Intro to Real-Time Embedded Systems

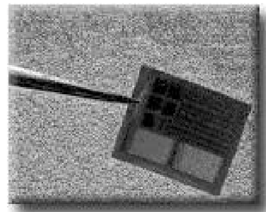
Hierarchy of Programming Languages

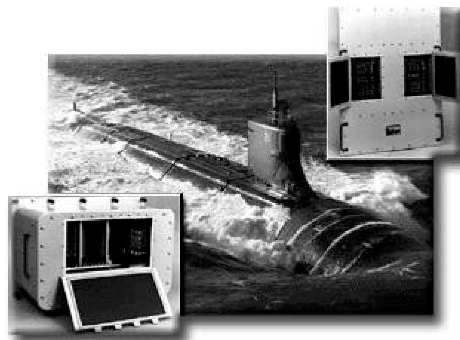
Understanding Compilers

DRAPER LABORATORY

Real Time Embedded Systems

- Multiple platforms-





- Multiple languages- C, C++, Ada, ASM…
- Common themes:
  - Critical deadlines
  - Well specified tasks, *engineered* solutions
  - Squeeze top performance from hardware suited to (or designed for) the task at hand

DRAPER LABORATORY

Real-time systems:

- Operate under more severe constraints than normal software systems
- Must perform reliably over long periods of time
- Most must operate with minimum memory footprint, minimum support hardware
- Real-time doesn't necessarily mean *fast*. It means *schedulable and bounded*.
- Late data is at best, worthless. At worst, it's *bad data*.
  - example: GPS info coming from receiver into navigation system

DRAPER LABORATORY

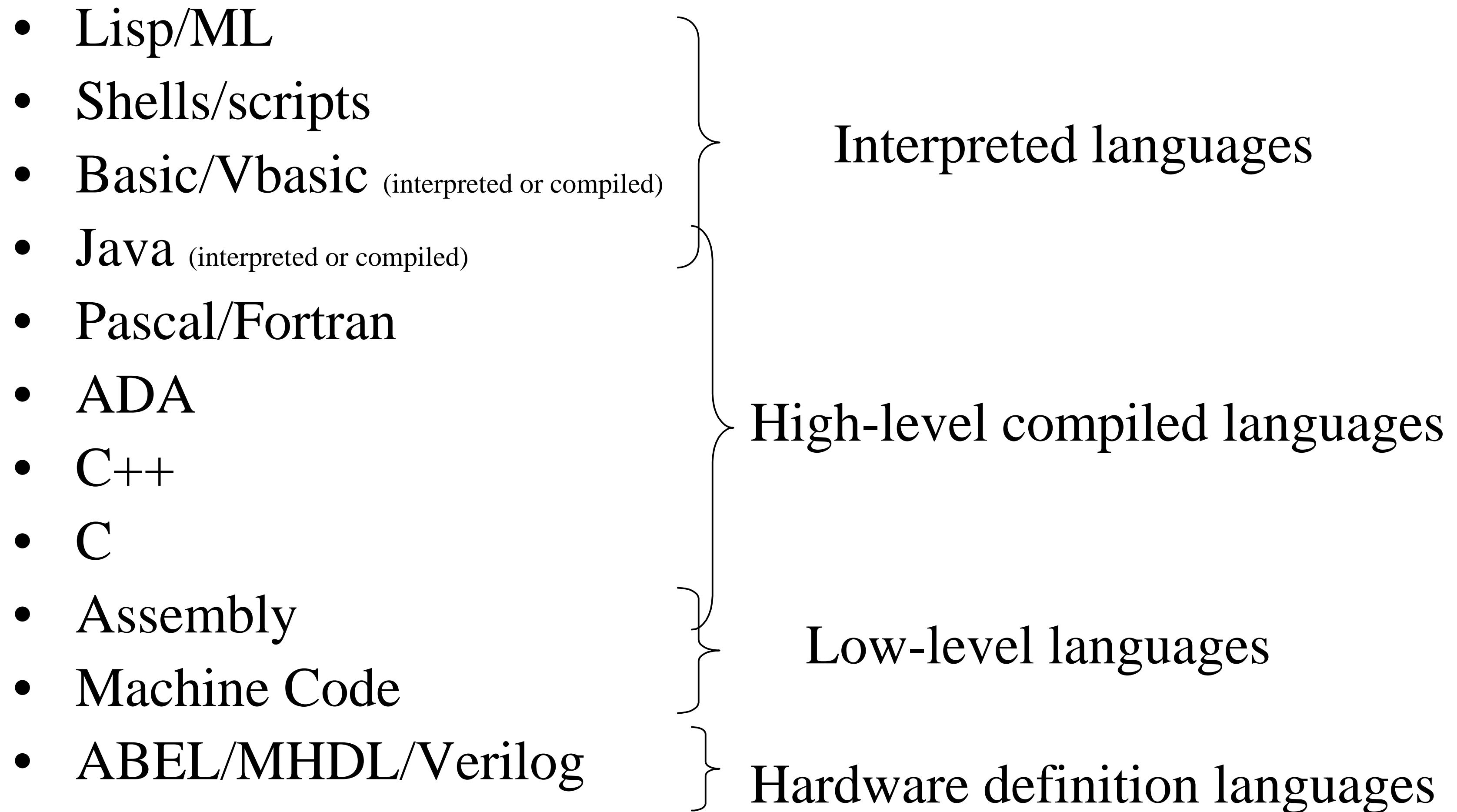
Real-time systems design:

- Select hardware, developing environment, and software appropriate to the task and talent at hand
- Develop performance metrics for system (which parts need to run at what speeds) and create prototypes for the most critical systems
  - *You may find yourself developing software for hardware that doesn't exist yet. It's more challenging that way.*
- Adjust hardware and software as necessary to meet requirements

DRAPER LABORATORY

Hierarchy of Languages

(very incomplete list)



DRAPER LABORATORY

Interpreted Languages

- Instructions interpreted by running program (BASIC interpreter, JAVA sandbox, Tcl shell interpreter)
- Interpreter translates to machine code instructions "on the fly"
- Performance is unpredictable- entire class of languages inappropriate for real-time programming.
- Interpreters too large for most embedded systems

DRAPER LABORATORY

High-level compiled languages

- Instructions support high-level paradigm (easy for you to understand, hard for the computer):
  - Object Orientation
  - List Processing
- May contain large libraries of tools that make your job easier
- Compiler creates “Object Code” (machine code), links in libraries, adds executable “stubs”
- Many interpreted/scripting languages now have compilers to create platform-specific object code (for speed and efficiency) i.e. perl, Matlab, Java

DRAPER LABORATORY

High-level compiled languages (cont.)

- Many high-level compilers have adjustable "optimization levels"- use graph theory and other techniques to optimize resulting code. Results vary by platform, source code, phase of moon, etc.
- High level languages require no knowledge of underlying system (uh- that's kind of the point).
- Code may be portable from one platform to another, but will require re-compiling.

DRAPER LABORATORY

Low-level languages

- Low-level languages grew from machine code and maintain similarities to computer architecture
- Greater control over resulting machine code means predictable execution.
- C is considered a low-level language by some because the data types and operations map directly to processor features.

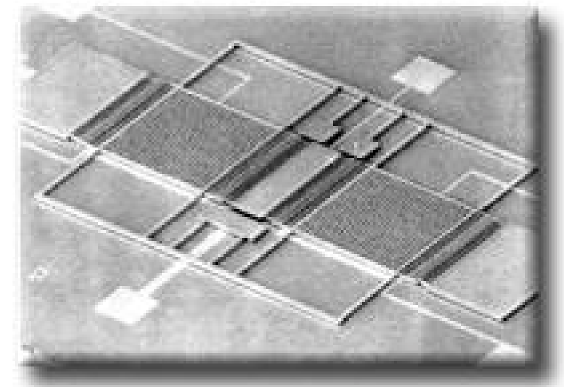
DRAPER LABORATORY

Low-level languages (cont.)

- Low-level languages are usually hardware-specific and non-portable. Assembly languages have similarities, but are non-portable from one architecture to another.
- Assembly languages access chip-specific features, and are the fastest programming languages available.

What if assembly isn't fast enough?

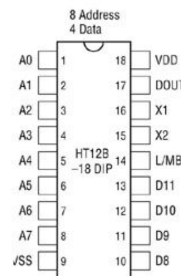
- *Application Specific Integrated Circuit (ASIC)* is a catch-all term for ICs (chips) that can be customized to meet your needs.

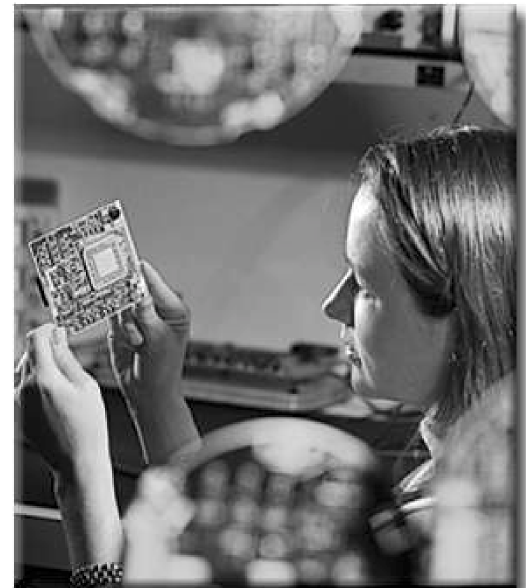


- You can code your solution in a *Hardware Definition Language (HDL)* and use the compiler to produce electrical schematics or code that is burned onto a device

Hardware Definition Languages

- Language defines logic and/or layout of components on a logic chip
- Language is sent to a fabrication lab to create the circuitry, or downloaded into a PAL (programmable array logic), EEPROM (we'll talk about these later) or FPGA.
- Two weeks later, you get an IC (*Integrated Circuit*) that you can plug into a board and *voila*- your code runs at electron speed.



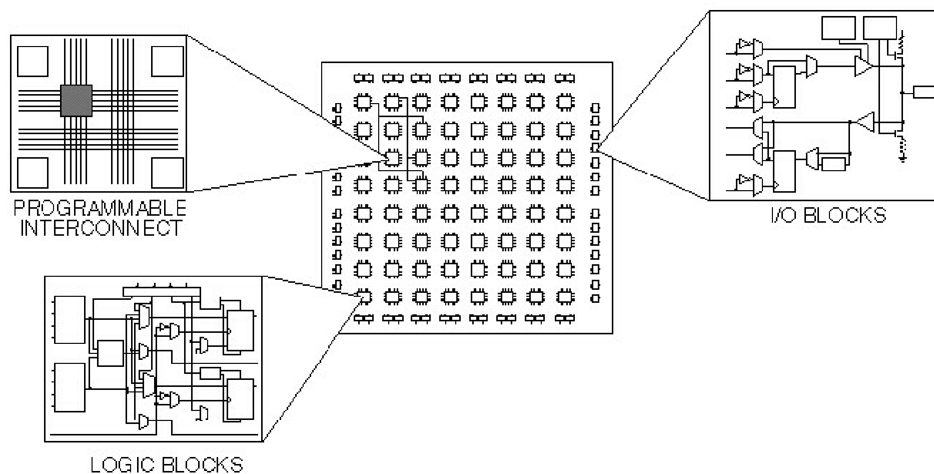


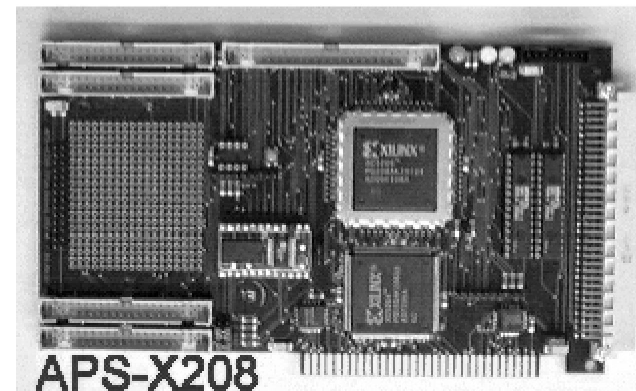
FPGA

- A young but growing technology for prototyping embedded systems
- Instead of spending the money ($5000+ ea) to build ASICS, a company can buy or build a board that supports an FPGA.
- A *Field Programmable Gate Array* is, in effect, a simulation of an IC. You write your program in a c-like language, or as an electronic schematic diagram, and download it to the FPGA. The chip then acts like a custom-made electronic device.

FPGA

- FPGA chip loads "program" (layout of logic gates and circuit traces) from ROM chip or over network. Modifying circuit layout is a matter of changing "program".
- Xilinx Spartan II: 100,000 logic gates on a $10, 200 Mhz chip





DRAPER LABORATORY

Translation of C to Machine Code

The 16.070 development environment

(*C code, developed on a workstation simulator, cross-compiled to a single-board computer (SBC) powered by a well-known processor*)

is one of the most common in the industry

Why???

DRAPER LABORATORY

Why is this the most common environment?

- ✓ Very large base of C programmers and large body of general knowledge.

  (comparatively few real-time gurus, though)

- ✓ C compilers have been under development for decades. Many optimization techniques are built in.

- ✓ Low cost parts and cheap (free!) development tools from Motorola, Intel, etc.

- ✓ It's easier to hack assembly generated from a C compiler than from VBasic!

DRAPER LABORATORY

What does the compiler do?

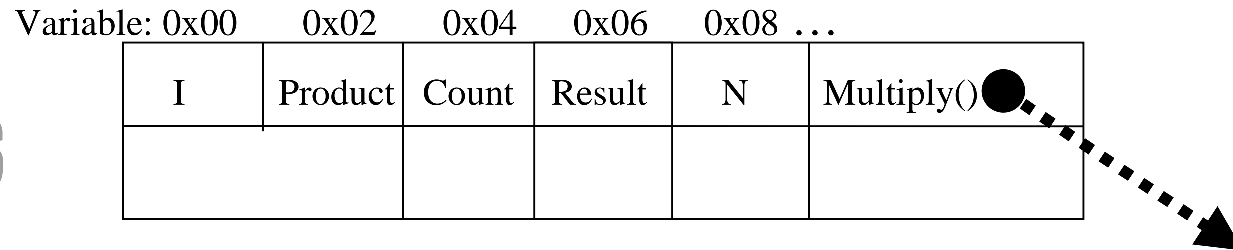
```
int fact(int n)  /* find n! */
{
  int product=1, count=2;
  while (n >= count)
  {
    product *= count;     /* =2*3*…*n */
    count ++;
  }
return product;
}
```

DRAPER LABORATORY

```
int fact(int n){int product=1,count=
2;while(n>=count)product*=count;coun
t++;}return product;}
```

Precompiler:
Removes comments, whitespace
Expands macros, #defines

Pass 1:
Sets up tables, link points

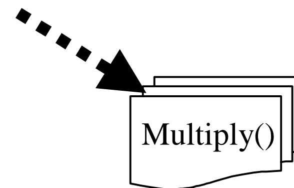


Variable: 0x00	0x02	0x04	0x06	0x08 …	
I	Product	Count	Result	N	Multiply()

```
ldd 0x02, #1
ldd 0x04, #2
LABEL 1:
cpd 0x08, 0x04
blt LABEL 2:
```

Pass 2:
ASM created from tables

Linker:
Library functions resolved



DRAPER LABORATORY

…EC021AEE04BD0000ED02EC04C30001ED04EC081AA3042
CE8EC02…

If you were to look at your code when the compiler was finished, it would look something like the string of Hex above or the string of Binary below. The Binary is what gets written into the memory of the computer, ready for the processor to get the first instruction and start following the program. This is a portion of the actual machine code created on an SGI Challenge by the gcc compiler for the C code on the previous slides.

…1110110000000001000011010111011100000010010111101000
0000000000000011101101000000101110110000000100110001…

```
{
 product =
 product * count;

count++;
}

while
(n >= count);
```

```
L2:
    ldd 2,x        Load Var #2
    ldy 4,x        Load Var #4
    mult           Multiply them
    std 2,x        Store result
    ldd 4,x        Load Var #4
    addd #1        Add 1 to it
    std 4,x        Store result
L3:
    ldd 8,x        Load var #8
    cpd 4,x        Compare to #4
    bge L2         Branch to L2 if >=
    ldd 2,x
L1:
```

```
0010
0010 EC02
0012 1AEE04
0015 BD0000
0018 ED02
001A EC04
001C C30001
001F ED04
0021 L3:
0021 EC08
0023 1AA304
0026 2CE8
0028 EC02
002A L1:
```

Why learn assembly?

- Inefficiencies in compiler
- Trim lines or µsecs
- Better understanding of timing/sizing issues
- Do-it-yourself memory management/scheduling

  (we'll talk more about this later)
- Take advantage of chipset features
- Interface with high-speed devices

- A good assembly programmer can still code circles around a good optimizing compiler.

  (of course, it takes the programmer a week or two longer)

DRAPER LABORATORY

Next time:

- 68HC11 assembly basics
- Optimization techniques
- Please read “Basic Concepts of Real Time Systems” and review lecture #9

DRAPER LABORATORY

Resources:

68HC11 Assembly language manuals and info:

http://www.owlnet.rice.edu/~elec201/Book/6811_asm.html

http://tomdickens.com/68hc11/manuals/instructions.html

http://mot-sps.com/mcu/documentation/

http://handyboard.com/techdocs/