

HIGH PERFORMANCE PROCESSORS INTRODUCTION

Justas Dilys

Grading Scheme:

- I. **Laboratory work** – 4 points
- II. **Presentation (PowerPoint)** – 3 points
- III. **Final exam** (last lecture before Christmas) – 3 points

Time schedule

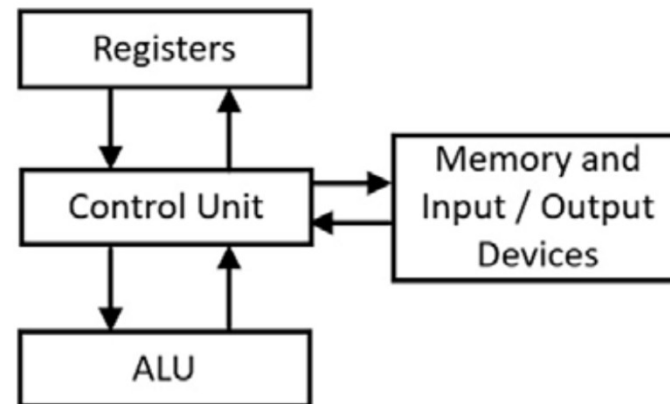
Day	Time	Activity	Location / Room	Notes
Monday	12:10-13:45	Lecture	P2 334	Main theory topic
Monday	14:30–16:05	Laboratory	P2 334	Practical exercises

- ❖ **Note:** There will be no laboratory sessions on Thursdays !!!
- ❖ **P.S.** There will be no lecture or laboratory on 2025-11-17 as well.

Processor Elements

A processor contains three logically distinct functional units:

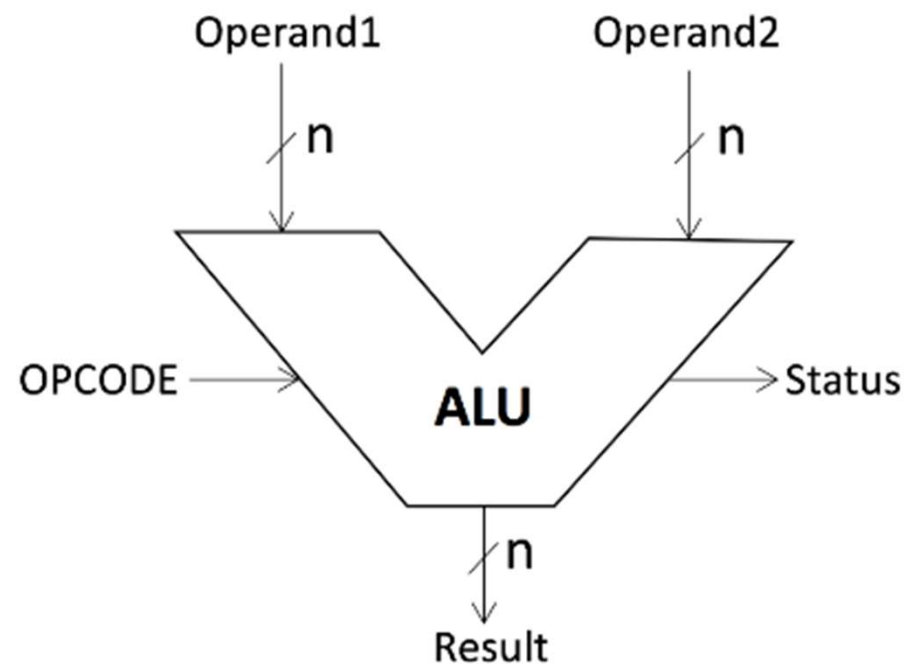
1. The **Control Unit (CU)**
2. The **Arithmetic Logic Unit (ALU)**
3. The **Register set**



The central processing unit (CPU) is considered the brains of the computer because it facilitates all **steps involved in executing instructions**. The CPU **handles reading instructions from memory, decoding the op-code to determine which instruction is being performed, and executing** the necessary steps to complete the instruction. The CPU also contains a set of registers that are used for general-purpose data storage, operational information, and system status. Finally, the CPU contains circuitry to perform arithmetic and logic operations on data.

Arithmetic Logic Unit (ALU)

The **ALU performs arithmetic and bit-oriented operations** under the direction of the control unit. To perform an operation, the ALU requires data input values, called **operands**, along with a code indicating the operation to be performed. The ALU output is the result of the operation. ALU operations may use one or more processor flags, such as the carry flag, as input, and set the states of processor flags as outputs. An ALU is a combinational circuit, which implies its outputs update asynchronously in response to changes at the inputs and it retains no memory of previous operations. To execute an instruction involving the ALU, the control unit applies inputs to the ALU, pauses to allow for the propagation delay across the ALU, and then transfers the ALU output to the destination specified by the instruction.

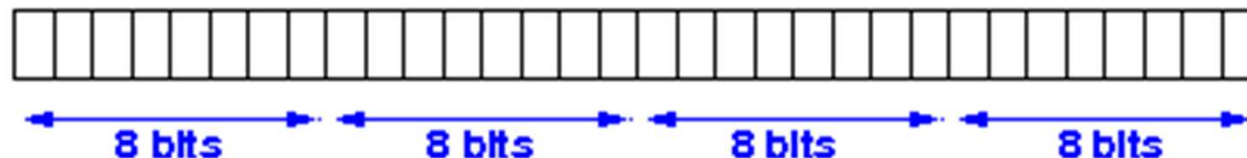


Registers

A *register* is a small, fast storage location inside the CPU used to hold data, addresses, or control information during program execution.

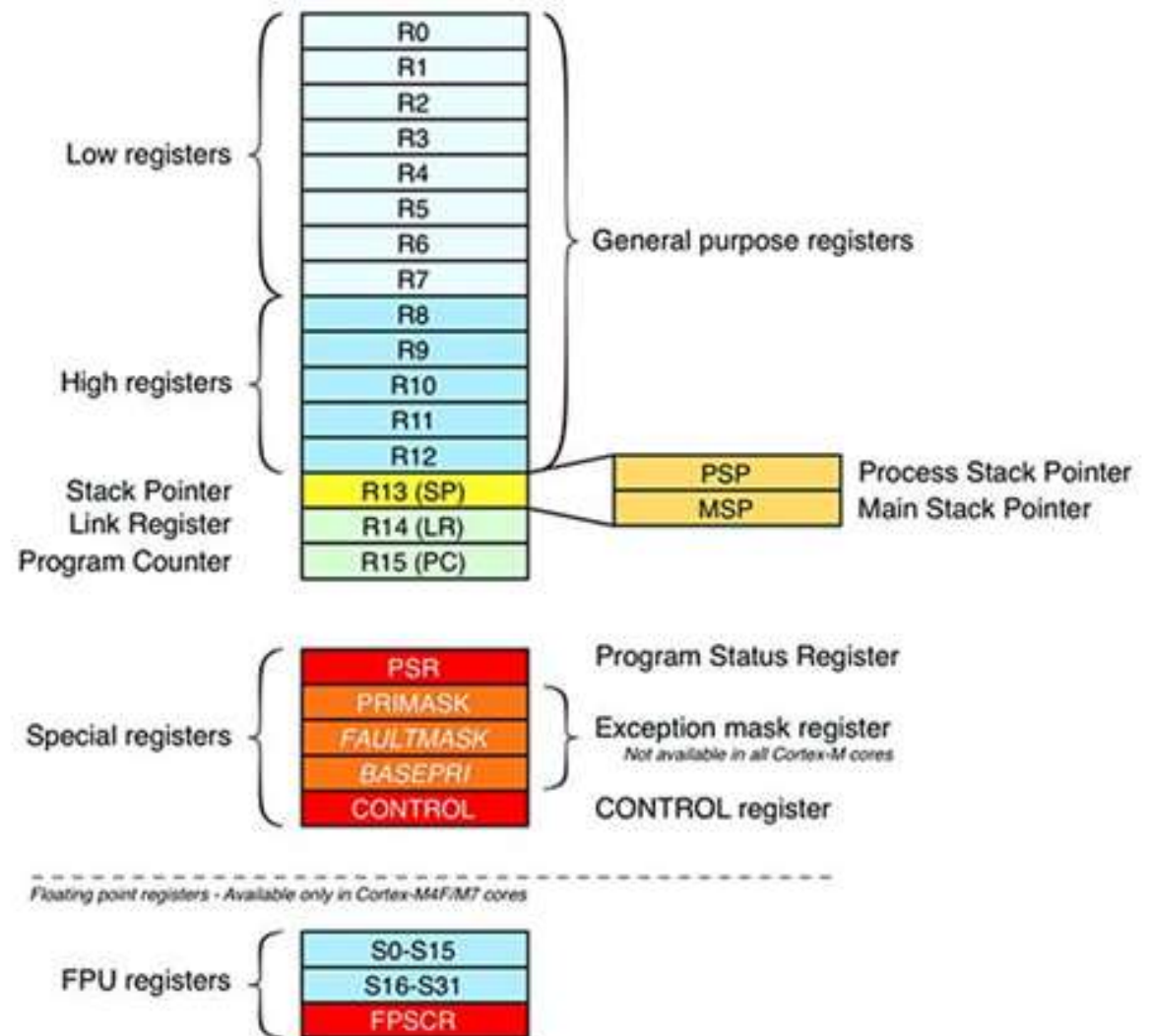
In the CPU, registers are used to store information temporarily. That information could be a piece of data to be processed, or an address pointing to the data to be fetched. All of ARM registers are 32-bit wide. These range from the MSB (most-significant bit) to the LSB (least-significant bit). **Although the ARM default data size is 32-bit many assemblers also support the single bit, 8-bit, and 16-bit data types.** The 32-bit data size of the ARM is often referred as “**word**”. In ARM the 16-bit data is referred to as **half-word**. Therefore, ARM supports byte, half-word (two bytes), and **word (four bytes)** data types.

ARM's register:



Example: ARM Registers

- ❑ On **ARM Cortex-M** cores, the **general-purpose registers** are the integer registers **R0-R12**.
- ❑ Some 16-bit Thumb instructions can only access a subset of these registers (low registers, R0-R7).



Example: The Program Counter

Register r15 is the program counter (pc) and contains the address of the next instruction to be fetched by the processor.

```
; Jump forward by 8 bytes (2 instructions)
```

```
    ADD     R0, R0, #1
```

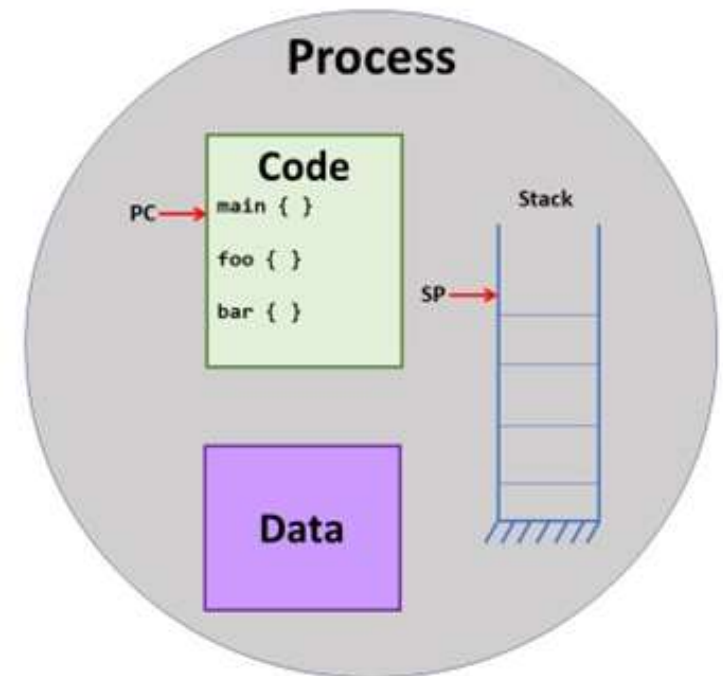
```
    B       skip           ; branch sets PC
```

```
    ADD     R1, R1, #1     ; skipped
```

```
skip:
```

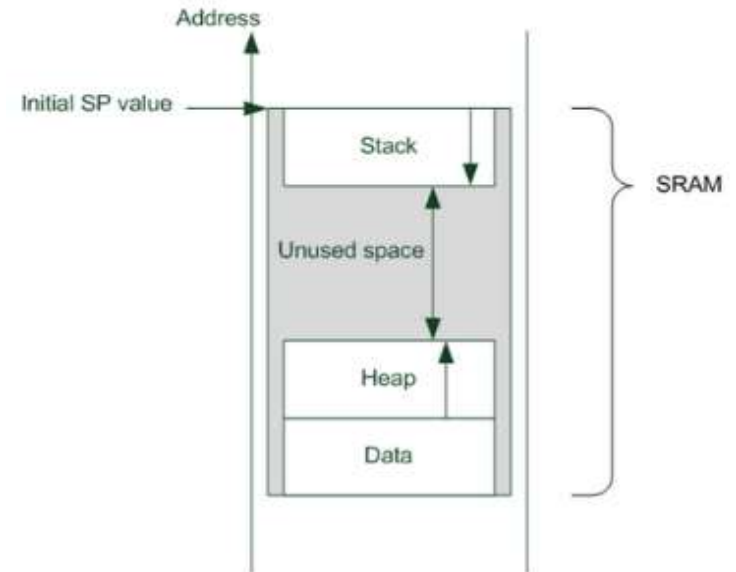
```
    SUB     R0, R0, #2
```

```
0x08000000:  ADD  R0, R0, #1
0x08000004:  B    skip
0x08000008:  ADD  R1, R1, #1  ; will be skipped
0x0800000C: skip:
0x0800000C:  SUB  R0, R0, #2
```



Example: Stack Pointer

The stack pointer register r13 (SP), **keeps track of the stack location** for the current thread and usually points to the logical “top” of the current thread stack. The stack region is used by programs to efficiently store and access local variable data for a given function and as general-purpose “scratch” memory for storing data such as function return addresses.



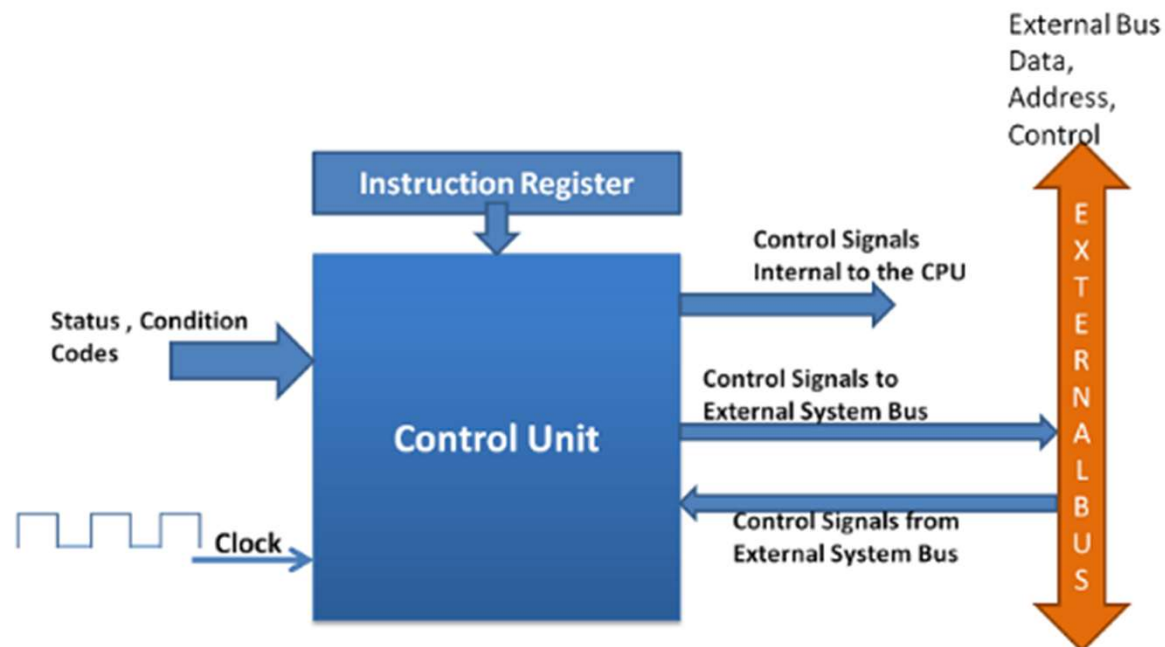
Typical arrangement 2: Stack region placed at the end of SRAM space

```
my_func:
    PUSH    {R4, LR}        ; save registers to stack, SP decremented by 8
    ADD     R4, R0, R1      ; use R4 as local variable
    MOV     R0, R4          ; return result in R0
    POP     {R4, PC}        ; restore registers from stack
```

Control Unit (CU) 1/2

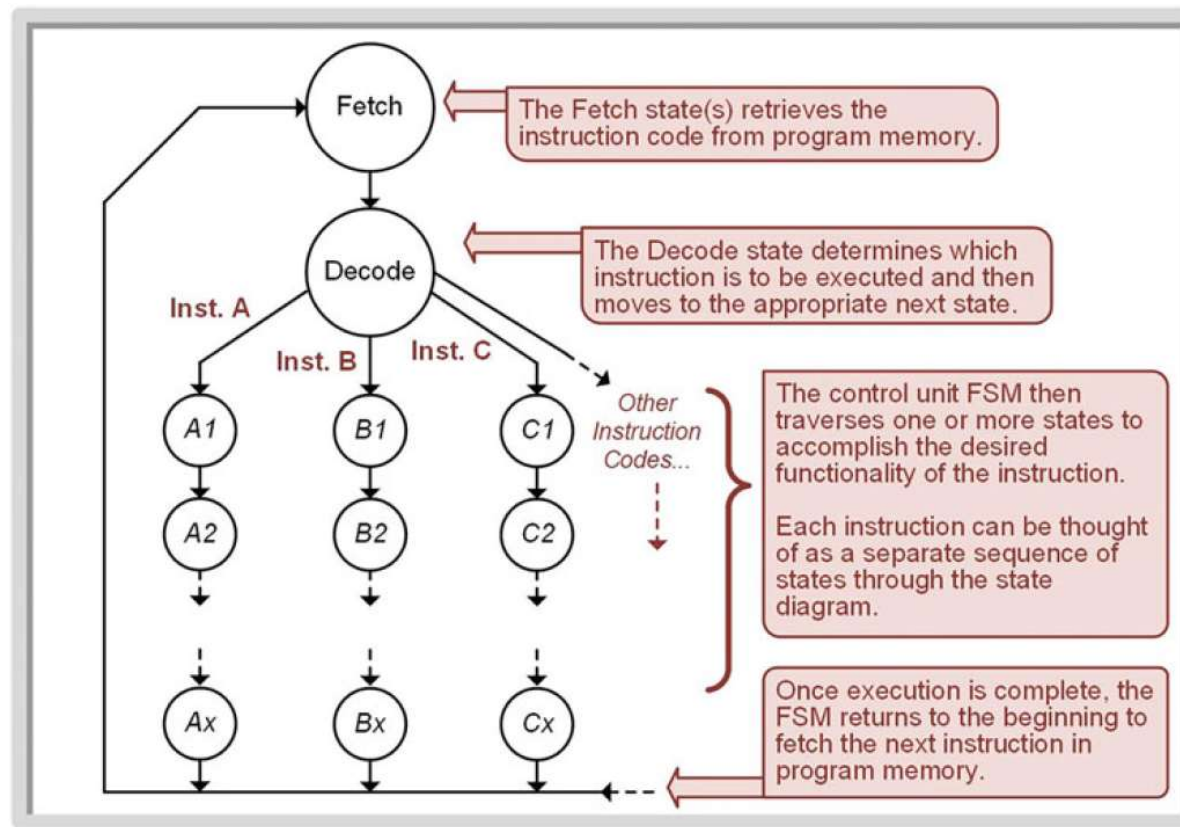
The purpose of the **CU** is to generate control signals. These control signals facilitate flawless execution of instructions in CPU, handling of **Interrupts** and **internal errors** by CPU, communication over the internal bus(es) in CPU.

When a computer system is powered on, the processor undergoes a reset process to initialize its internal components. Following a reset, the processor loads the **Program Counter (PC)** with the memory location of the first instruction to be executed. **The PC is a central component of the control unit.**



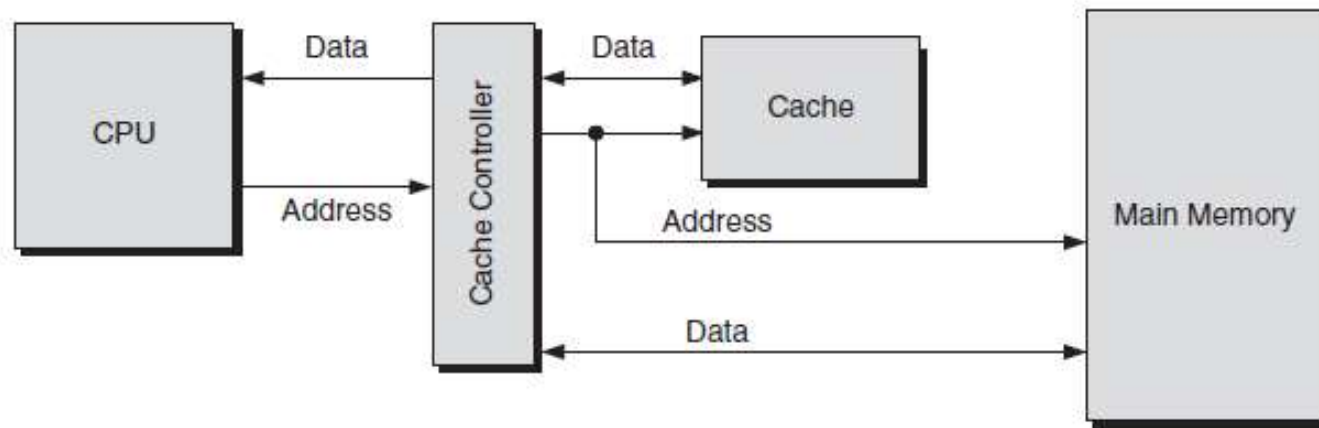
Control Unit (CU) 2/2

The **control unit** is a **finite state machine (FSM)** that **controls the operation of the computer**. This FSM has states that perform **fetching** the instruction (i.e., reading it from program memory), **decoding** the instruction op-code, and **executing** the appropriate steps to accomplish the functionality of the instruction. This process is known as the **fetch ! decode ! execute cycle** and is repeated each time an instruction is executed.

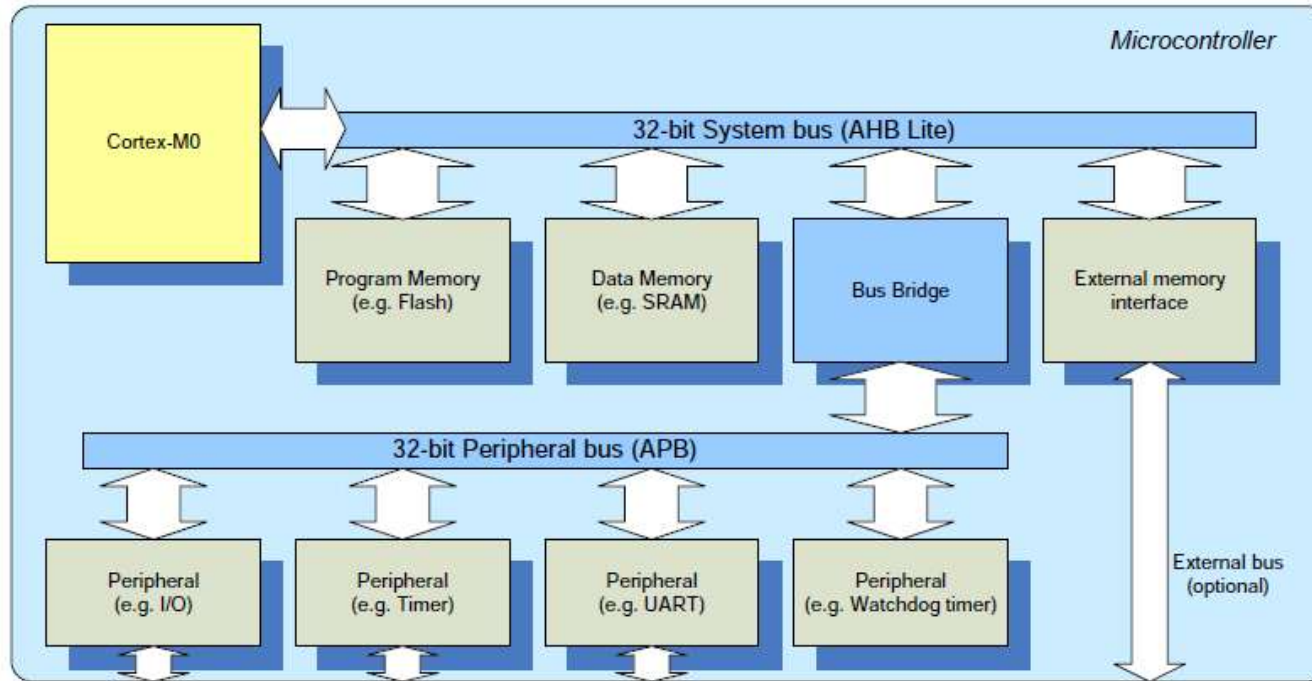


Bus System

The **bus system of a computer handles routing all signals between the CPU and memory**. Various control signals are also included in the bus system to facilitate reading and writing. One key concept of the bus system is that the I/O ports, data memory, and program memory share the address and data buses. Every specific location, regardless of being an I/O port, a location in data memory, or a location in program memory, is assigned a unique address. This is called a memory mapped system. Each microcontroller has a specific memory map, which gives the addresses for all locations in memory.



Example: Cortex-M0



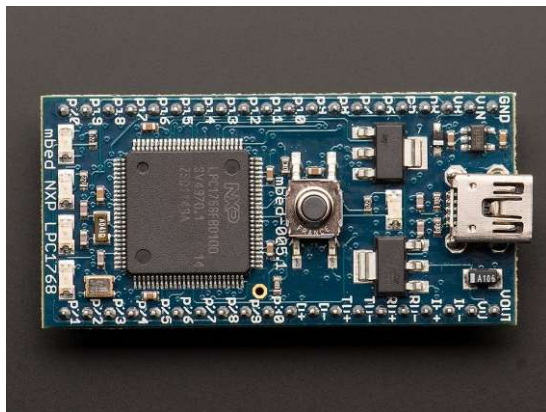
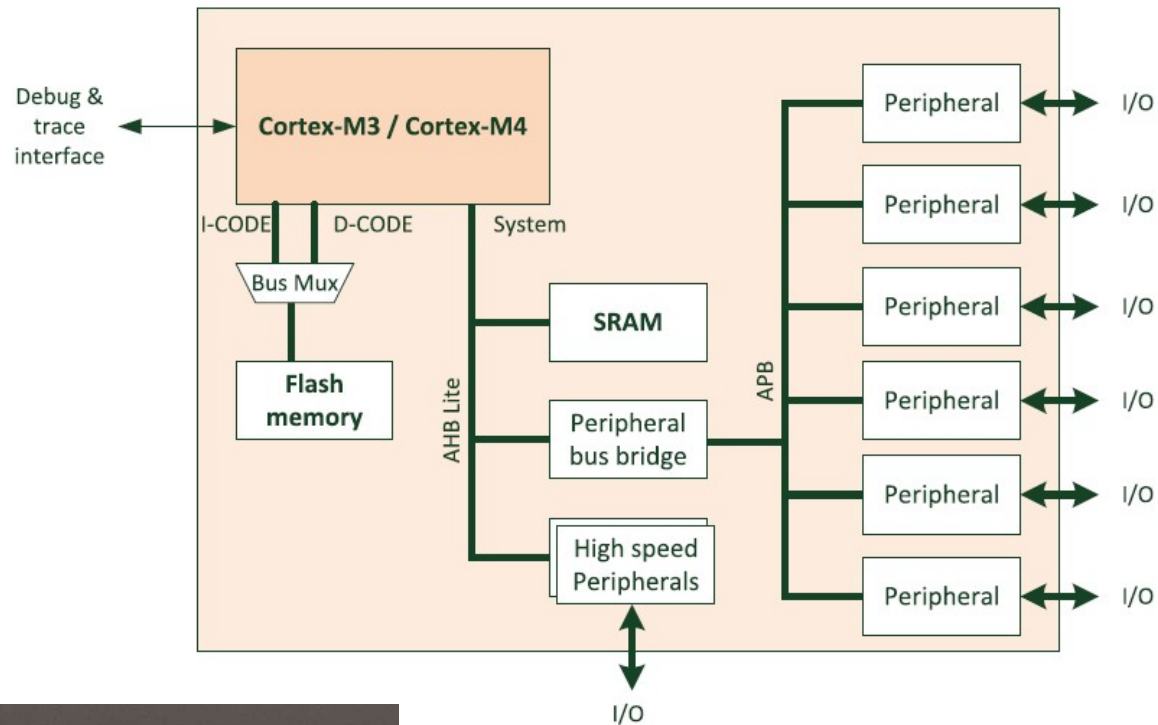
PIO0_8/MISO0/CT16B0_MAT0	1	PIO0_4/SCL	20
PIO0_9/MOSI0/CT16B0_MAT1	2	PIO0_2/SSEL0/CT16B0_CAP0	19
SWCLK/PIO0_10/SCK0/CT16B0_MAT2	3	PIO0_1/CLKOUT/CT32B0_MAT2	18
R/PIO0_11/AD0/CT32B0_MAT3	4	RESET/PIO0_0	17
PIO0_5/SDA	5	Vss	16
PIO0_6/SCK0	6	VDD	15
R/PIO1_0/AD1/CT32B1_CAP0	7	XTALIN	14
R/PIO1_1/AD2/CT32B1_MAT0	8	XTALOUT	13
R/PIO1_2/AD3/CT32B1_MAT1	9	PIO1_7/TXD/CT32B0_MAT1	12
SWDIO/PIO1_3/AD4/CT32B1_MAT2	10	PIO1_6/RXD/CT32B0_MAT0	11

LPC1110FD20
LPC1112FD20/
102

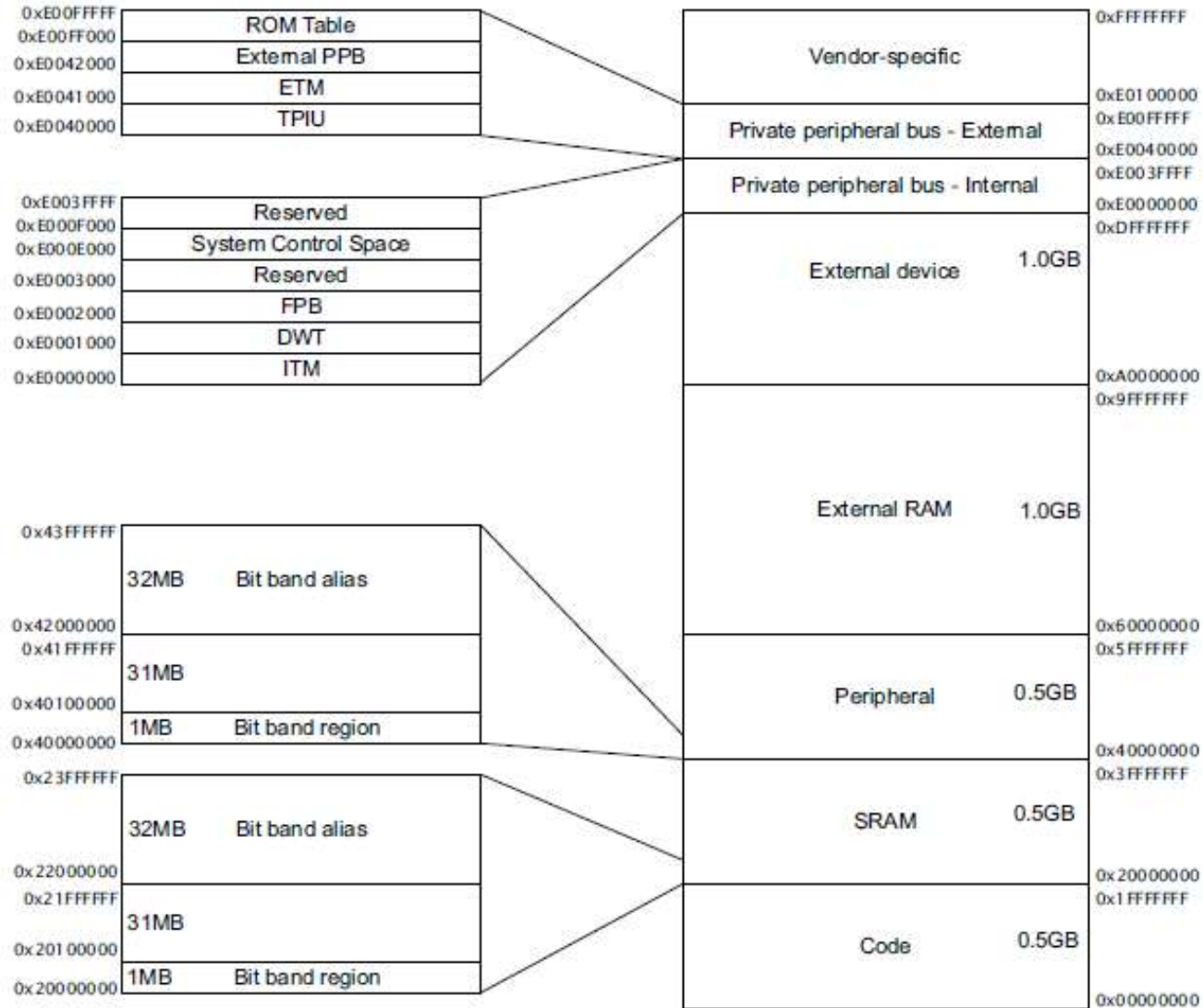
002aag995



Example: Cortex-M3/M4 - Harvard

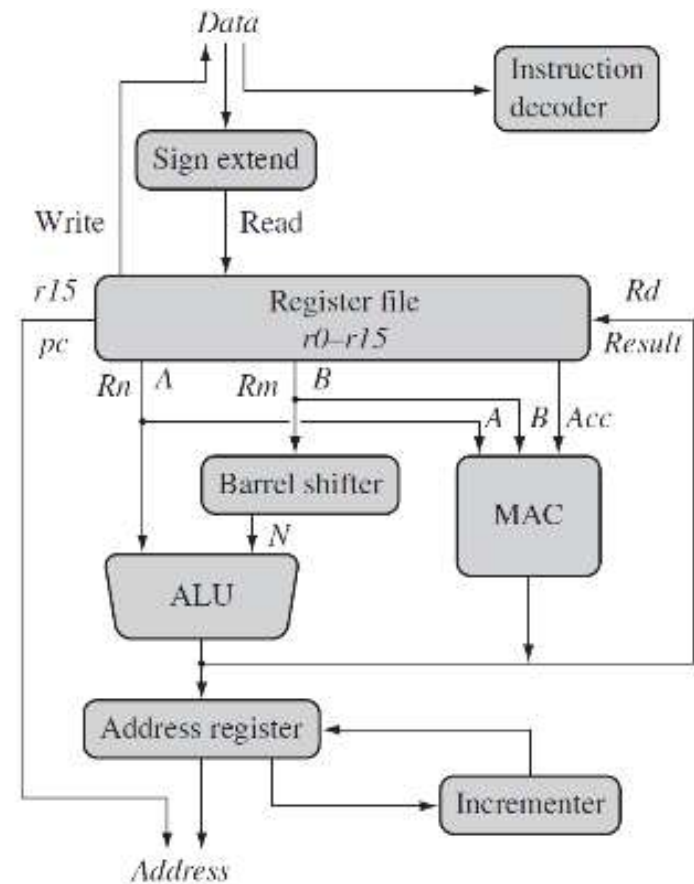


Example: The Memory Map



Example - ARM CPU

- ❑ The ARM processor, uses a **load-store architecture**.
- ❑ Data items are placed in the **register file**—a storage bank made up of 32-bit registers.
- ❑ Since the ARM core is a **32-bit processor**, most instructions treat the registers as holding signed or unsigned 32-bit values.
- ❑ ARM instructions typically have **two source registers, Rn and Rm** , and a single result or **destination register, Rd** .
- ❑ Source operands are read from the register file using the **internal buses A and B** , respectively.



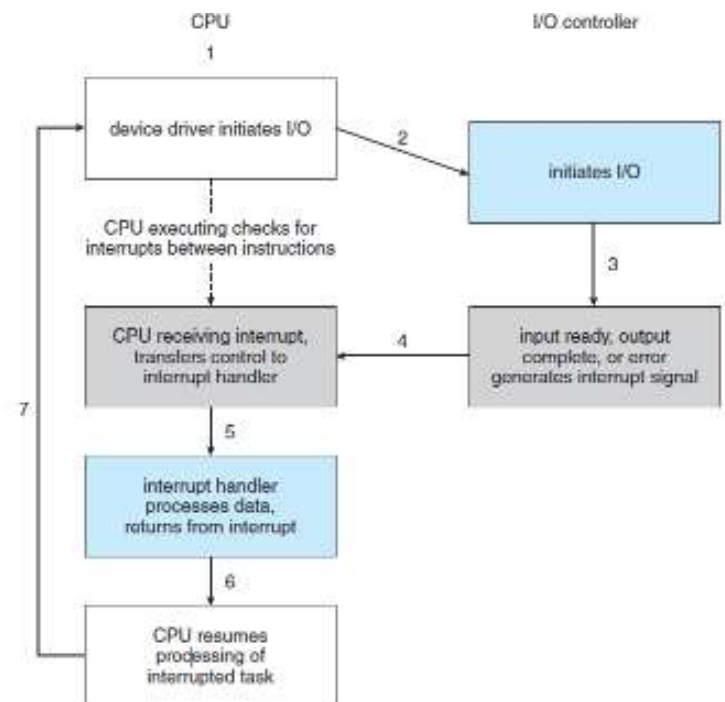
CPU Interrupts

The hardware mechanism that enables a device to notify the CPU is called an **interrupt**.

The CPU hardware has a wire called the **interrupt-request line** that the CPU senses after executing every instruction. When the CPU detects that a controller has asserted a signal on the interrupt-request line, the CPU performs a state save and jumps to the **interrupt-handler routine** at a fixed address in memory.

The interrupt mechanism accepts an **address**—a number that selects a specific interrupt-handling. This address is an offset in a table called the **interrupt vector**.

The interrupt mechanism is also used to handle a wide variety of **exceptions**, such as dividing by zero, accessing a protected or nonexistent memory address, or attempting to execute a privileged instruction from user mode.

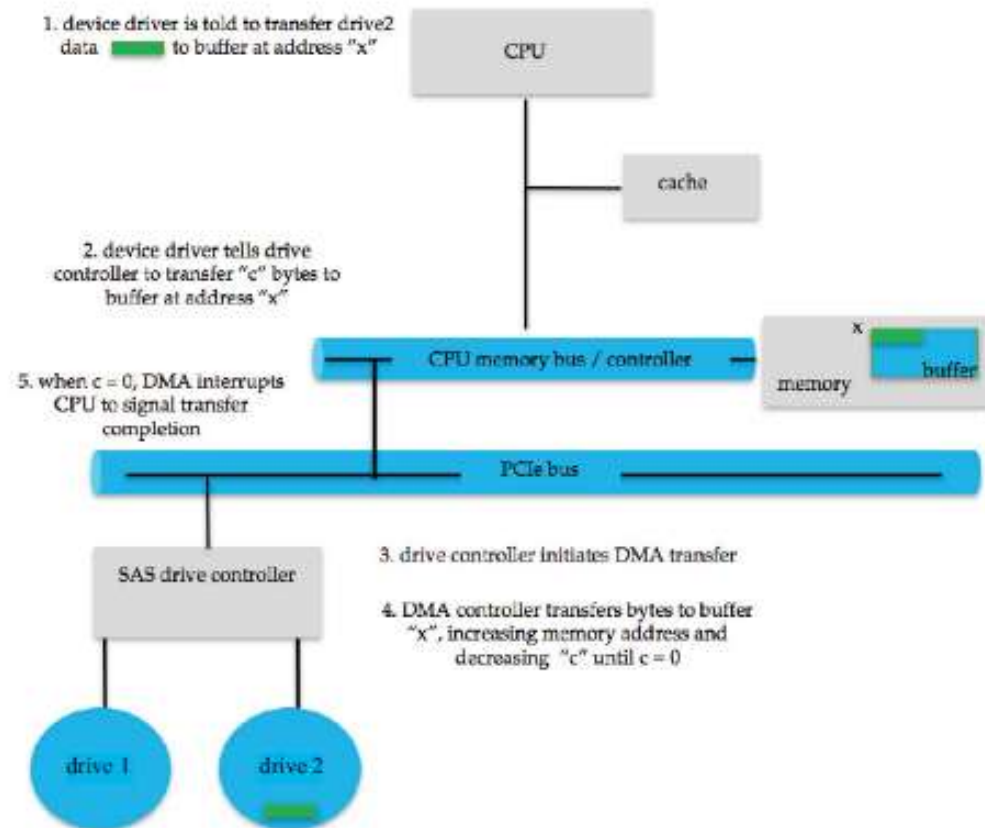


Direct memory access (DMA)

There are situations in which data must be moved very rapidly to or from a device. The classic examples are fast mass-storage devices like magnetic and optical disk drives, and network connections. Interrupt-initiated programmed processing of each data transfer in these examples would be both awkward and, worse, too slow. The solution to these problems is direct memory

access (DMA), a method for direct communication from peripheral to memory.

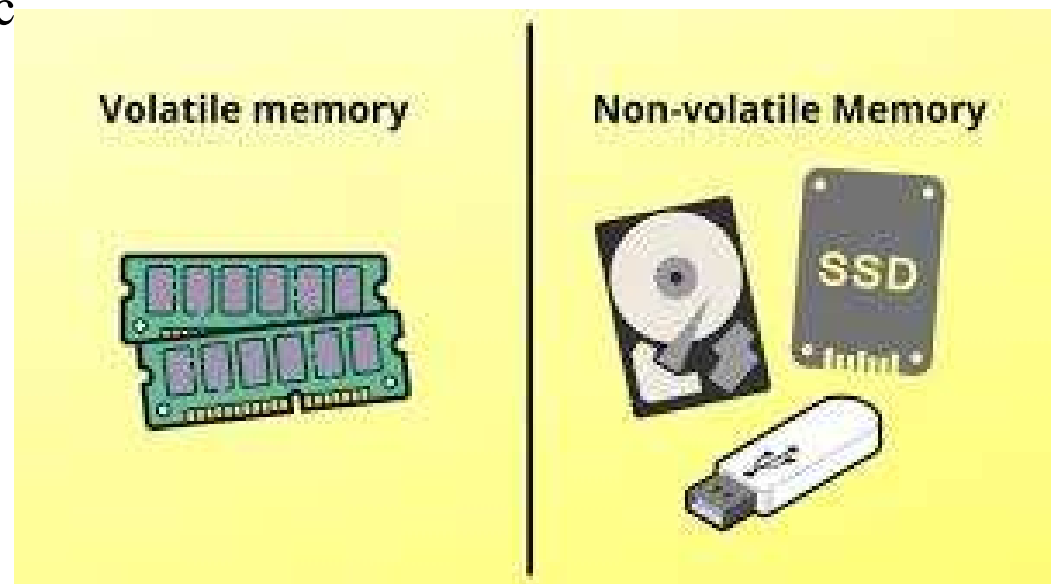
The only effect on the executing program is some slowing down of execution time of CPU.



Memory types

For **many** applications there's **no need to retain memory contents** when power is **turned off**. A computer, for example, freshly **loads its operating system**, application programs, and data into working memory **during the boot process**, so it's OK if that memory is volatile. Those programs and data must of course, be **retained** somewhere **when power is off**, and that is the **function of non-volatile mass memory**, usually in the form of a hard disk. By accepting volatile memory you **gain in speed, density, endurance** (number of erase/write cycles before wear out), and price compared with those of currently available non-volatile technologies.

Both static RAM (SRAM) and dynamic RAM (DRAM) are volatile, whereas flash and EEPROM (along with some interesting new technologies) are nonvolatile.



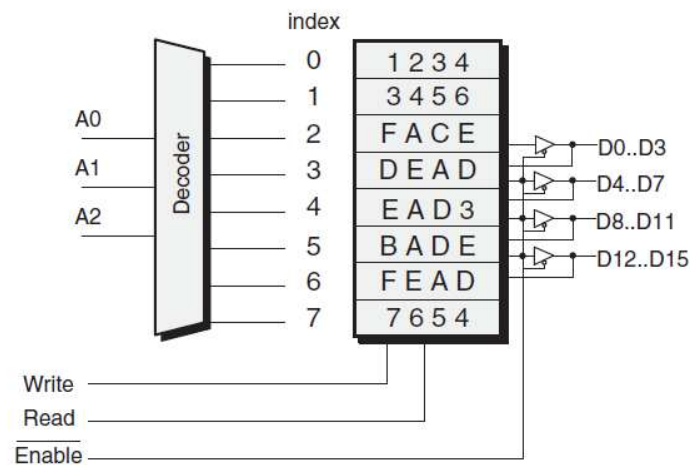
A general memory interface

A value (**Data**) can be assigned to a location in an array and the value of a piece of data that has been stored there can be read. For an array, we identify where the data is stored by an **index** number.

Index	
0	1 2 3 4
1	3 4 5 6
2	F A C E
3	D E A D
4	E A D 3
5	B A D E
6	F E A D
7	7 6 5 4

↕
Data

A general memory interface



Common Memory Control Signals

	Chip Select (CS)	Output Enable (OE)	Read (R)	Write (W)	Column Address Strobe (CAS)	Row Address Strobe (RAS)
ROM	X	X				
SRAM	X	X	X	X		
DRAM	X	X	X	X	X	X

SRAM vs DRAM

Static RAM stores bits in an array of **flip-flops**, whereas dynamic RAM stores bits as **charged capacitors**. A bit once written in a SRAM stays there until rewritten or until the power is turned off. In a DRAM the data will disappear in less than a second, typically, unless "refreshed." In other words, a DRAM is always busy forgetting data, and it is rescued only by periodic clocking through the "rows" of the two-dimensional pattern of bits in the chip.

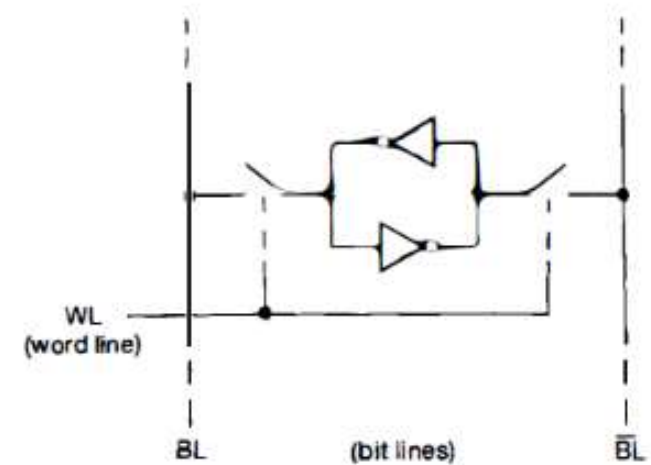


SRAM vs DRAM

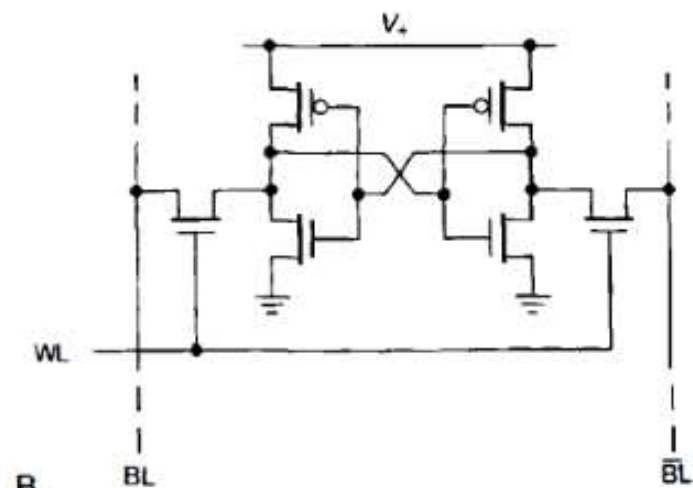
Feature	Static RAM (SRAM)	Dynamic RAM (DRAM)
Storage Method	Flip-flops	Capacitors
Data Retention	Holds data until power is off	Loses data unless refreshed
Refresh Required	No	Yes (periodic refresh)
Speed	Faster	Slower due to refresh cycles
Density	Lower (fewer bits per chip)	Higher (more bits per chip)
Cost	Expensive	Cheaper (~1/10 the cost of SRAM)
Power Consumption	Very low (especially idle)	Higher (due to refresh)
Complexity	Simple timing	Complex timing (needs refresh logic)
Typical Use	Cache memory, small embedded systems	Main memory in PCs, large systems

Static RAM (SRAM)

Static RAM stores each bit in a flip-flop cell the flip-flop itself consists of a pair of cross-connected inverters (each made from a complementary pair of pMOS and nMOS switches), with two additional nMOS transmission gates to couple it to the outside. This is known as a “6T” (six-transistor) SRAM cell.



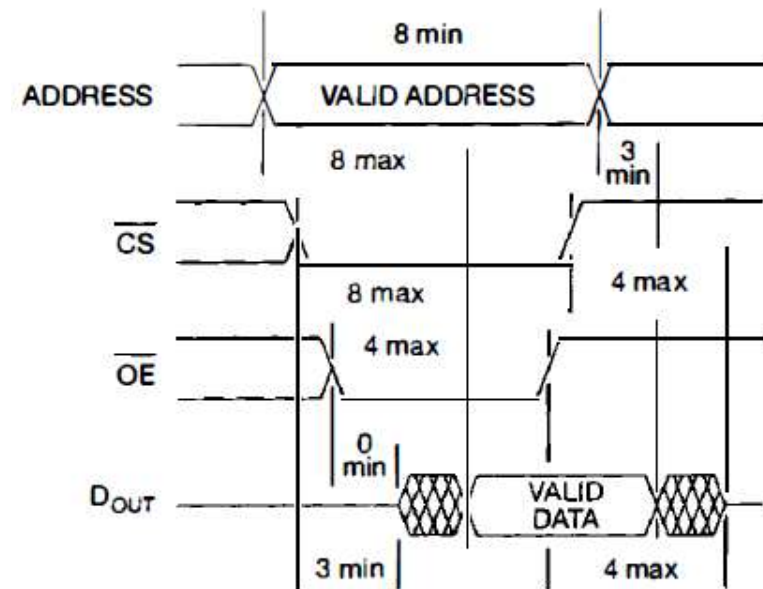
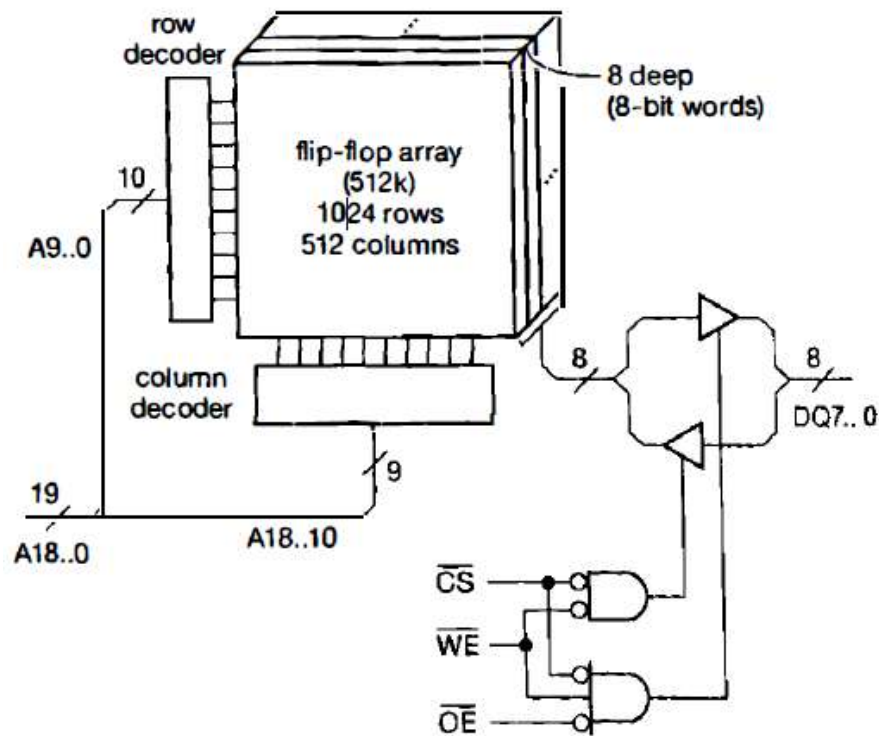
A.



B.

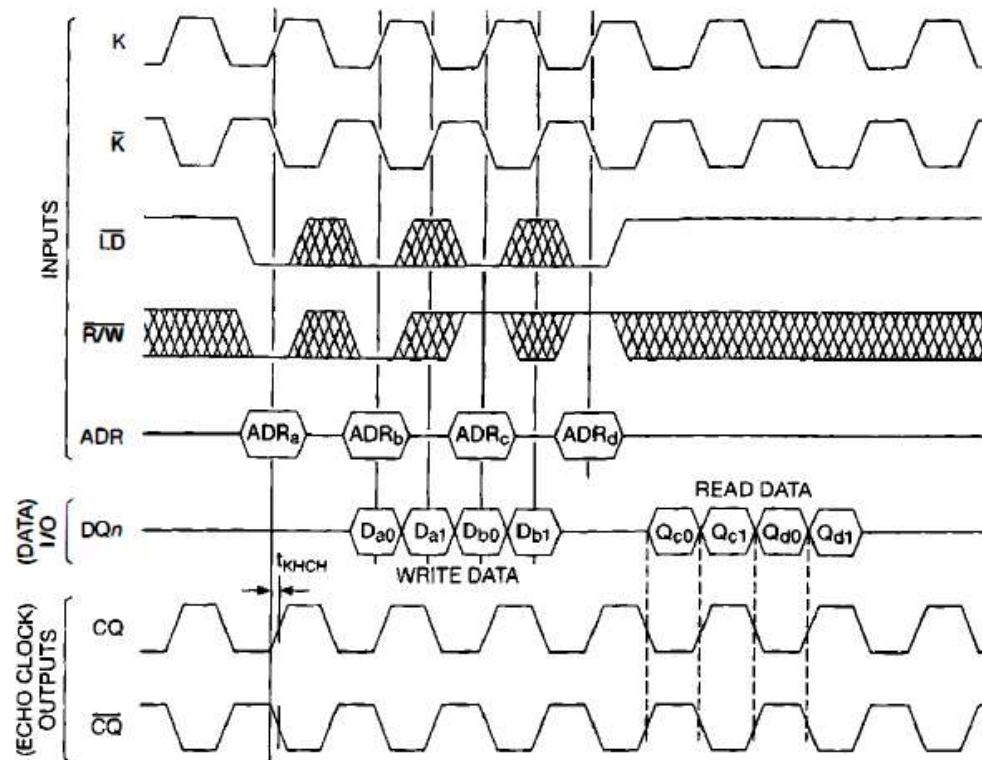
Asynchronous SRAM

Traditional SRAM is asynchronous, meaning that there is **no clocking input**; instead, you apply address, data, and control signals with proper timing, and data comes out (READ cycle) or is written (WRITE cycle) accordingly



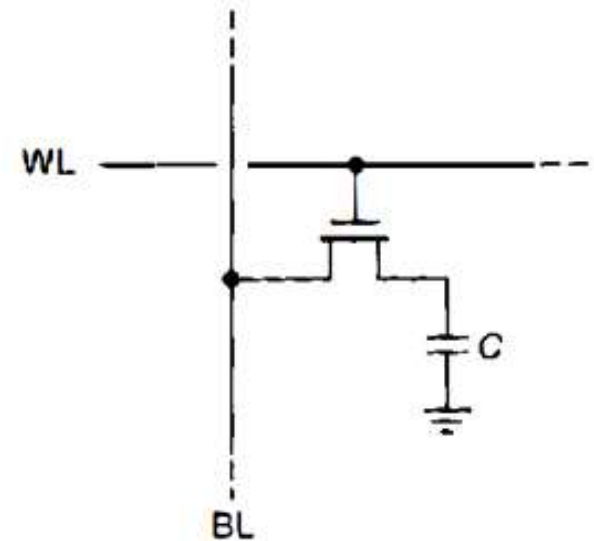
Synchronous SRAM

You can wrap a synchronous clocked state machine, complete with its data registers, around the intrinsically asynchronous memory-array core. Then you've got **synchronous** SRAM. Because it is clocked, the speed of synchronous SRAM is given as a maximum **frequency**



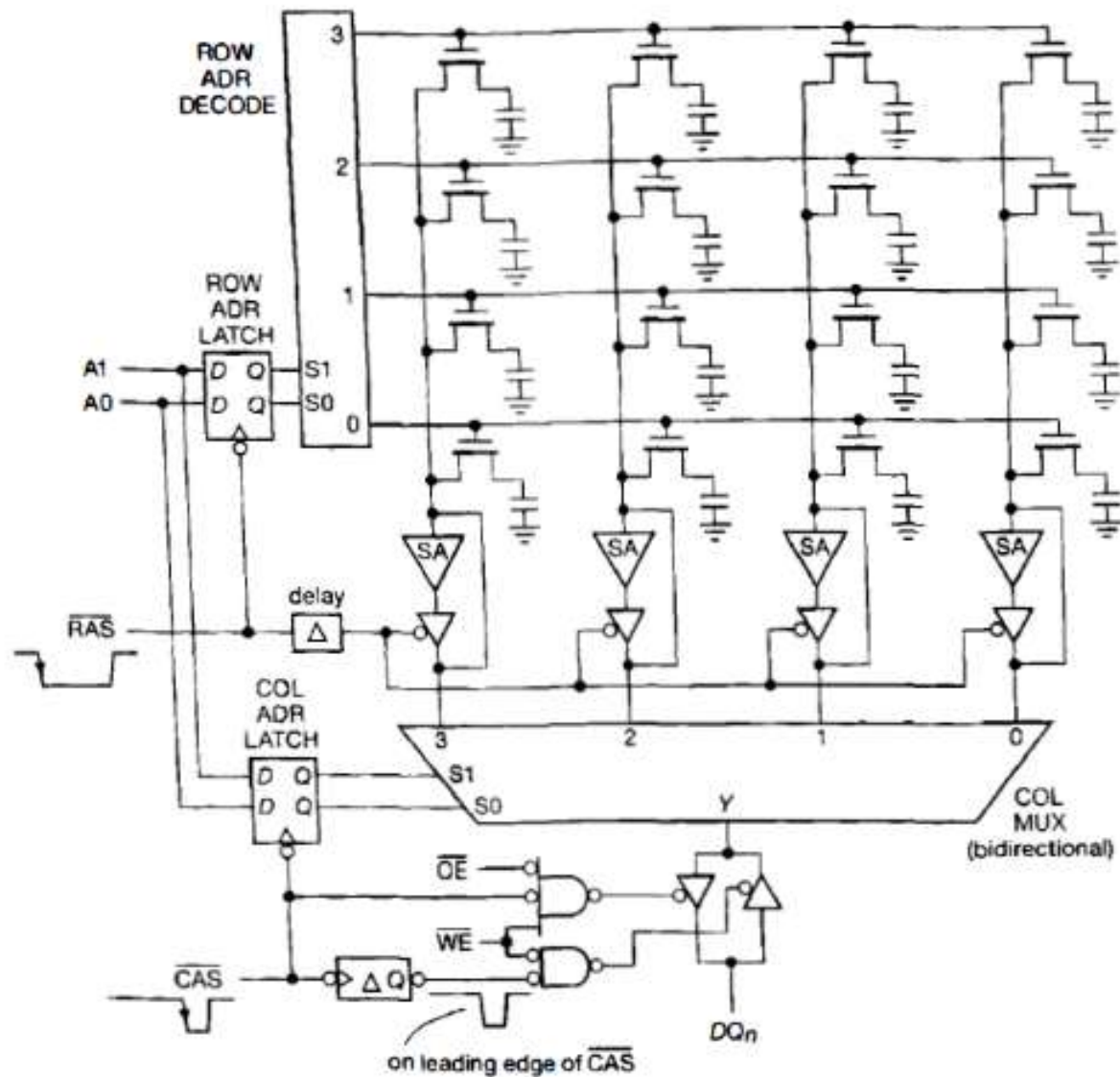
Dynamic RAM

We can save a lot of space on the chip by going to a one-transistor memory cell (with the state held on a small capacitor) if you're willing to carry out the periodic refresh of the capacitor's charge. That's dynamic RAM. with its "1T1C" memory cell.

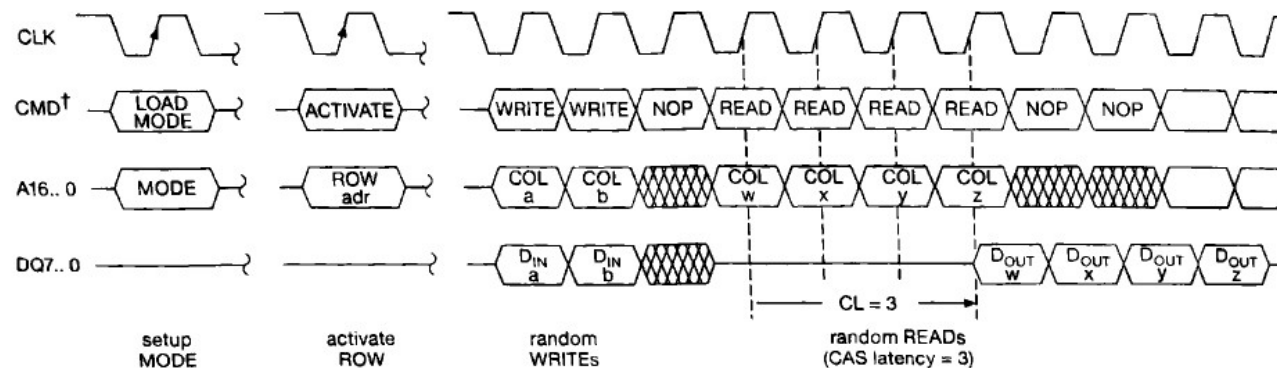


Dynamic-RAM "1T1C" bit cell. Each bit is held as a charged (~ 1 V) or discharged (0 V) capacitor, whose state is read, written, or refreshed by a bit line (BL) when asserted by a controlling word line (WL). Typically $C = 30$ femtofarads.

Asynchronous DRAM



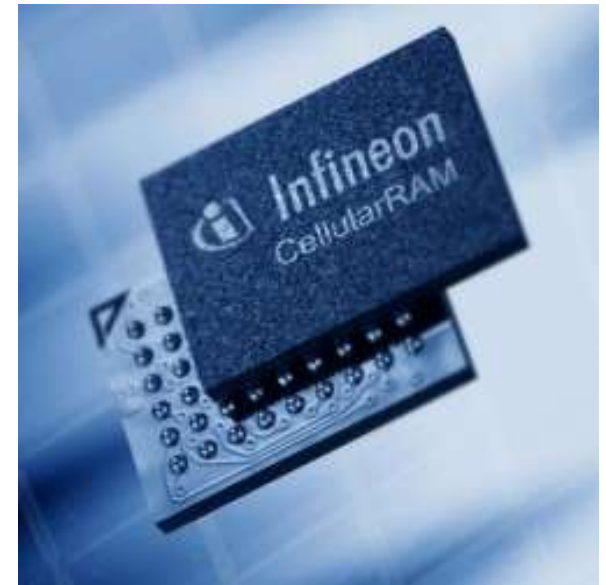
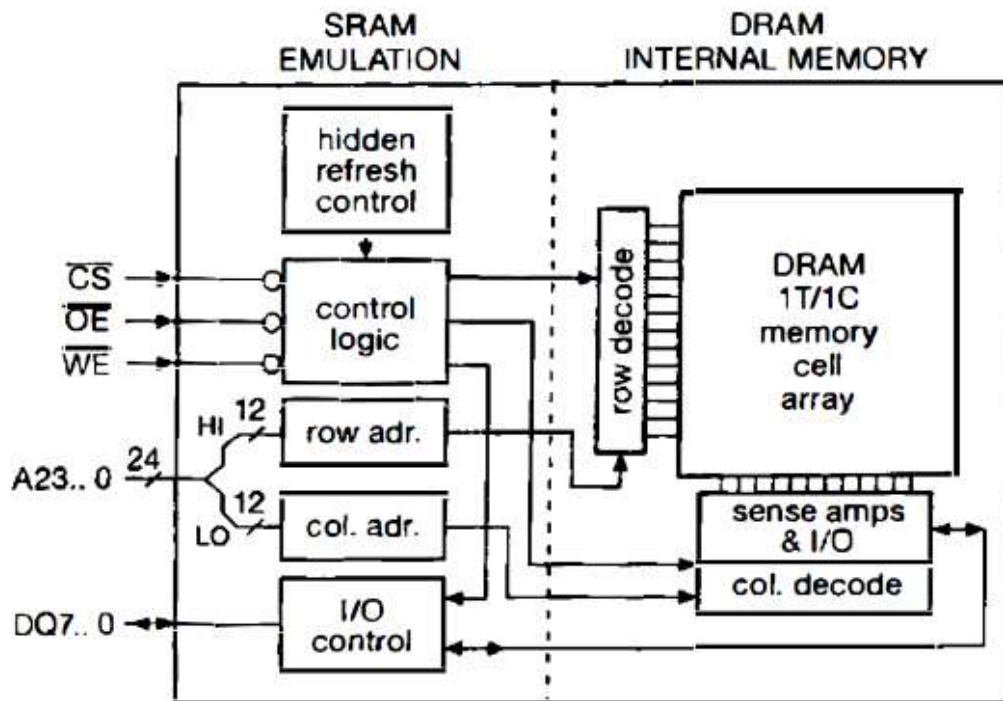
Synchronous DRAM



Feature	SDR	DDR	DDR2	DDR3
Data Transfers per Clock	1	2 (rising + falling edges)	2	2
Internal Prefetch	1-bit	2-bit	4-bit	8-bit
Typical Clock Speed	66–133 MHz	100–200 MHz	200–400 MHz	400–800 MHz
Voltage	3.3 V	2.5 V	1.8 V	1.5 V (1.35 V DDR3L)
Main Improvement	Baseline	Double-edge clocking	Higher prefetch + speed	Higher prefetch + efficiency
Typical Use	Early PCs, microcontrollers	Early DDR systems	Mid-generation PCs	Modern PCs, embedded systems

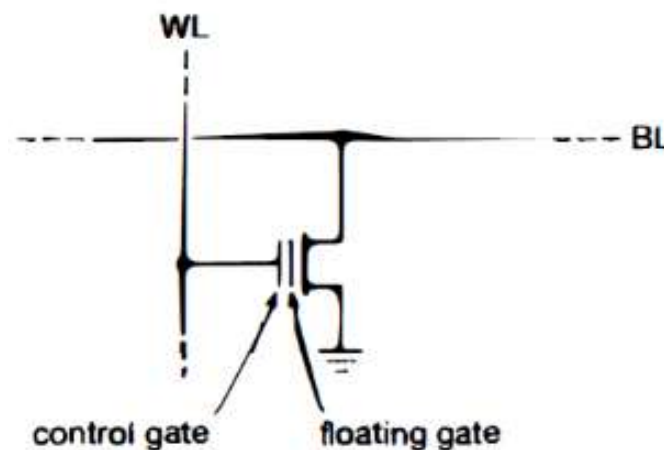
Pseudo Static RAM (CellularRAM)

"Pseudo static RAM" looks for all the world like a classic asynchronous SRAM. But its simple external interface conceals the truth: an efficient 1T/1C DRAM core, wrapped in an SRAM-emulating layer of logic (complete with hidden refresh)



FLASH/EEPROM CELL

"Floating-gate" bit cell, used in non-volatile memories such as EEPROM and flash ROM. Data is written to the floating gate by tunneling or hot-electron injection; its charge alters the threshold voltage, as read out by the control gate and bit line. The leakage currents out of the floating gate are so low that data is retained for at least ten years, requiring no power or refresh.

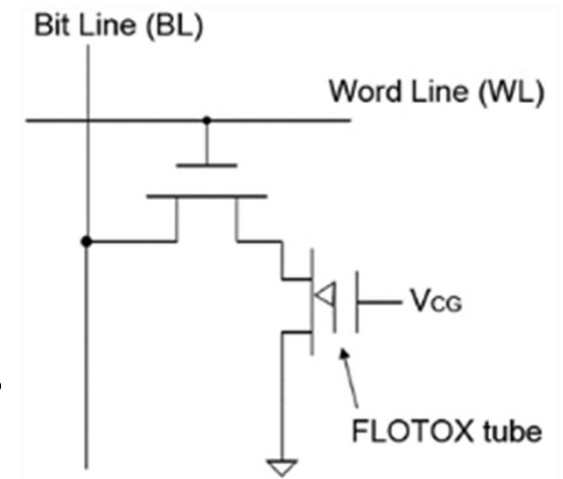


EEPROM

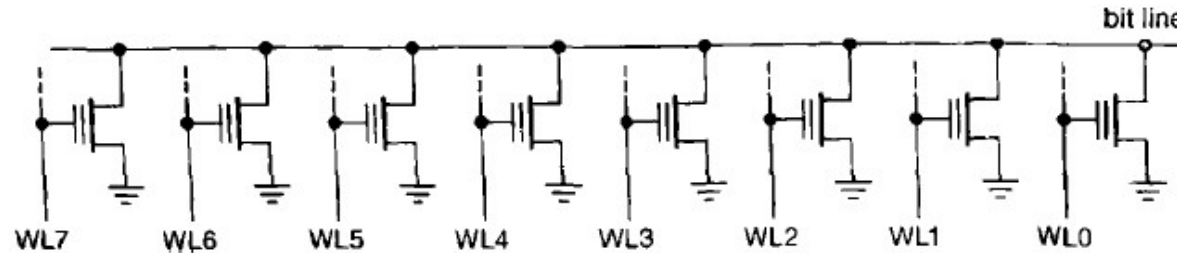
The modern era in non-volatile memory was ushered in with the electrically erasable programmable read-only memory (EEPROM).

The trick here is to use elevated voltages to discharge the floating gate, by a quantum-mechanical phenomenon known as *tunneling* (in which a particle, here an electron, with insufficient energy to overcome the potential barrier of the gate insulator can, under the right circumstances, just magically appear on the other side. It's officially called Fowler-Nordheim tunneling, usually abbreviated F-N.

EEPROMs use a memory-cell configuration (for example, a two-transistor bit cell) that permits **bits or words to be erased and reprogrammed individually**. That's a nice flexibility, but the circuitry that allows this flexibility takes up space that could be used for additional storage

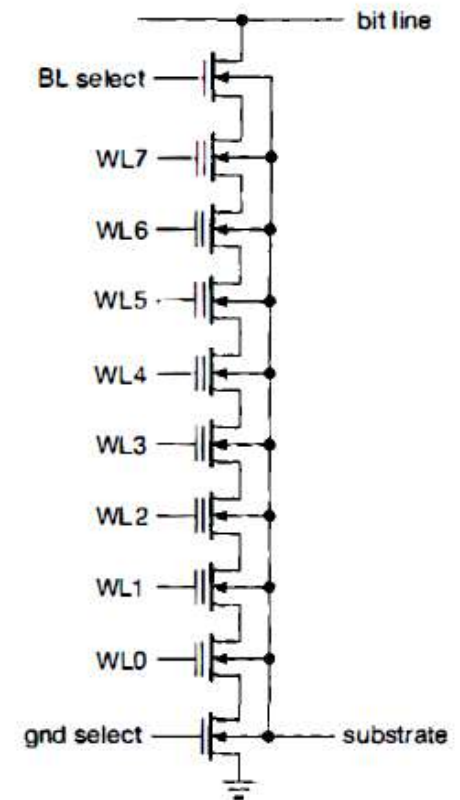


FLASH



Flash memory **discards the bit or byte writability of EEPROM**, instead performing block-sized erase. The good news is that it's faster (it erases "in a flash," hence the name), if you want to erase many bytes. The bad news is that you cannot modify smaller amounts of data.

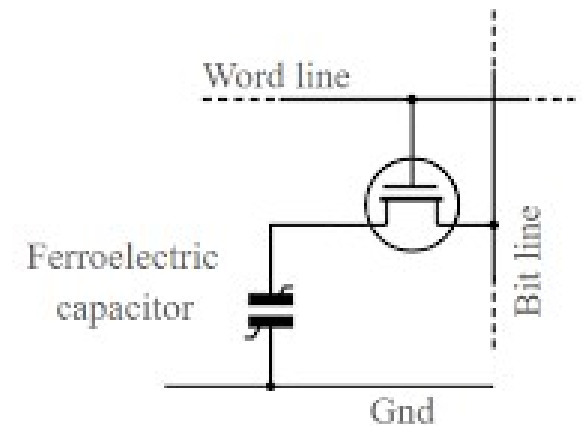
Currently available flash memory comes in two flavors, called NOR and NAND.



FRAM

A ferroelectric material is the electric analog of a ferromagnetic material; that is, it holds the state of electric polarization.

The idea is to make a 1T1C-style DRAM bit cell, but with the capacitor replaced with a thin film of ferroelectric material (e.g., a few atom layers of strontium bismuth titanate). You write a bit as with DRAM, applying a field across the film. Readout is different. however: you read "destructively," by sensing whether a write changed the state (producing a pulse of current); then you restore the state. FRAM potentially can deliver randomly.



Basic ferroelectric RAM, FRAM memory cell

MSP430 MCU with FRAM

- Ultra-Low-Power Ferroelectric RAM (FRAM)
 - Up to 16KB of Nonvolatile Memory
 - Ultra-Low-Power Writes
 - Fast Write at 125 ns per Word (16KB in 1 ms)
 - Built-In Error Correction Coding (ECC) and Memory Protection Unit (MPU)
 - Universal Memory = Program + Data + Storage
 - 10^{15} Write Cycle Endurance
 - Radiation Resistant and Nonmagnetic

