# Embedded Systems

**Arijit Mondal**
Dept. of Computer Science & Engineering
Indian Institute of Technology Patna
arijit@iitp.ac.in

# Memory

# Memory

- Different kind of memories are usually required for embedded systems
  - Volatile, Non-volatile
- Hierarchical design of memory is preferred
  - Trade-off between performance and cost
- Address space of processor is divided to provide different kind of services
  - Support for program, interaction through I/O devices

# Memory technology

- RAM - random access memory
  - Read and write are relatively fast
  - Static RAM (SRAM) is faster than Dynamic RAM (DRAM)
  - DRAM holds data for short time and need to be refreshed periodically
  - SRAM holds data as long as power is maintained
  - SRAM and DRAM are volatile in nature
  - SRAM takes more silicon and it is costly
  - Refresh cycle in DRAM can introduce variability in access time

# Non-volatile memory

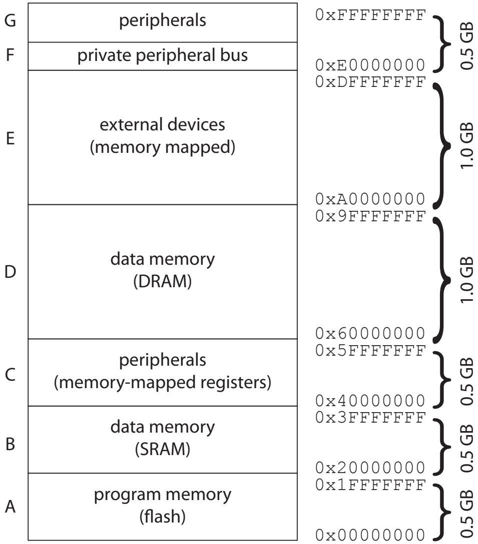
- Memory is not lost when power is withdrawn
- Early form of non-volatile memory is magnetic tape
- Read only memory (ROM) - content is written once only
  - Firmware
- EEPROM - electrically erasable programmable ROM
  - Write time is usually higher compared to read time
  - Flash memory - write time is high
    - Not a good substitute of working memory
  - NAND flash - less expensive, faster erase and write time, reads a data block (100-1000 bits) at a time
  - NOR flash - longer erase and write time, can be accessed like a RAM
- Flash can only be written and erased bounded number of time
- Disk memory is non-volatile, good secondary storage - CD, DVD

# Memory hierarchy

- Processors use different kind of memories to increase overall capacity while optimizing cost, latency, energy consumption
  - Typically small amount of on-chip SRAM and large amount of off-chip DRAM are used
- For an application programmer, it is not necessary to know fragmented form of memory
- Virtual memory
  - Physical address translation is done by OS and/or hardware
  - Translational look aside buffer

# Memory map

- **Harvard architecture - separate program and data memory**
- **Von-Neumann architecture - stores program and data in the same memory**



# Register file

- Very limited amount usually some power of 2
- Cost is determined by the instruction word
- ISA determines how many registers are accessed simultaneously

# Scratchpads and caches

- **Intermediate memory is accessed before actual memory**
- **If program is responsible for read and write from this closer memory then it is called scratchpad**
- **If the hardware manages such read/write then it is known as cache**
- **Architecture typically support much larger address space than physical memory of the processor**
- **Processor may have memory management unit**
- **Cache miss, page fault**

# Basic organization of cache

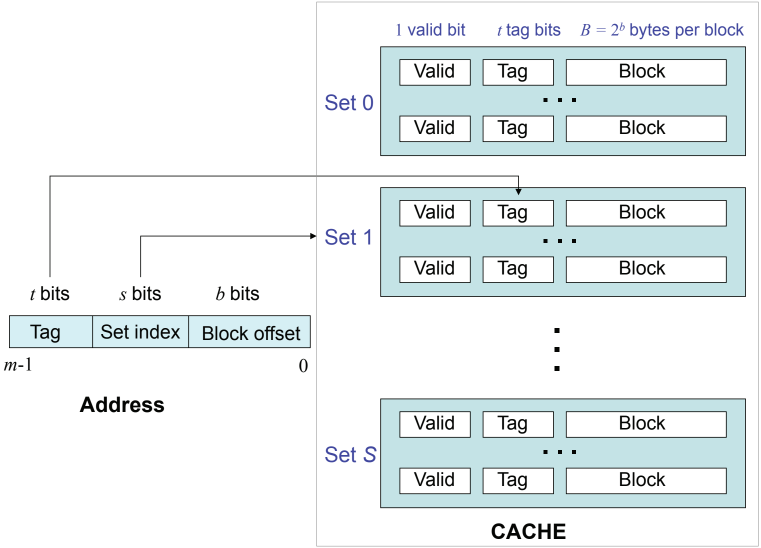



image source: Introduction to Embedded Systems book

# Memory address

- 32 bit architecture can represent address 0 to $2^{32} - 1$
- Each address refers to a byte
  - In C char is 1 byte, int is at least 2 bytes (4 in general)
- Alignment - int occupies 4 consecutive bytes starting at an address that is multiple of 4
  - In hexadecimal notation - 0, 4, 8, C
- Byte order - little vs big endian
  - Intel, ARM - little endian, IBM, PowerPC - big endian
  - Byte order matters in network protocol

# Stack

- A region of memory that is dynamically allocated
- Stack pointer contains the memory address of the top of the stack
  - When an item is pushed, stack pointer is incremented
  - When an item is popped, stack pointer is decremented
- Typically used for implementation of procedure/function call
- Stack overflow

# Dynamic memory allocation

- A program can request to operating system for additional memory at any time
- Such memory is allocated in heap
  - It tracks which portion of memory used by which application
- Garbage collector - a task runs periodically or when memory gets tight that frees the memory that no longer referred within the program
- Memory fragmentation
- Memory leak
- Real time issue with garbage collector

# C program

```
int x = 2;

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

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