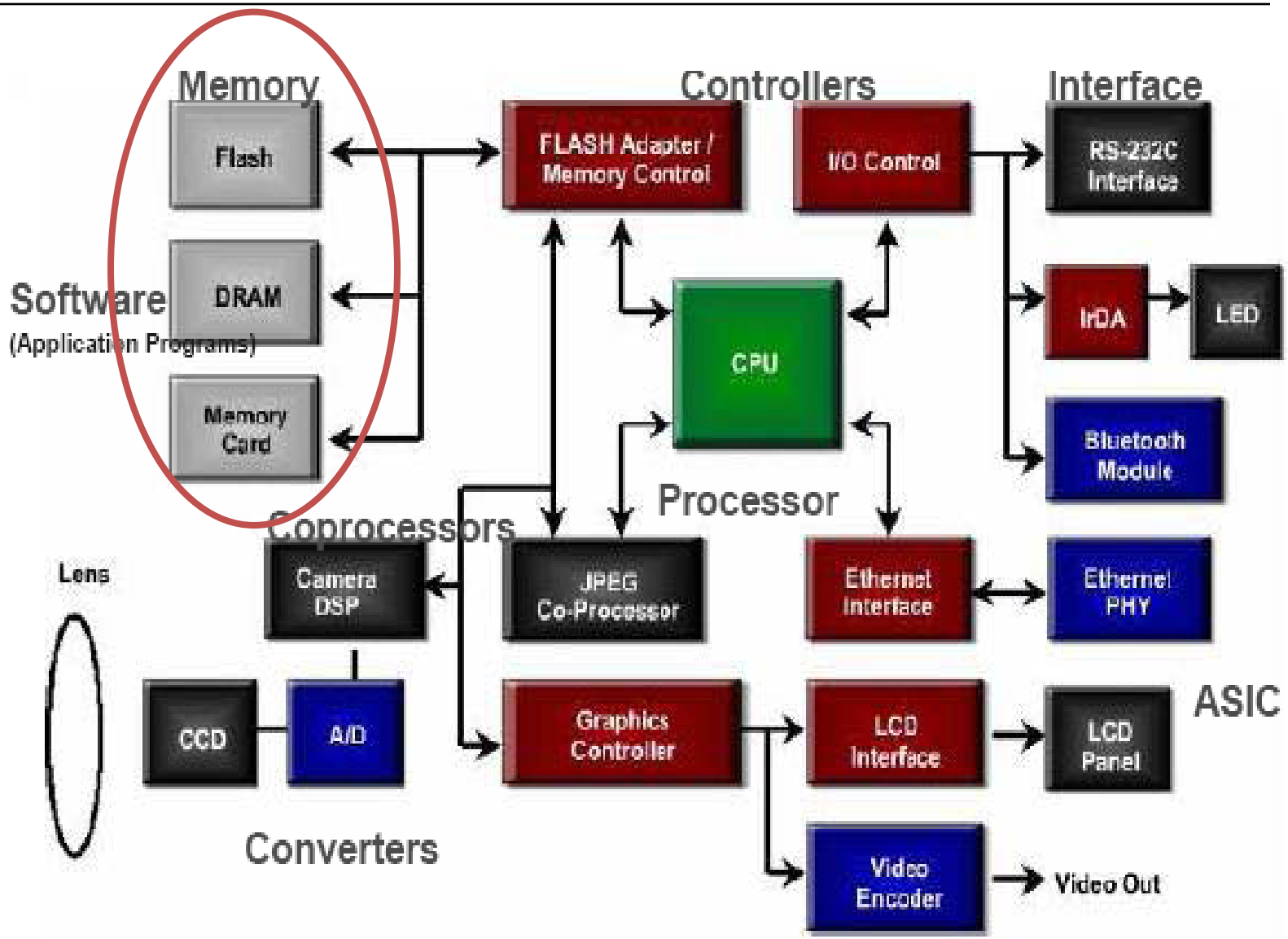
Embedded Memory Architectures and Management

Baris Aksanli
Department of Electrical and Computer Engineering
San Diego State University

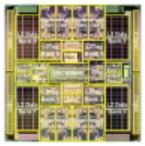
Outline

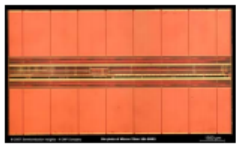
- Traditional/embedded memory hierarchy
- Memory types
- SRAM/DRAM
- ROM/EPROM/EEPROM/Flash
- Memory types in AVR

Hardware platform architecture



Traditional Memory Hierarchies



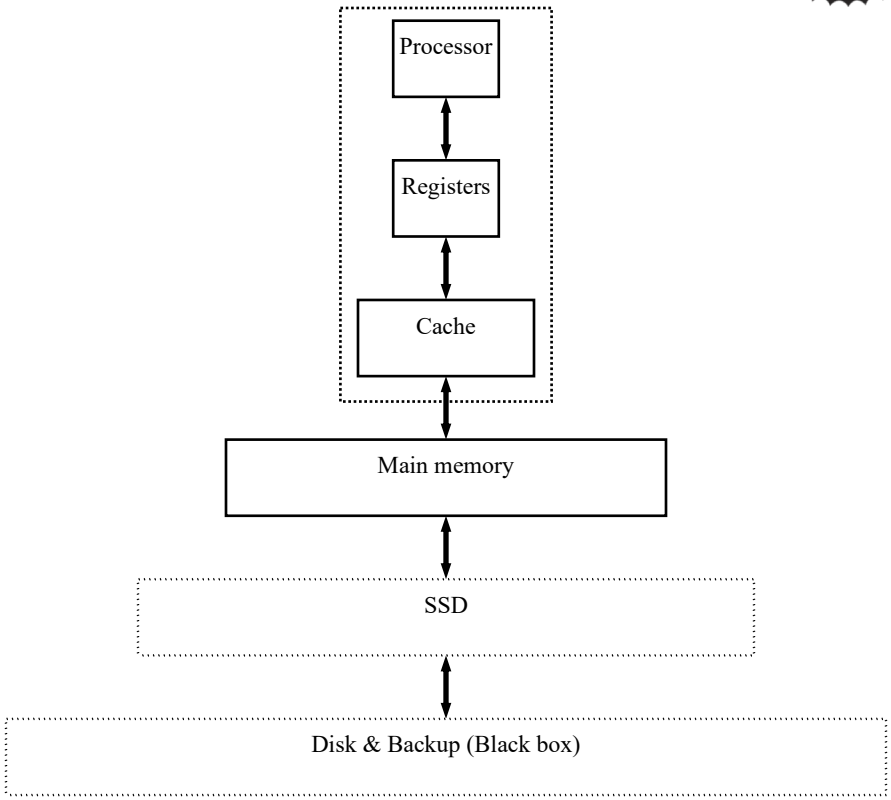


	On-chip memory (SRAM)	Off-chip memory (DRAM)	Solid State Disk (Flash Memory)	Secondary Storage (HDD)
Latency: (Cycles)	1~30	100~300	25000~2000000	>5000000

- Why SRAM as Cache? **Speed**
- Why DRAM as main memory? **Density**
- Why Flash as SSD? **Non-volatility & speed**
- Why HDD as Secondary storage? **Price & density**

Embedded Memory Hierarchy

- Registers
  - Very fast, next to ALU, power hungry
- Cache
  - Small, expensive, fast memory stores a copy of likely accessed parts
  - L1, L2, L3
- Predictability
  - Scratchpad memory
- Main memory
  - Large, inexpensive, slower
- Permanence
  - Non-volatile memories



Memory Volatility

- Volatile:
  - Loses contents when power is removed
  - Used for temporary storage of changing values:
    - Variables
    - Stacks
- Non-Volatile:
  - Retains contents after power loss
  - Used for permanent storage of:
    - Programs
    - Constants
    - Look-up tables

Read/Write vs. Read Only

- Read/Write
- Read & Write with equal ease and speed
- Application:
  - Temporary storage
  - Usually Volatile

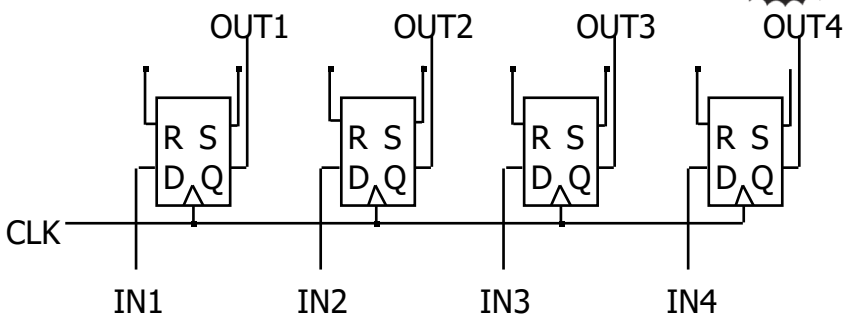
- Read Only
- New data cannot be written as easily or quickly as it is read
- Application:
  - Program storage
  - Constants
  - Non-volatile

Access Methods

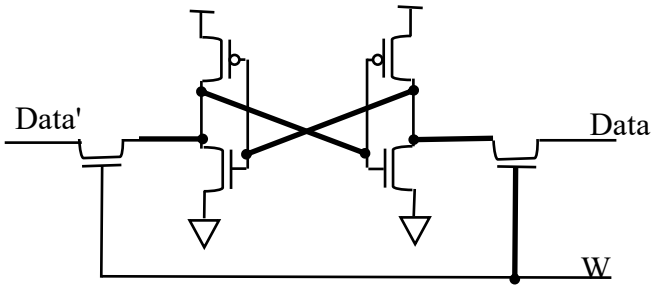
- Random access
  - RAM, DRAM, NOR Flash, byte access
- Sequential access
  - NAND flash, block/page access
- Direct (Hybrid random/sequential) access
  - Select track, rotation to sector access
  - Magnetic disk
  - Optical disk

Volatile Memory

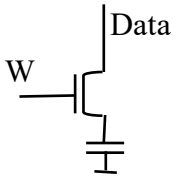
- Register file
  - Fastest
  - But biggest size – built from D-FFs
- SRAM: Static RAM
  - Memory cell uses flip-flop to store bit
  - Requires 6 transistors
  - Holds data as long as power supplied
- DRAM: Dynamic RAM
  - Memory cell uses MOS transistor and capacitor to store a bit
  - DRAM scaling limit is around 55 nm, at which point the charge stored in the capacitor is too small to be detected
  - More compact than SRAM
  - "Refresh" required due to capacitor leak
    - word's cells refreshed when read
  - Slower to access than SRAM



SRAM



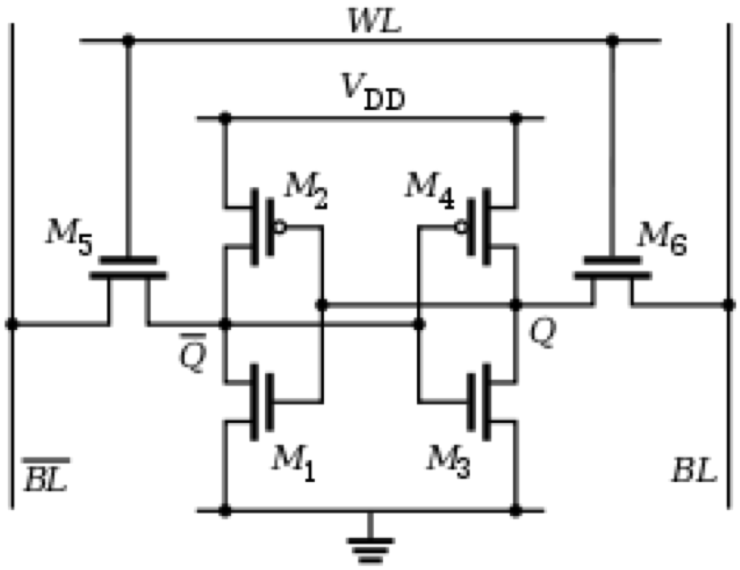
DRAM



Static RAM (SRAM)

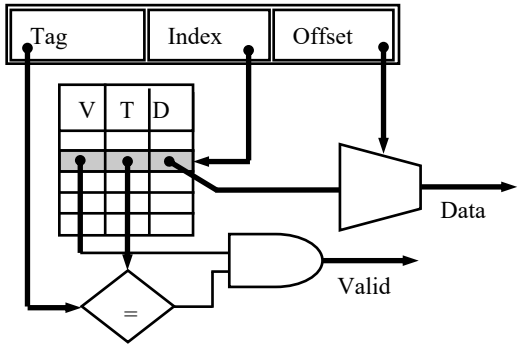
- Each bit stored by a flip-flop
- Volatile - need power to retain data
- Maintains data as long as power is applied
- Requires ~4-6 transistors per bit
- Can be very low power when inactive
  - Typically <1uA at 3 V to retain contents

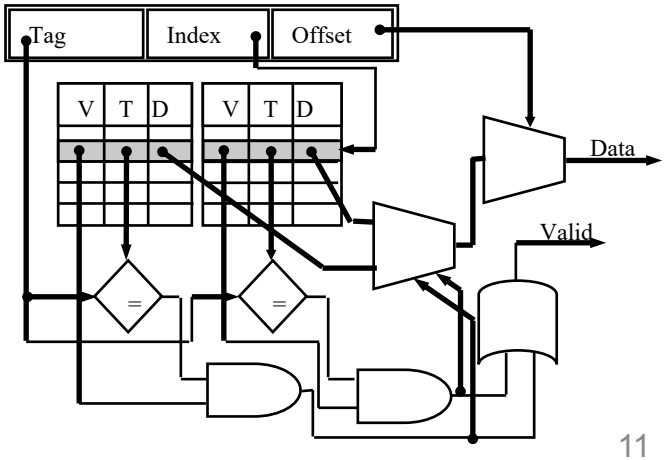
Example SRAM design



Cache

- **Designed with SRAM, usually on same chip as processor**
- **Cache operation:**
  - Request for main memory access (read or write)
  - First, check cache for copy
    - cache hit
    - cache miss
- **Design choices**
  - cache mapping
    - Direct - each memory location maps onto exactly one cache entry
    - Fully associative – anywhere in memory, never implemented
    - Set-associative - each memory location can go into one of n set
  - write techniques
    - Write-through - write to main memory at each update
    - Write-back – write only when "dirty" block replaced
  - replacement policies
    - Random
    - LRU: least-recently used
    - FIFO: first-in-first-out





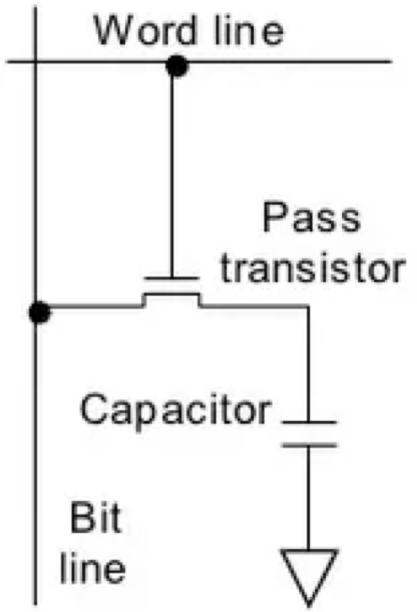
Cache impact on system performance

- Most important parameters in terms of performance:
  - Total size of cache (data and control info – tags etc)
  - Degree of associativity
  - Data block size
- Larger caches -> lower miss rates, higher access cost
  - Average memory access time (h1=L1 hit rate, h2=L2 hit rate)
    - $t_{av} = h_1 t_{L1} + (1-h_1)h_2 t_{L2} + (1- h_1)(1- h_2)t_{main}$
  - e.g., if miss cost = 20
    - 2 Kbyte: miss rate = 20%, hit cost = 2 cycles, average access time 5.6 cycles
    - 4 Kbyte: miss rate = 10%, hit cost = 3 cycles, access 4.7 cycles
    - 8 Kbyte: miss rate = 8%, hit cost = 4 cycles, access 5.28 cycles

Dynamic RAM (DRAM)

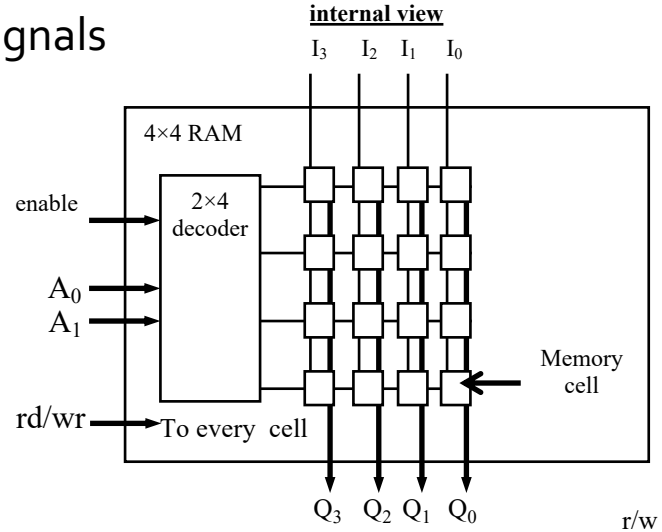
- Data stored as charge on a leaky capacitor
- Charge determines state of switch
- Charge leaks away
- Must be refreshed
- Only requires one transistor per bit

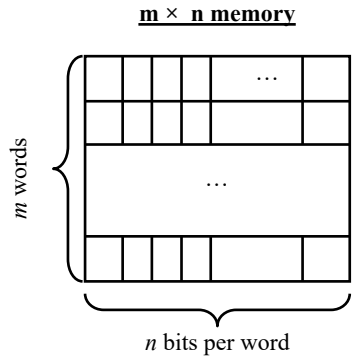
Example DRAM design

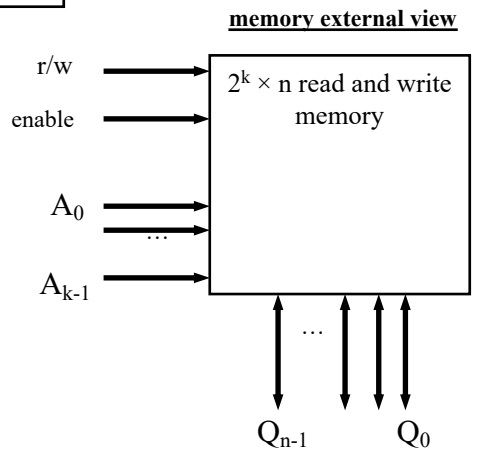


RAM organization

- Stores large number of bits
  - *m* x *n*: *m* words of *n* bits each
  - k = $\text{Log}_2(m)$ address input signals
  - or *m* = 2^k words
  - e.g., 4,096 x 8 memory:
    - 32,768 bits
    - 12 address input signals
    - 8 input/output data signals
- Memory access
  - r/w: selects read or write
  - enable: read or write only when asserted
  - multiport: multiple accesses to different locations simultaneously







Types of ROM

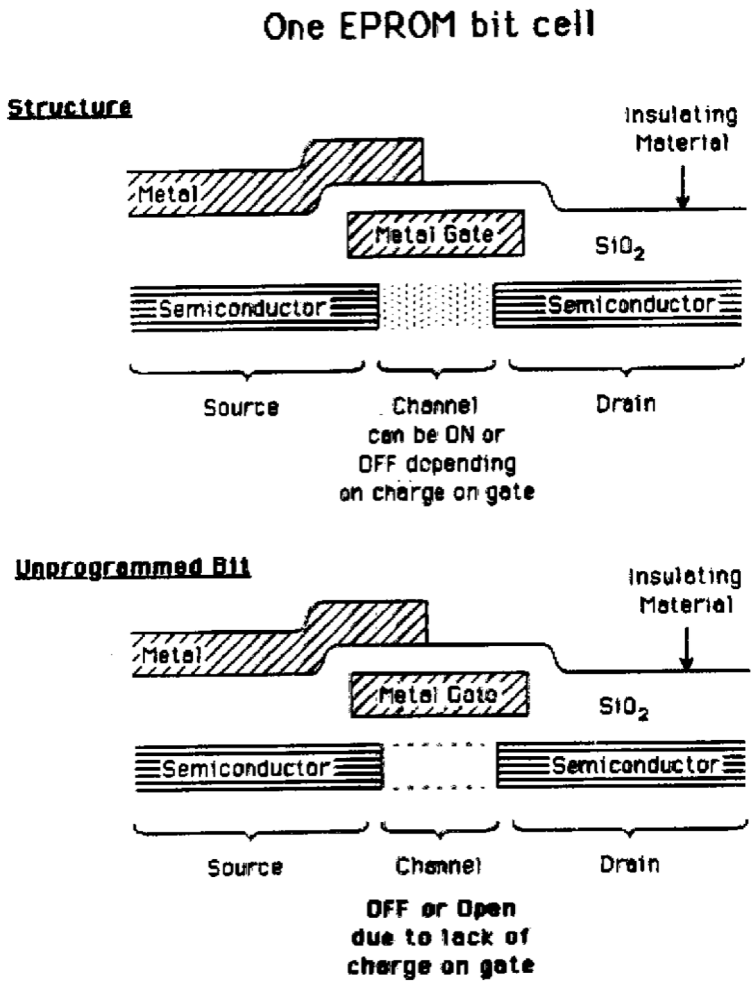
- Factory programmed "Mask" ROM
  - Defined at time ROM is manufactured
  - Cannot be changed
  - Very low unit cost
- User Programmable ROM (PROM)
  - Can be programmed just prior to use
  - Special device required for programming:
    - PROM Programmer

Floating Gate NV Memories

- EPROM, EEPROM
  - Read/write/erase single locations
- Flash Memory
  - NOR – Code storage, fast, random access
  - NAND – Higher density, lower cost, sequential
- Limited number of write/erase cycles
- Flash erasure in blocks or pages
- Flash read/write in bytes

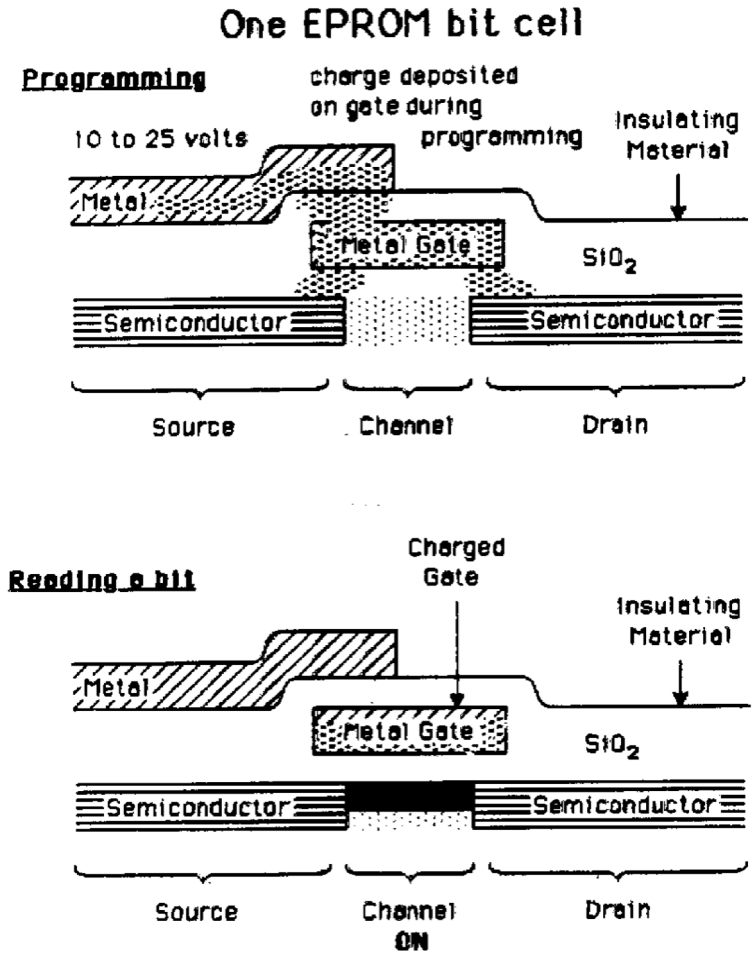
Erasable PROM (EPROM)

- Programmable
  - Write electrically
  - Using a programmer
- UV erasable (obsolete)
  - Shine UV light on chip
  - Erase chip via window
  - Can be re-programmed
- Erases the entire chip
- Uses 1 floating gate transistor



EPROM Programming

- Programming:
  - High voltage strands charge on a floating transistor gate
  - Byte programmable
- Data read back:
  - Like DRAM, charge is stored on gate
  - Turns channel on



Electrically Erasable PROM

- EEPROM or E²PROM
- Program and erase electrically
- Fowler-Nordheim Quantum Electron Tunneling
- Byte programmable ***and*** erasable
- Uses 1 floating gate and 1 normal transistor
- Often serial I²C or SPI a.k.a. "Serial PROM"

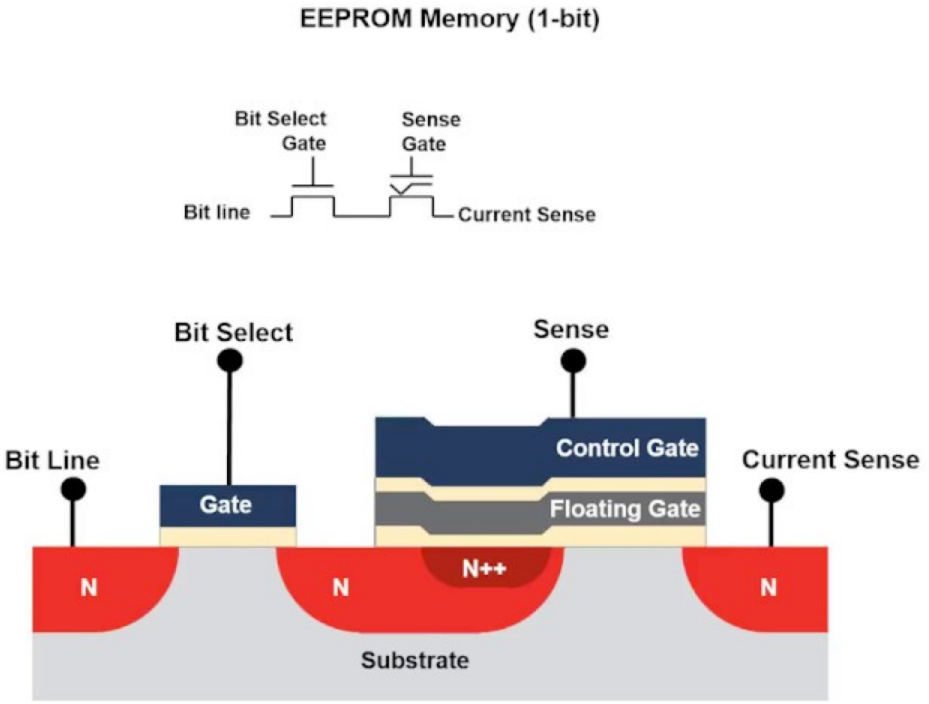


Figure source: Microchip

Flash Memory

- Hybrid of EEPROM and EPROM
- Uses Fowler-Nordheim Electron Tunneling like EEPROM
- Byte programmable but block erasable

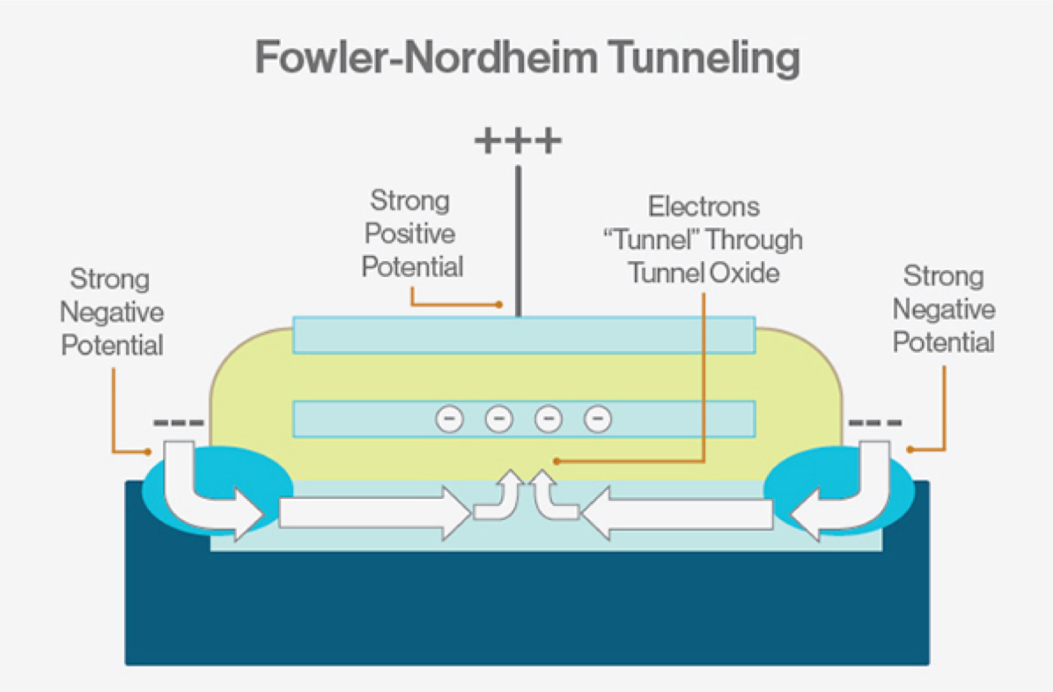
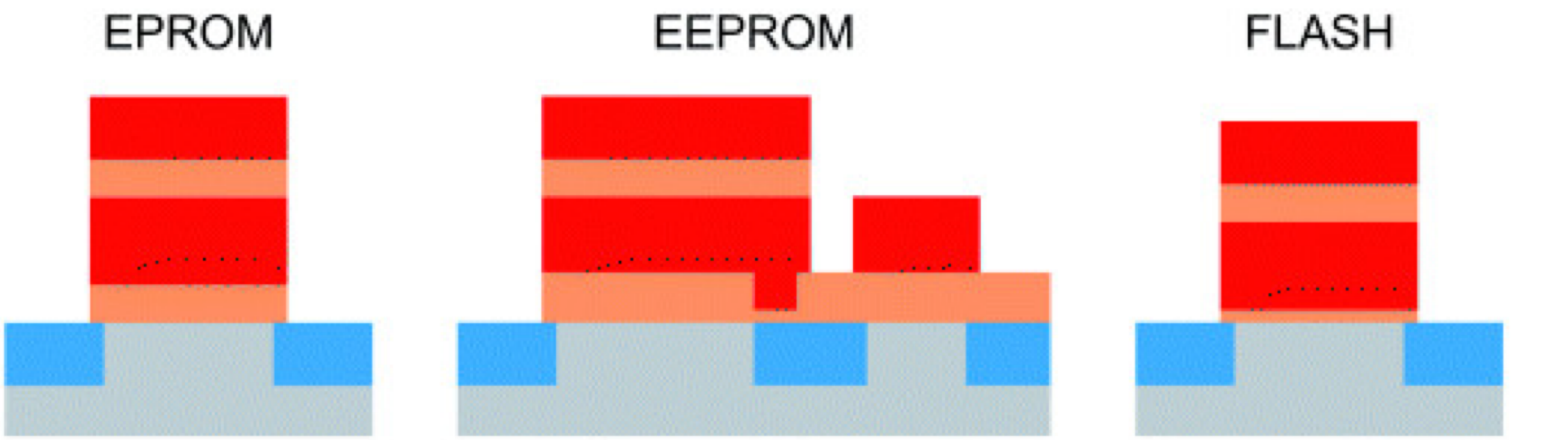


Figure source: Jim Handy, Objective Analysis

Comparison


EPROM
EEPROM
FLASH
Byte programmable
UV erasable
1 Transistor
Thick gateoxide
Byte programmable
(electrical) Byte erasable
2 Transistors
Tunneling window
Byte programmable
(electrical) Block erasable
1 Transistor
Thin gateoxide

Other NV Memories

- Other types emerging:
  - FRAM – Ferroelectric RAM
  - PCM – Phase change memory

- More NV Memories
  - Battery-Backed RAM
  - Micron/Intel "Matrix"

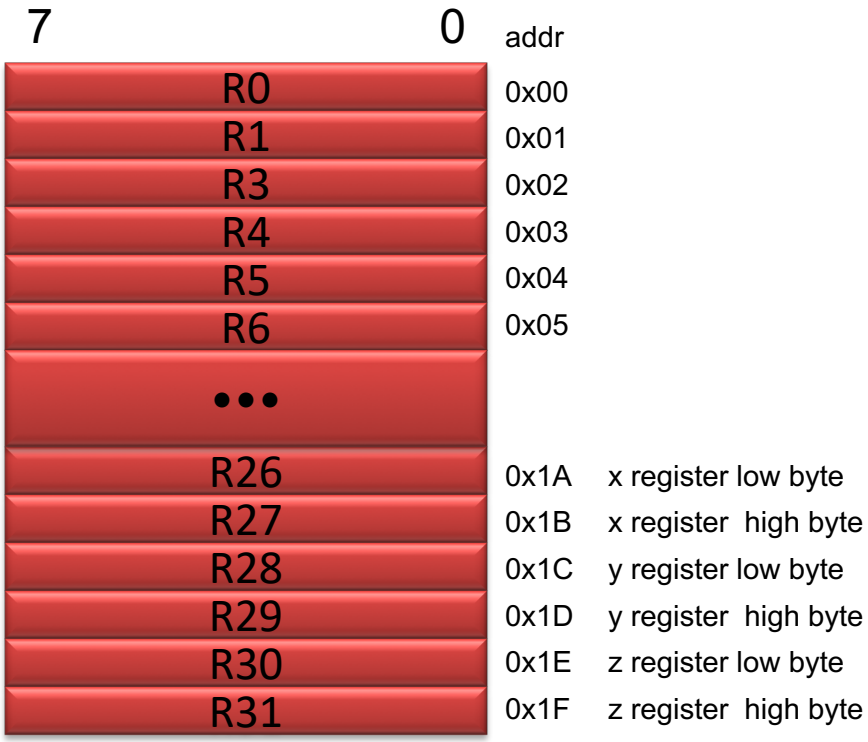
Characteristics of Different Memory Types

Type	*Volatile?*	*Writeable?*	*Erase Size*	*Max Erase Cycles*	*Cost (per Byte)*	*Speed*
SRAM	Yes	Yes	Byte	Unlimited	Expensive	Fast
DRAM	Yes	Yes	Byte	Unlimited	Moderate	Moderate
Masked ROM	No	No	n/a	n/a	Inexpensive	Fast
PROM	No	Once, with a device programmer	n/a	n/a	Moderate	Fast
EPROM	No	Yes, with a device programmer	Entire Chip	Limited (consult datasheet)	Moderate	Fast
EEPROM	No	Yes	Byte	Limited (consult datasheet)	Expensive	Fast to read, slow to erase/write
Flash	No	Yes	Sector	Limited (consult datasheet)	Moderate	Fast to read, slow to erase/write
NVRAM	No	Yes	Byte	Unlimited	Expensive (SRAM + battery)	Fast

AVR Register File

- 32 8-bit registers
- Mapped to address 0-31 in data space
- Most instructions can access any register and complete in one cycle
- Last 3 register pairs can be used as 3 16-bit index registers
- 32 bit stack pointer

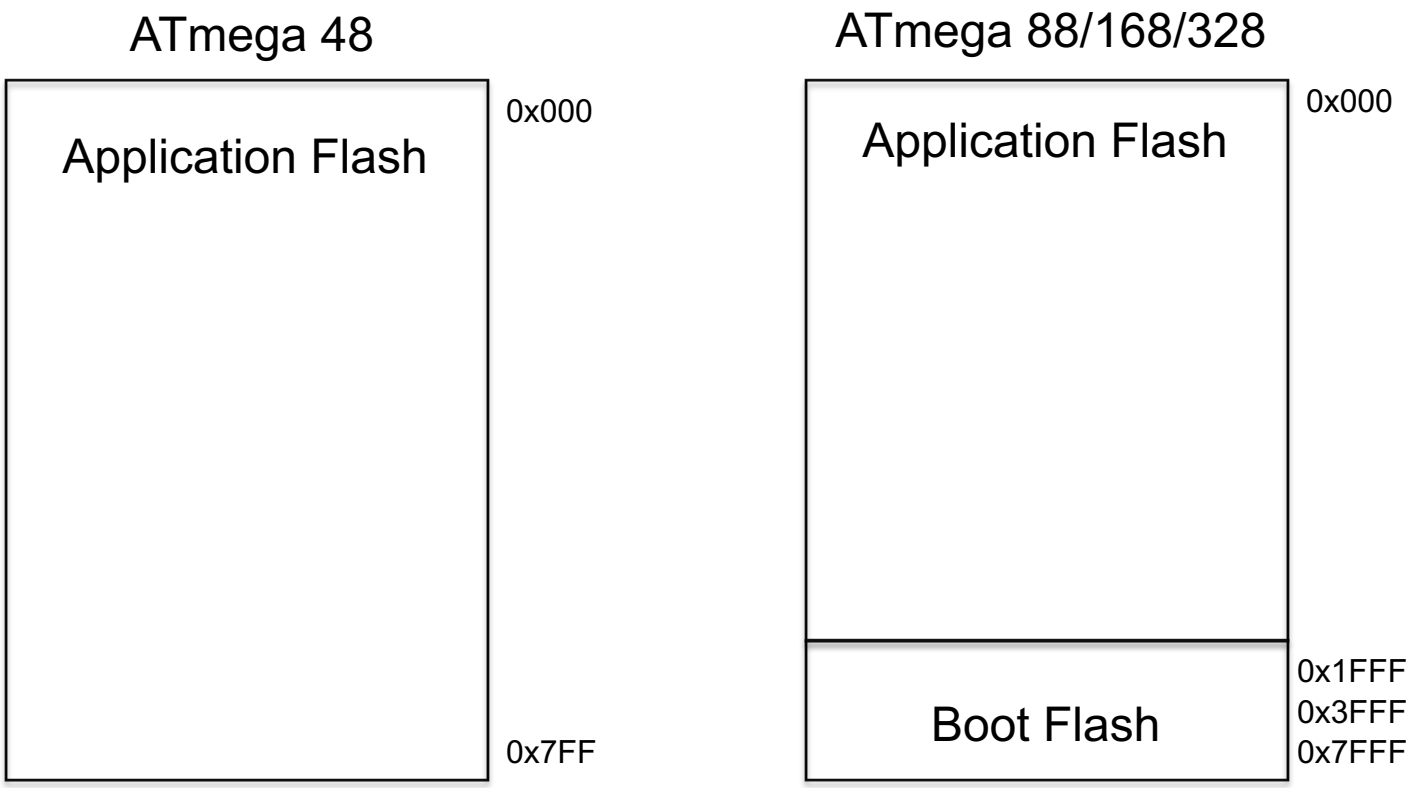
Register File



AVR Memory - FLASH

- Non-volatile program space storage
- 16-bit width
- Some devices have separate lockable boot section
- At least 10,000 write/erase cycles

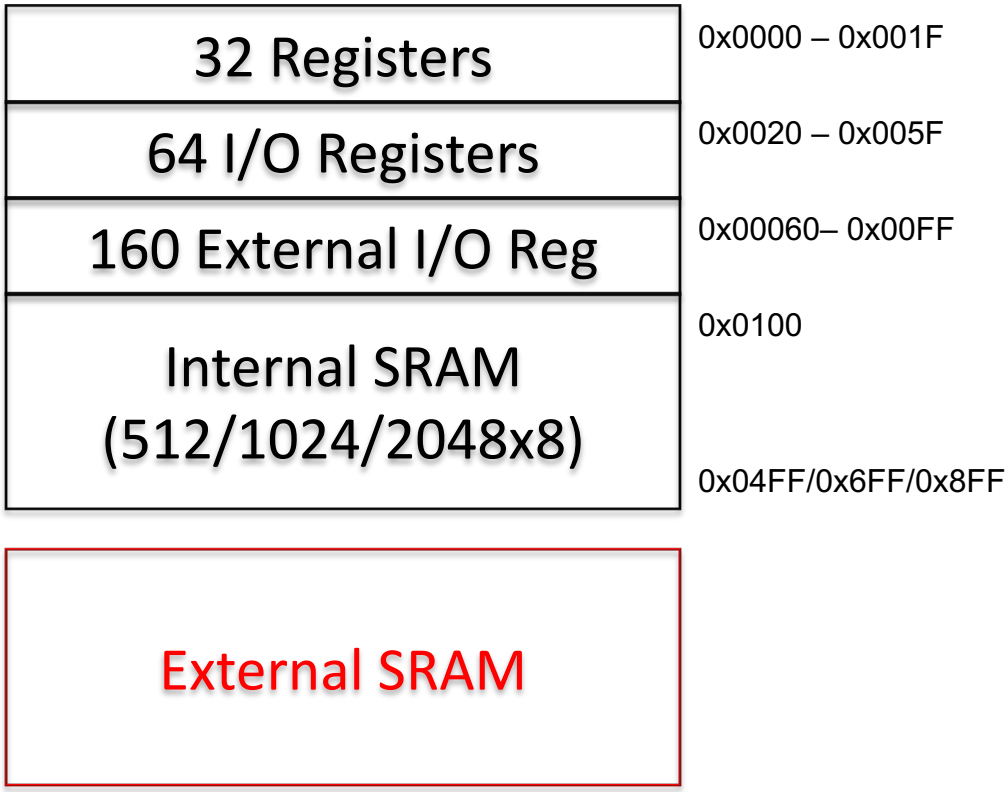
AVR Memories- FLASH – Memory Map



AVR Memories - SRAM

- Data space storage
- 8-bit width

SRAM – Memory Map



AVR Memories- EEPROM

- Electrically Erasable Programmable Read Only Memory
- Byte (8-bit) or word (16-bit) width
- Requires special write sequence
- Non-volatile storage for program specific data, constants, etc.
- At least 100,000 write/erase cycles

AVR Memory Programming

- Flash
  - Program storage, where your code is stored
  - It is programmed during code upload
  - It is possible to dynamically program, but not preferred
  - Look into "Boot Loader Support" in the datasheet
- EEPROM
  - What you will use in Lab 10
  - You can write/edit/read individual byte locations during runtime
  - Program-related variables can be stored
  - The data will not be lost after power is lost
- SRAM
  - Everything else you do dynamically during runtime is stored in SRAM

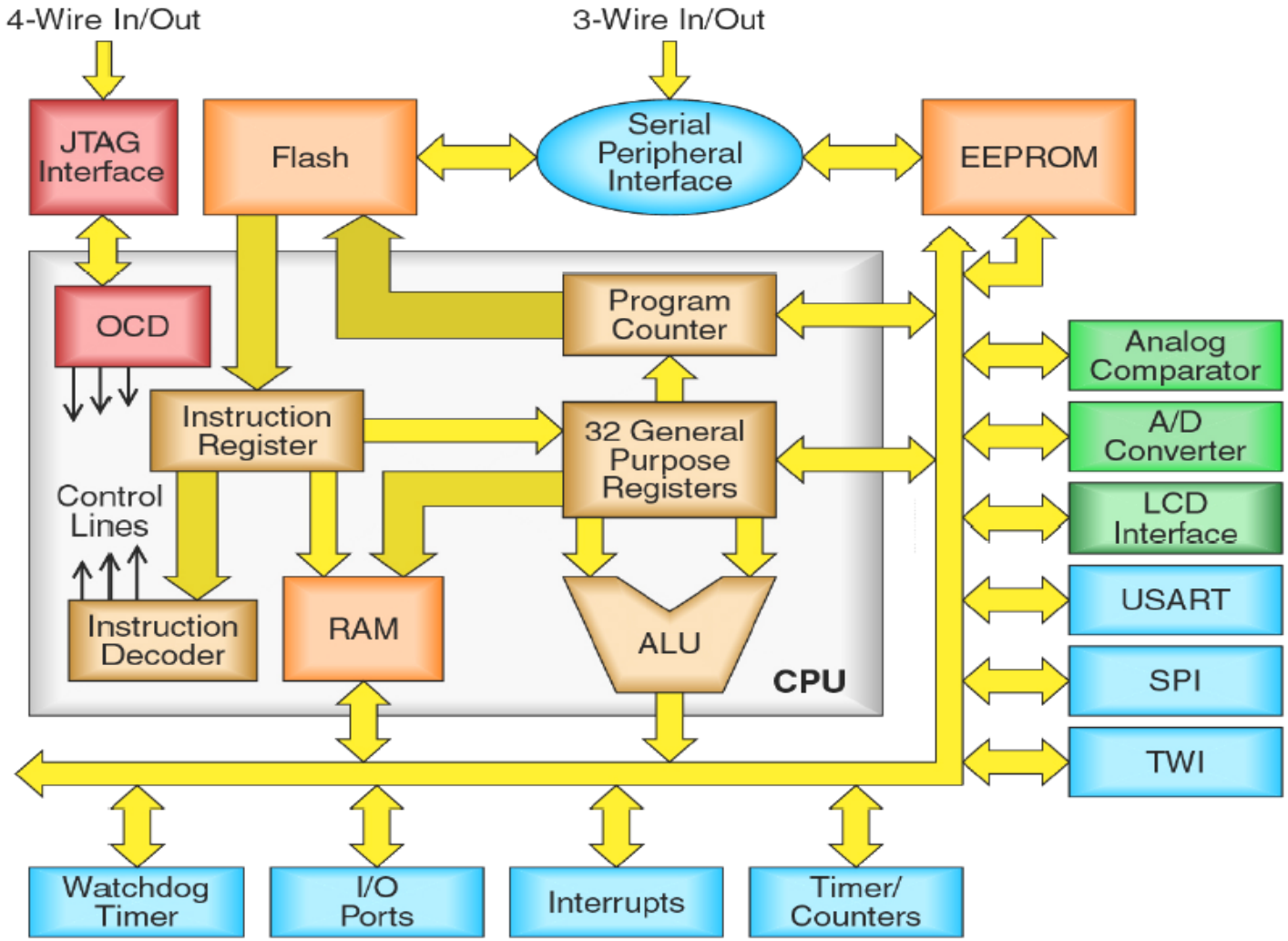
EEPROM Programming

- Two ways
  - Using EEPROM registers manually (Section 8.4 in datasheet)
    - EEAR – eeprom address regisger
    - EEDR – eeprom data register
    - EECR – eeprom control register
    - See page 25 for coding examples
  - Using "eeprom.h" library
    - Recommended way
    - Provides already defined libraries to read/write data in bytes and words
    - Example functions:
      - eeprom_read_byte(…)
      - eeprom_write_byte(…)
      - eeprom_update_byte(…)

AVR Memories

DEVICE	FLASH	EEPROM	SRAM
ATmega48A	4K Bytes	256 Bytes	512 Bytes
ATmega48PA	4K Bytes	256 Bytes	512 Bytes
ATmega88A	8K Bytes	512 Bytes	1K Bytes
ATmega88PA	8K Bytes	512 Bytes	1K Bytes
ATmega168A	16K Bytes	512 Bytes	1K Bytes
ATmega168PA	16K Bytes	512 Bytes	1K Bytes
ATmega328	32K Bytes	1K Bytes	2K Bytes
ATmega328P	32K Bytes	1K Bytes	2K Bytes

AVR Block Diagram



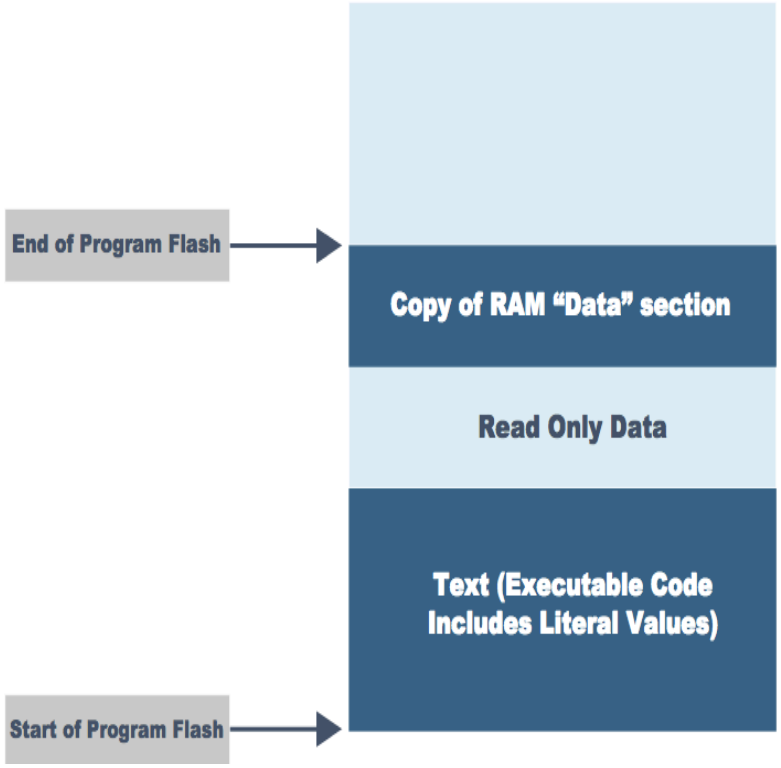
Memory Management (where in memory your code/data is stored)

Memory in an Embedded C Program

- Memory in a C program includes code (executable instructions) and data
  - Code is typically read-only and executable
  - Data memory is non-executable, can be either read-only or read-write, and is either statically or dynamically allocated
- In RAM-constrained embedded systems, the memory map is divided in to a section for flash memory (code and read-only data) and a section for RAM (read-write data)

Flash: Code and Read-Only Memory

- Code and read-only data are stored in flash memory
- The beginning of the program (the lowest memory location at the bottom of the diagram) is the text section which includes executable code
- This section also includes numerical values that are not assigned to any specific C variable called "literal values"



Flash: Code and Read-Only Memory

- The "data" section which contains the initial values of global and static variables
  - This section is copied to RAM when the program starts up
- Example:
  - **read_only_variable** is stored in the read-only data section because it is preceded by the **const** keyword
  - The compiler assigns **read_only_variable** a specific address location (in flash) and writes the value of 2000 to that memory location
  - When the variable x within **my_function()** is assigned the literal value 200, it references the value stored in a "literal pool" within the text section
  - A copy of the initial value, 500, assigned to **data_variable** is stored in flash memory and copied to RAM when the program starts

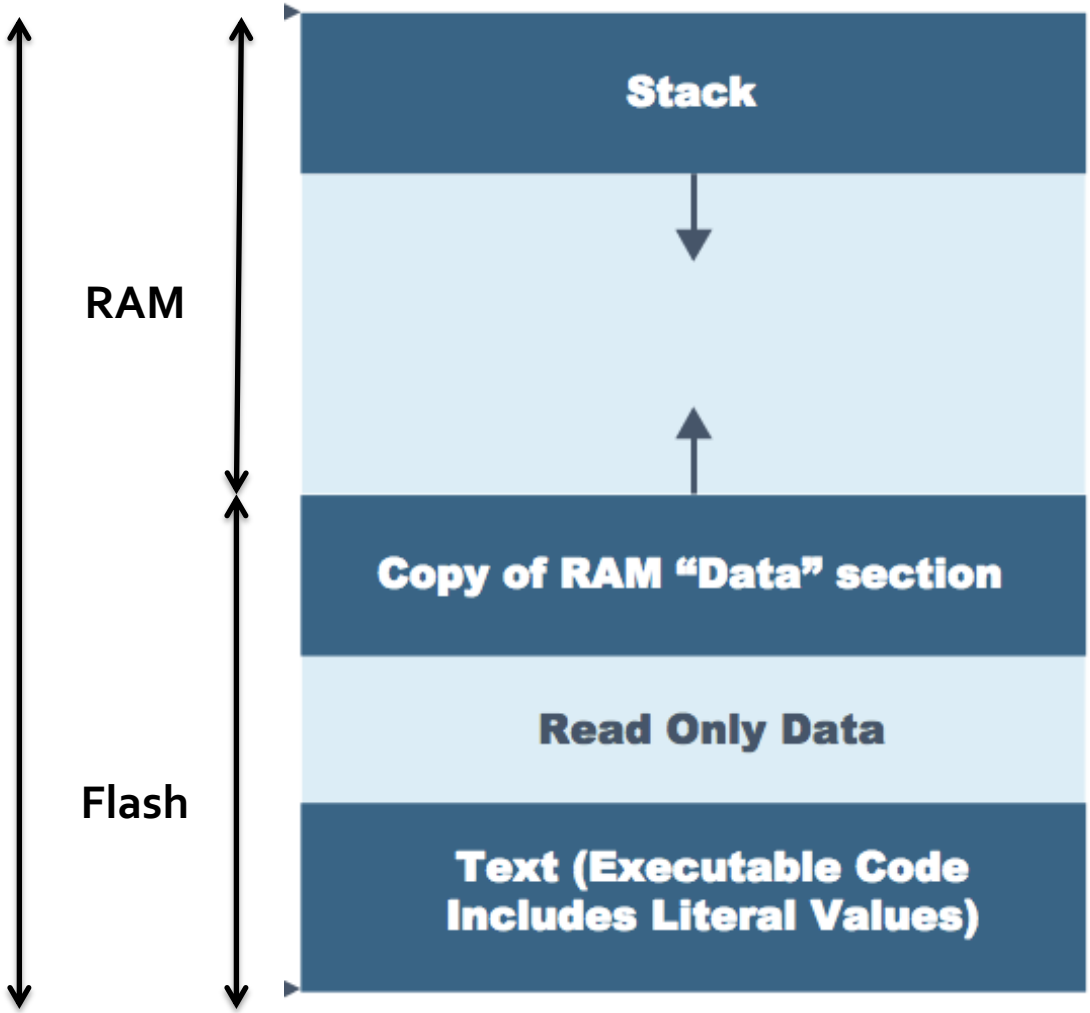
```
#include <stdio.h>
const int read_only_variable = 2000;
int data_variable = 500;

void my_function(void){
     int x;
     x = 200;
     printf("X is %d\n", x);
}
```

RAM: Read-Write Data

- After the program starts running

Memory requirement of the entire C program

The read-write data that is stored in RAM is further categorized as statically or dynamically allocated



Statically Allocated Data

- Statically allocated memory means that the compiler determines the memory address of the variable at compile time
- Static data is divided in two sections: data and bss
  - Data is assigned an intial, non-zero value when the program starts
  - All variables in the bss section are initialized to zero
- When the program starts:
- C runtime (CRT) start function loads the memory location assigned to **data_var** with 500
  - Copy the value from Flash to RAM
- The CRT start function then sets the memory locations for bss_varo and bss_var1 to zero which does not require any space in flash memory

```c
#include <stdio.h>

//these variables are globally allocated
int data_var = 500;
int bss_var0;
int bss_var1 = 0;

void my_function(void){
     int uninitialized_var;
     printf("data_var:%d, bss_var0:%d\n", data_var, bss_var0);
}
```

Dynamically Allocated Data

- The locations of dynamically allocated variables are determined while the program is running
- Heap vs. Stack
- The stack grows down (from higher memory address to lower ones) and the heap grows up
- If memory usage is ignored in the design, the stack and heap can collide causing one or both to become corrupted
- The heap is managed by the programmer while the compiler takes care of the stack

Stack vs. Heap

- The beginning of the heap is just above the last bss variable
- To manage the heap: malloc() and free()
- Variables that are declared within a function, known as local variables, are either allocated on the stack or simply assigned a register value
- Whether a variable is allocated on the stack or assigned to a register depends on many factors
  - Compiler, the microcontroller architecture, number of variables already assigned to registers, etc.