpredictably
• If you’re lucky, device will reset itself.
Outline

• Overview
• Memory types
• Memory hierarchy
• Registers
• Memory models
• Memory protection
• Dynamic memory
Dynamic Memory

Most μCs provide mechanisms for dynamic memory allocation
• E.g., can use malloc( ) and new in Arduino

BUT... dynamic memory allocation is NOT recommended for embedded systems.
• Programmer should know memory requirements
• Embedded software does not require as much flexibility
• There’s typically NO NEED to allocate memory dynamically
Dynamic Memory

What if you used dynamic memory in μCs?

What could go wrong?

- Remember, virtual memory is not implemented (typically)
- A system could run out of memory
- Remember- dynamic memory is NOT recommended in CPS
- Make everything static
- Pre-allocate everything you’ll need