ECE 471 – Embedded Systems
Lecture 7

Vince Weaver

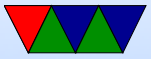
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

16 September 2020

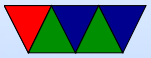
Announcements

- HW#2 was due
- Whill post HW#3
- Will send e-mail about parts distribution



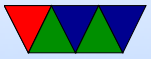
Why is Linux used in Embedded Systems?

- Linux is free (no per-copy cost)
- Source code is available.
- Linux on ARM is widely supported (although upstream support can be a mess)
- Lots of tools and experience
- Embedded systems have gotten more powerful



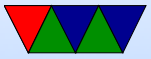
Free Software Licensing

- Linux under GPLv2.
- The Free Software Foundation has moved most of its software (including gcc compiler) to the less popular GPLv3 which most companies don't like.
- Companies often prefer BSD license which has fewer restrictions; can use code and release binaries without having to release the source (a GPL requirement).
- Apple and Google both trying to replace as much code as possible with BSD versions.



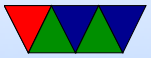
Free Software / Copyright Law

- Really need to be a lawyer
- Copyright automatically provided to creative works, prevent others from copying w/o permission
- You can provide a license that allows others to copy
- Commercial licenses (long tiny print, etc) or free licenses
- Copyright was originally 14 years + 14 year renewal
- Now author's life+70 years or 95-120 years if company
- Patents not the same
- Trademarks also not the same



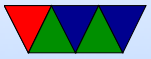
How Executables are Made

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



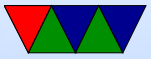
Tools – Compiler

- takes code, usually (but not always) generates assembly
- Compiler can have front-end which generates intermediate language, which is then optimized, and back-end generates assembly
- Can be quite complex
- Examples: gcc, clang
- What language is a compiler written in? Who wrote the first one?



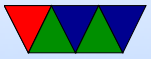
Tools – Assembler

- Takes assembly language and generates machine language
- creates object files
- Relatively easy to write
- Examples: GNU Assembler (gas), tasm, nasm, masm, etc.



Tools – Linker

- Creates executable files from object files
- resolves addresses of symbols.
- Links to symbols in libraries.
- Examples: ld, gold

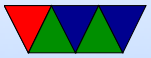


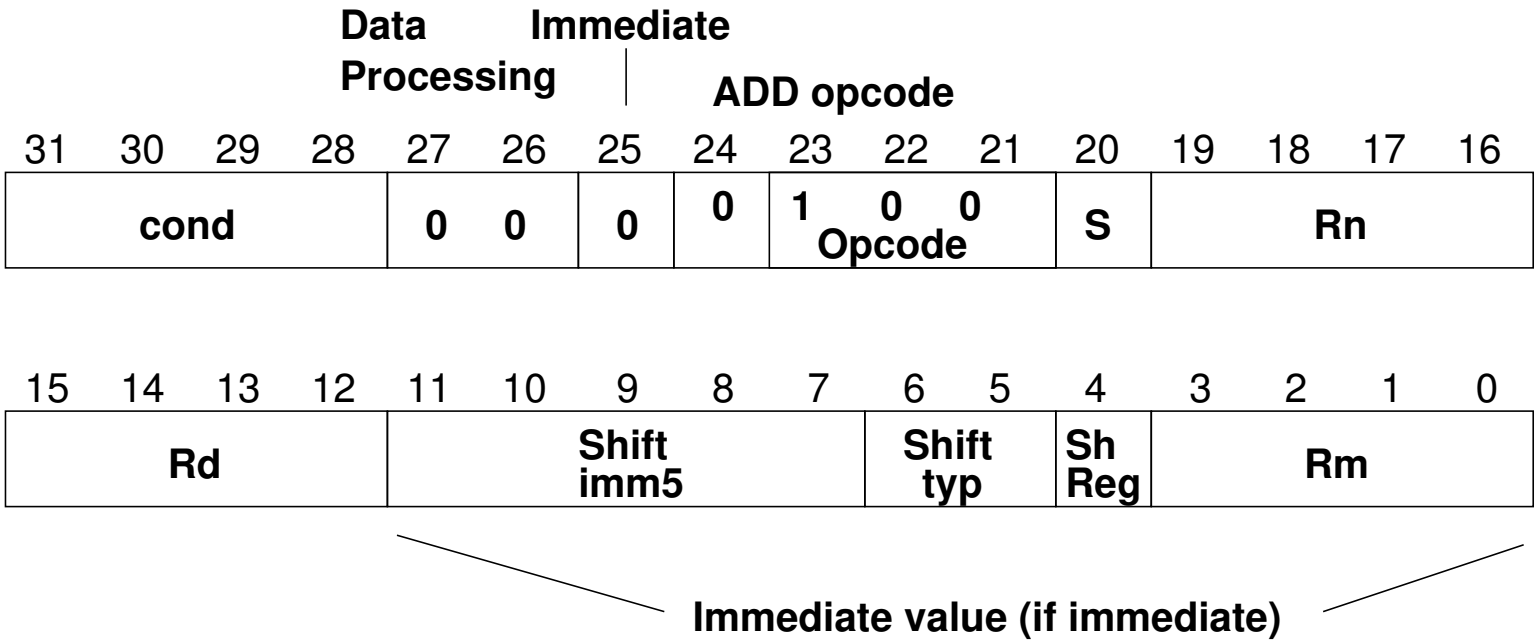
Converting Assembly to Machine Language

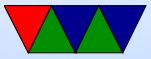
Thankfully the assembler does this for you.

ARM32 ADD instruction – `0xe0803080` == `add r3, r0, r0, lsl #1`

```
ADD{S}<c> <Rd>,<Rn>,<Rm>{,<shift>}
```



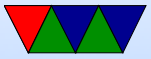

Data Processing
Immediate
ADD opcode
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
cond
0 0
0
0
1 0 0
Opcode
S
Rn
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Rd
Shift imm5
Shift typ
Sh Reg
Rm
Immediate value (if immediate)



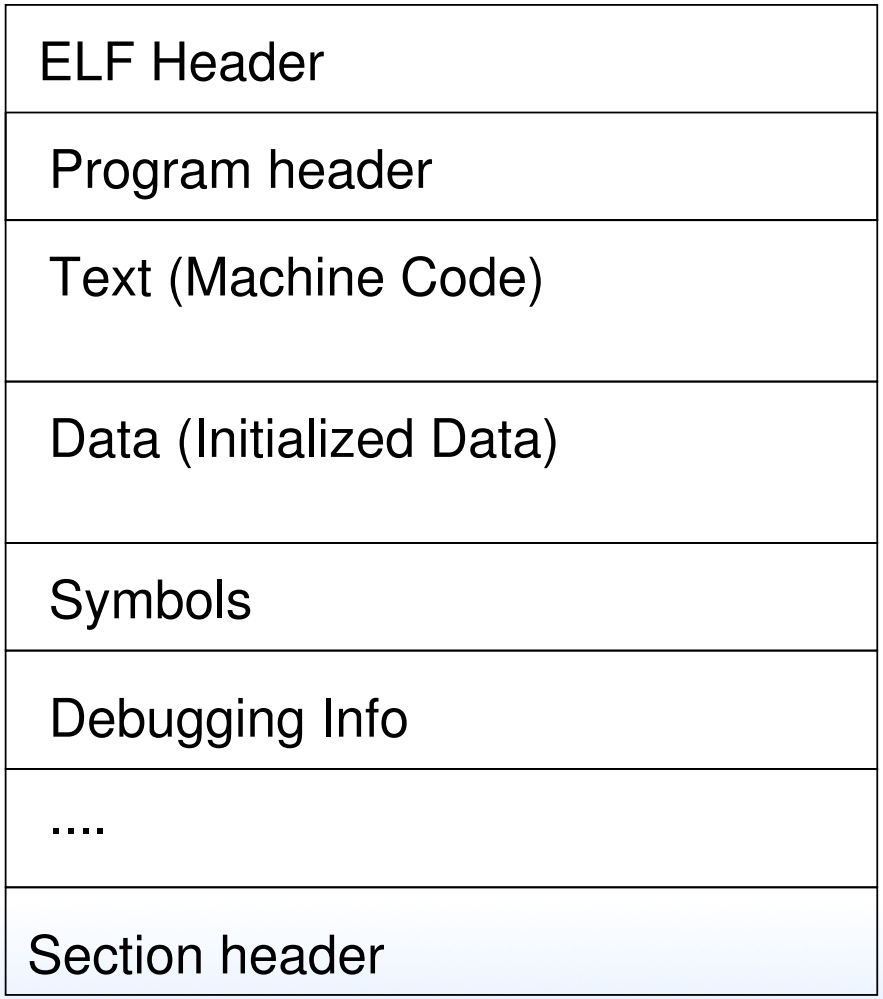
Executable Format

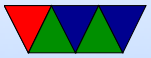
- ELF (Executable and Linkable Format, Extensible Linking Format)
  Default for Linux and some other similar OSes
  header, then header table describing chunks and where they go

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



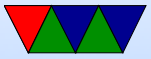
ELF Layout


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



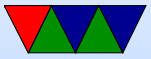
ELF Description

- 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

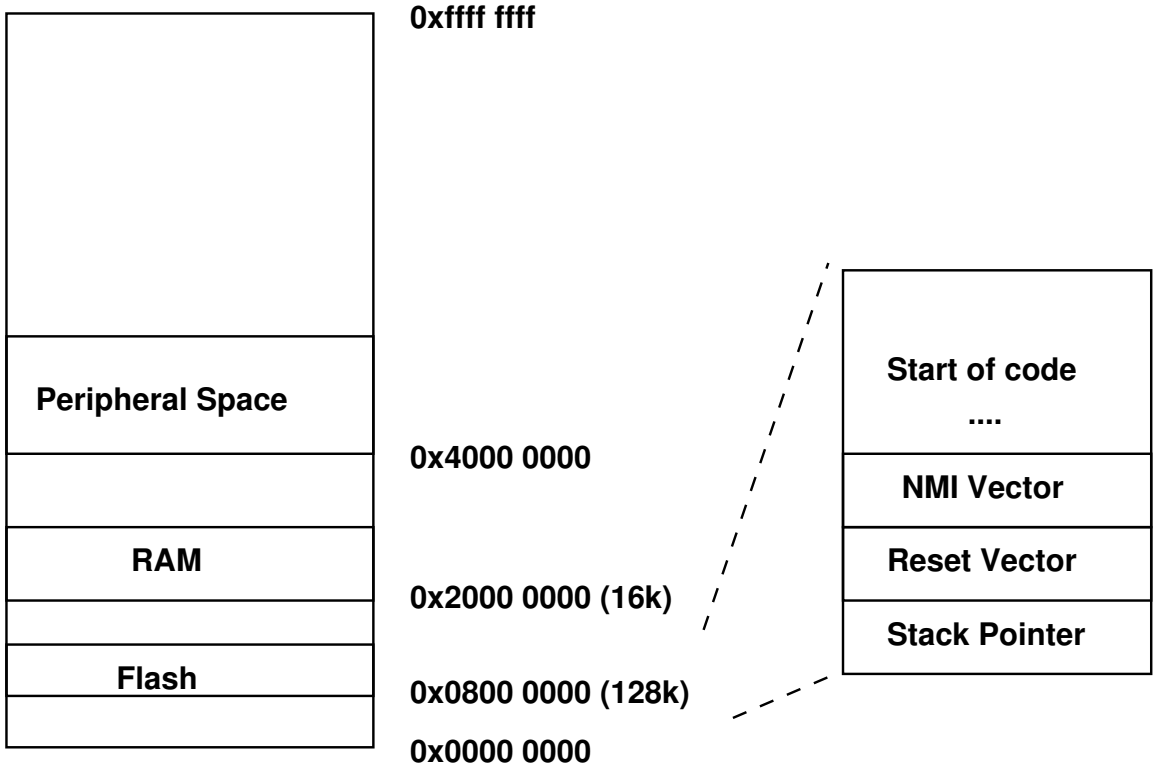


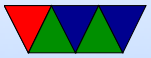
- Other data: things like symbol names, debugging info (DWARF), etc.
  DWARF backronym = "Debugging with Attributed Record Formats"

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

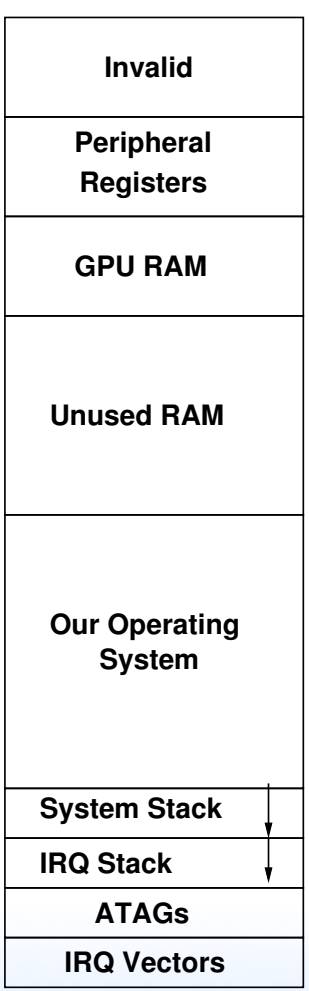


STM32L-Discovery Physical Memory Layout





Raspberry Pi Layout



0xffff ffff (4GB)

0x2100 0000 (528MB)

0x2000 0000 (512MB)

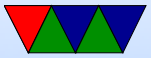
0x1c00 0000 (448MB)

0x0000 8000 (32k)

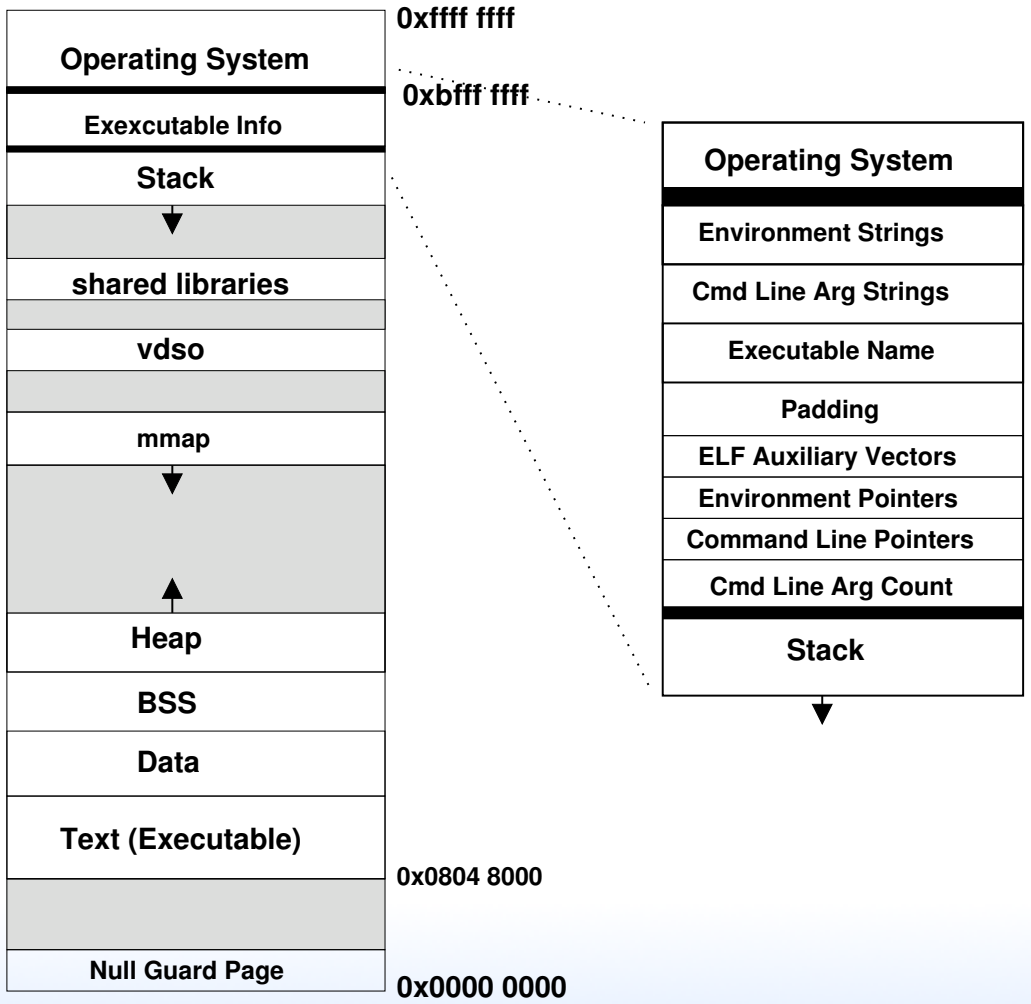
0x0000 4000 (16k)

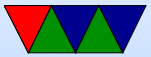
0x0000 0100 (256)

0x0000 0000



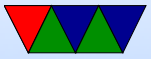
Linux Virtual Memory Map





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
- 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.