ECE 471 – Embedded Systems
Lecture 12

Vince Weaver

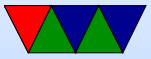
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

30 September 2024

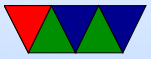
Announcements

- HW#4 was posted
- If you need any parts (LED, breadboard) let me know



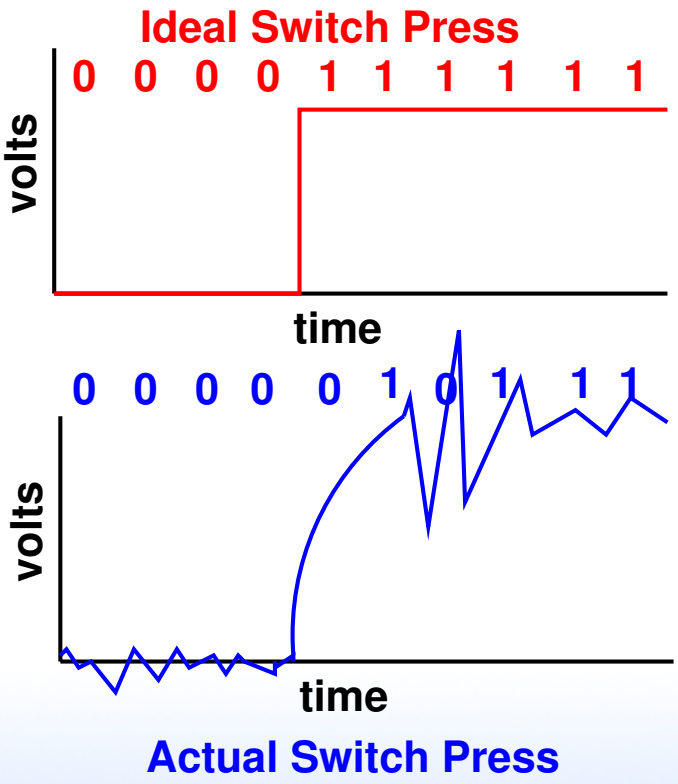
LEDs and resistors

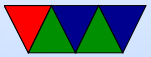
- Why not just hook an LED directly from GPIO to GND?
  Need to limit current, which a resistor does
  Does 5V vs 3.3V matter? Not as long as you don't over current and as long as above the turn-on voltage
- Why resistor when using GPIO as input?
  If there's a chance you can connect Vcc to GND directly that will cause a short and bad things can happen. Input by default high-impedance, but what if somehow accidentally configured for output of 0?



Debouncing (from last time)

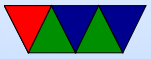
- Noisy switches, have to debounce





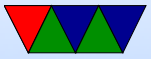
Debouncing!

- Can you fix in hardware?
  - Capacitors (not for homework, will be grading software)
  - Built-in debounce (shift-registers?) like on STM32L?
- Can you fix in software? Algorithms
  - Wait until you get X consecutive values before changing
  - Get new value, wait short time and check again
  - These all have tradeoffs and can get caught by different



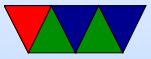
patterns of noise

- Don't read once and then just delay without reading again



(Review) How Executables are Made

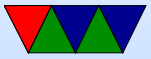
- Compiler generates ASM (Cross-compiler)
- Assembler generates machine language objects
- Linker creates Executable (out of objects)



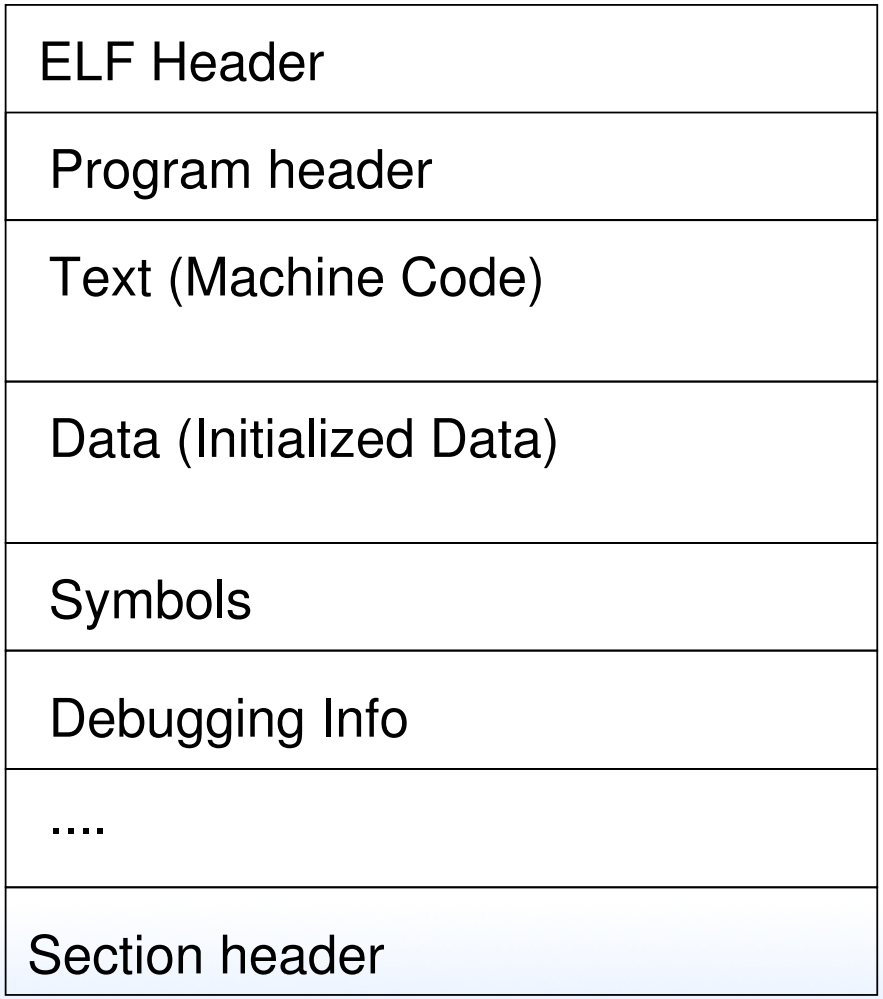
Executable Format

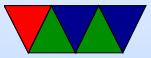
- ELF (Executable and Linkable Format, Extensible Linking Format)
  Default for Linux and some other similar OSes
  header, then header table describing chunks and where they go

- Other executable formats: a.out, COFF, binary blob



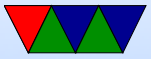
ELF Layout


ELF Header
Program header
Text (Machine Code)
Data (Initialized Data)
Symbols
Debugging Info
....
Section header



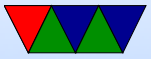
ELF Description

- ELF Header includes a "magic number" saying it's 0x7f,ELF, architecture type, OS type, etc. Also location of program header and section header and entry point.
- Program Header, used for execution:
  has info telling the OS what parts to load, how, and where (address, permission, size, alignment)
- Program Data follows, describes data actually loaded into memory: machine code, initialized data

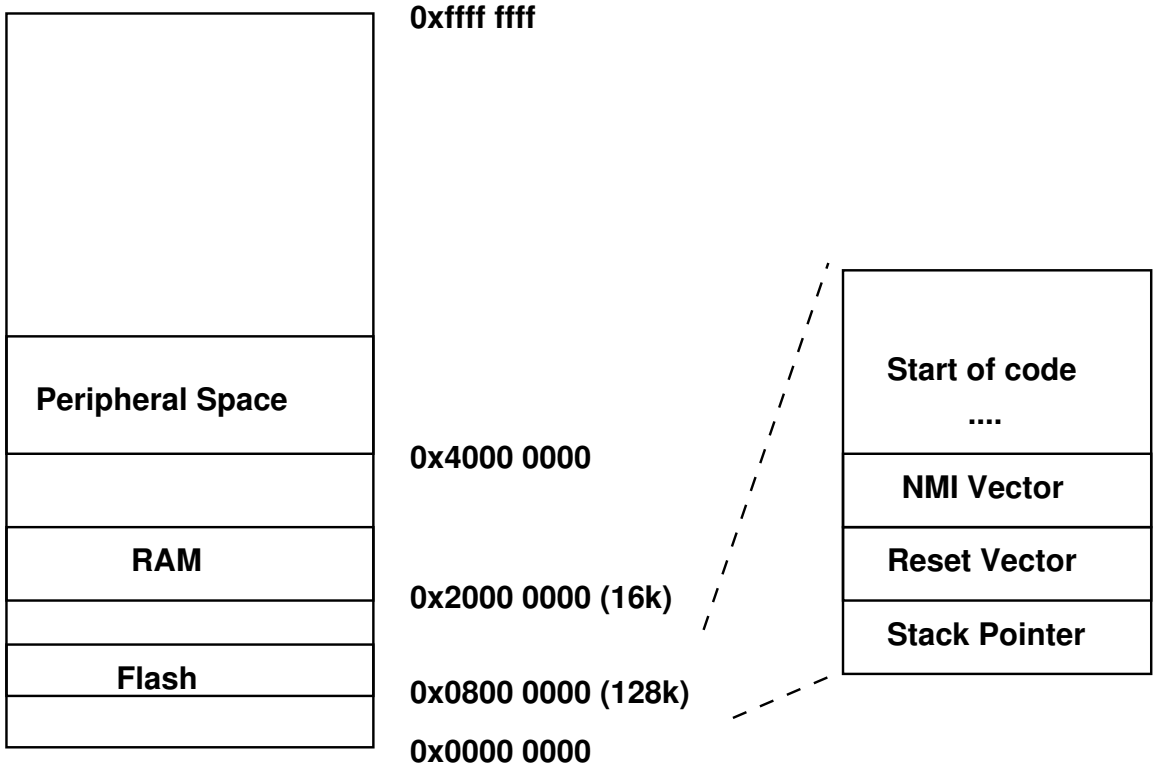


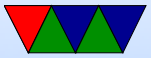
- Other data: things like symbol names, debugging info (DWARF), etc.
  DWARF backronym = "Debugging with Attributed Record Formats"

- Section Header, used when linking:
  has info on the additional segments in code that aren't loaded into memory, such as debugging, symbols, etc.

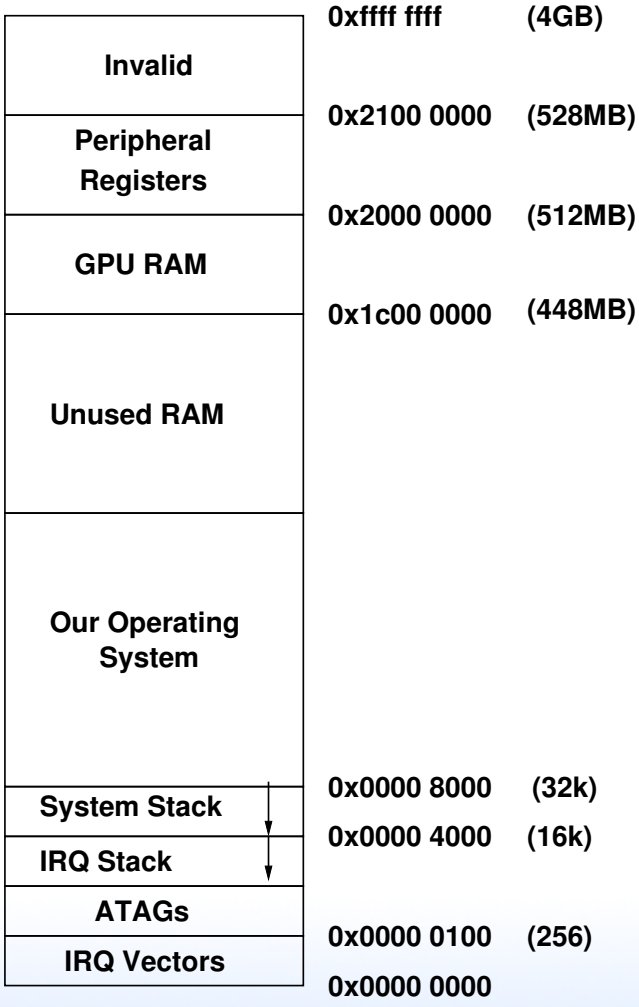


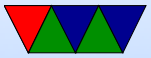
STM32L-Discovery Physical Memory Layout



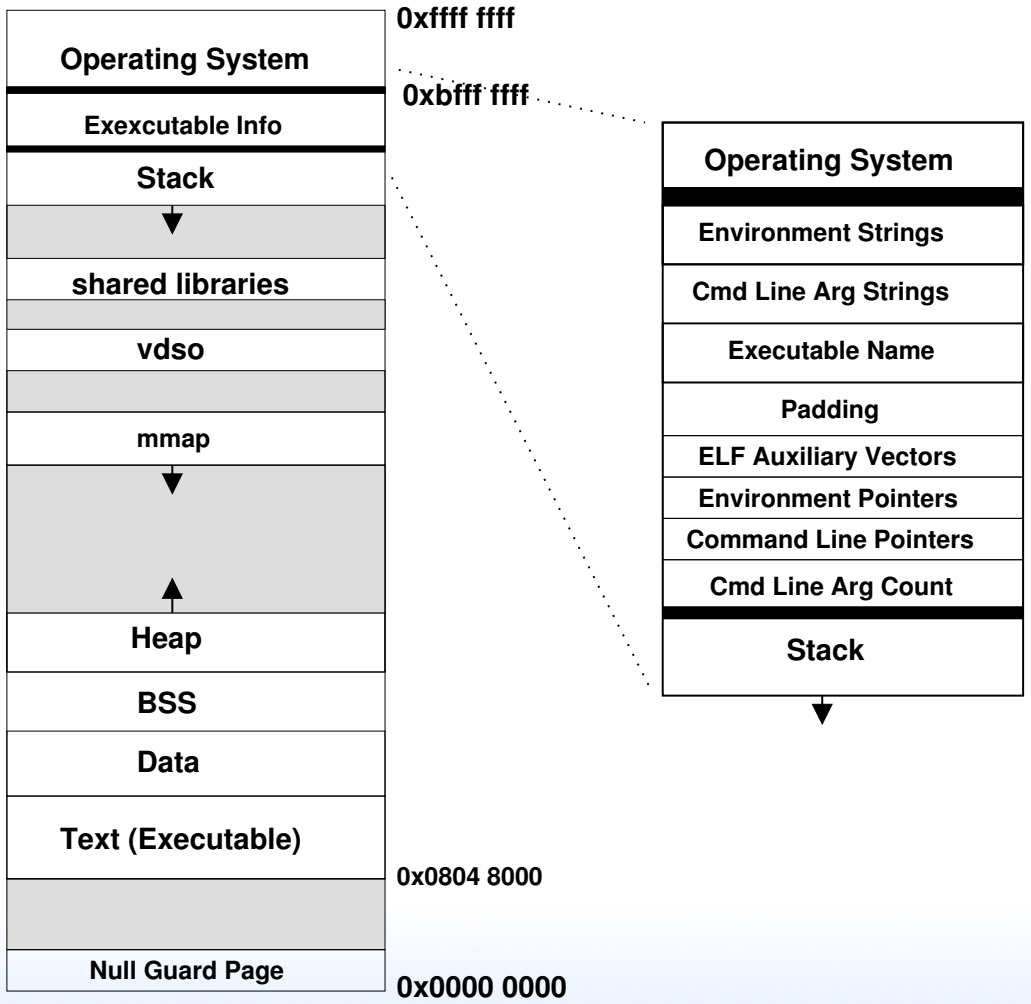


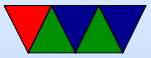
Raspberry Pi (32bit) Physical Layout





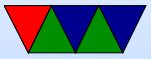
Linux 32-bit Virtual Memory Map





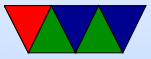
Program Memory Layout on Linux

- Text: the program's raw machine code
- Data: Initialized data
- BSS: uninitialized data; on Linux this is all set to 0.
- Heap: dynamic memory. `malloc()` (`brk()` syscall) and C++ `new()`. Grows up.
- Stack: LIFO memory structure. Grows down.



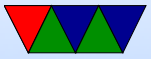
Program Layout

- Kernel: is mapped into top of address space, for performance reasons (but security...)
- Command Line arguments, Environment, AUX vectors, etc., available above stack
- For security reasons "ASLR" (Address Space Layout Randomization) is often enabled. From run to run the exact addresses of all the sections is randomized, to make it harder for hackers to compromise your system.



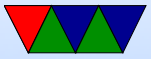
Brief overview of Virtual Memory

- Each program gets a flat 4GB (on 32-bit) view of memory
- CPU and Operating system work together to provide this illusion
- See this much RAM even if it is more than physically available (swapping/paging)

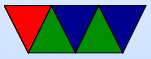


Physical vs Virtual Memory

- OS/CPU deal with "pages", usually 4kB chunks of memory.
- Every mem access has to be translated
- The operating system looks in "page table" to see which physical address your virtual address maps to
- This is slow. How to improve slow memory in CPU? Cache!
- TLB caches pagetable translations
- As long as you don't run out of TLB entries this goes

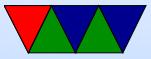


fast.



Benefits of Virtual Memory

- Demand paging: the OS doesn't have to load pages into memory until the first time you actually load/store them.
- Context Switch: when you switch to a new program, the TLB is flushed and a different page table is used to provide the new program its own view of memory.
- Flat memory space, all processes can start at same memory location without having to recompile
- Security: different processes can't see others memory (or



over-write it)