Fundamentals of Embedded & Real-Time Systems

EMBSYS CP100

Dave Allegre

October 10, 2016

About Me

- In embedded systems for about the last 15 years.
- 20 years in the solar industry.
- Advanced application support for power inverters.
- Currently work for OutBack Power
- Wired remotes
- Automatic generator start module
- Three bank marine charger

About Me

ME-ARC

ME-AGS

About Me – Contact Information

- Email: allegre@uw.edu
  - Checked a couple times a day.
- Phone: (425) 405-5788
  - Leave a voicemail if I don't answer.
- Discussion Forums
  - Checked a couple times a day. This should be your first resource!
- LinkedIn
  - www.linkedin.com/in/dallegre

Ground Rules

- Attendance
  - UW policy is you must attend 8 out of 10 classes.
  - Attendance will be taken at the beginning of class.
- Etiquette
  - Turn off cell phone! If you need to take a call, step out of the room.
  - Private conversations can be distracting to your fellow students.
  - We have online students, that can hear just about everything.
  - No advertisements. You can recommend relevant articles, books, products, or services as long as you are not selling them.
  - Students logged into Adobe Connect: Mute your mic. In class students can cause feedback issues.

Ground Rules - Continued

- Be patient
  - We come from different backgrounds.
- Ask questions in class and online discussion forums.
  - Help others. Use each other as a resource.
- Collaboration
  - Please don’t collaborate on homework, quizzes, or the final exam.
  - BUT! You may ask and answer questions, and share ideas.
  - Do not post your solutions or source code.
- Plagiarism
  - Not tolerated. Give credit if you use someone else’s work.
  - Can hurt your academic career
  - Can hurt your career, or your company.

About You!

- > Your turn to tell me about you…
- > Quick Introduction
  - – Name
  - – Are you working? And where?
  - – Any experience with embedded systems?
  - – Other?
- > Online:
  - – Type your responses in chat.

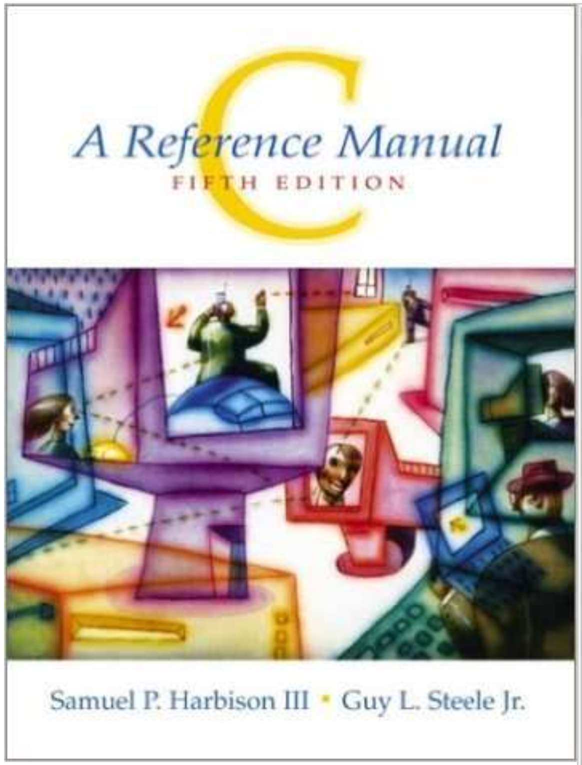
How This Course Works

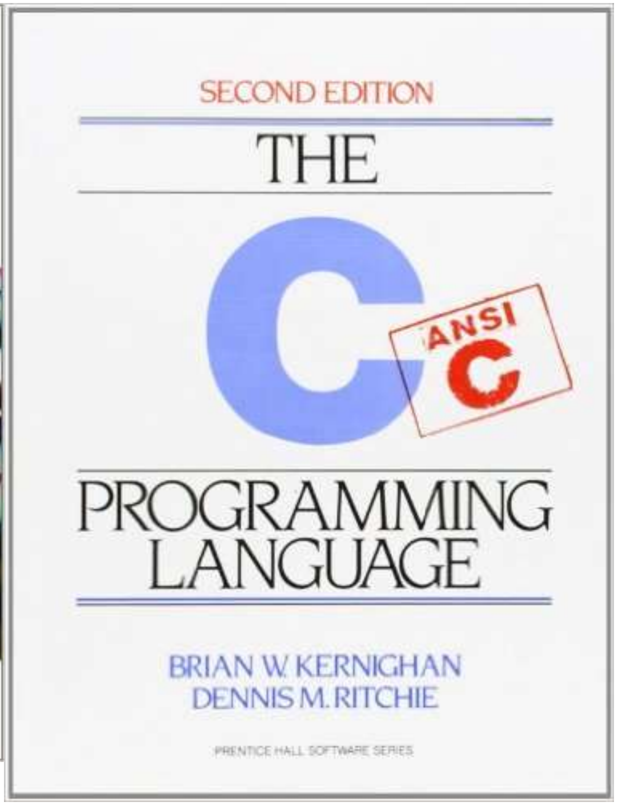
- Lectures
  - Once per week
  - Focus is geared towards hand-on learning
- Quizzes
  - Help me gauge our progress and identify potential problems
- Labs and homework
  - Labs are in-class and a starting point for the homework
  - Homework is your chance to extend the lab on your own
- Discussions
  - Weekly topics related to the course material

Prerequisites and Non-prerequisites

- Prerequisites
  - C programming
    - Proficient in C or C++
    - If your last experience was some time ago, consider picking up a book
    - Pointers – A good refresher
- Non-prerequisites
  - Computer architecture
  - Hardware knowledge
  - Assembly language programming

C Programming Books





C: A Reference Manual – 5th Edition, ISBN-13: 978-0130895929

The C Programming Language – 2nd Edition, ISBN-13: 978-0131103627

Certificate Program Summary

- Three course series
- EMBSYS CP100: Fundamentals of Embedded & Real-Time Systems
  - Fall of 2016
  - Instructor: David Allegre
- EMBSYS CP105: Programming with Embedded & Real-Time Operating Systems
  - Winter 2017
  - Instructor: Nick Strathy
- EMBSYS CP110: Design & Optimization of Embedded & Real-Time Systems
  - Spring 2017
  - Instructor: Lawrence Lo

EMBSYS CP100 Summary

- Embedded programming fundamentals
  - Number systems
  - Computer math
  - Digital logic
  - Assembly language and C
- ARM Core
  - Programmer’s model
  - Instruction set
  - Exception model
- Fundamental I/O device programming
  - Parallel I/O
  - Serial I/O
  - Interrupt-driven I/O

EMBSYS CP105 Summary

> Programming with a Real-Time Operating System (RTOS)
  - Tasks
  - Inter-task communications
  - Memory Management
  - Context switching
> Extending RTOS with a device driver framework
  - Use a standard device driver API
  - Write a device driver to handle devices
  - Advanced I/O

EMBSYS CP110 Summary

> Roll your own OS
> Optimizing designs
> Real-Time scheduling
> Memory
> Power management
> Open to your imagination!

EMBSYS CP100 Goals (High level)

- Course orientation & Intro to hardware. Computer math
- Nucleo STM32F401 kit. Digital logic.
- Leverage the use of the developments tools
- Software interactive with hardware
- ARM core and assembly language intro
- Parallel I/O and assembly language
- Serial I/O and embedded C
- Timers and counters
- Interrupts and Interrupt driven I/O
- Exceptions and system startup
- Guest speaker?

Assignments

- Assignments
  - Programming
  - Online discussions
  - “Book” assignments.
- Due on date posted
  - Post questions in the assignment forums for help well in advance of the due date.
  - Turning in late could get less instructor feedback.
- Grading within a reasonable time frame.

Assignments
Programming Guidelines

- Project sources turned in maybe checked to see if they build and run.
- Follow instructions included in each assignment
- If you’re stuck?
  - Try the forums first.
    - Somebody else might be having the same problem.
  - Answer questions in the forums.
    - Your experience might be a help to others

Final Exam

> “Take home”

> Due on last day of class

> Open book, open notes and most likely, online.

Grading

- Programming assignments – 50%
  - Hands on portion
  - Incomplete or erroneous projects may be returned for correction (and more points).
- Non-Programming – 25%
  - Quizzes
  - Discussion Board Participation
- Final Exam – 25%
  - No collaboration
- Why?

Canvas

- > https://canvas.uw.edu
  - Need your UWNetID and password for access
  - Announcements, assignments, homework, lecture slides and discussion forums.
- > Quick tour.

Required For Class

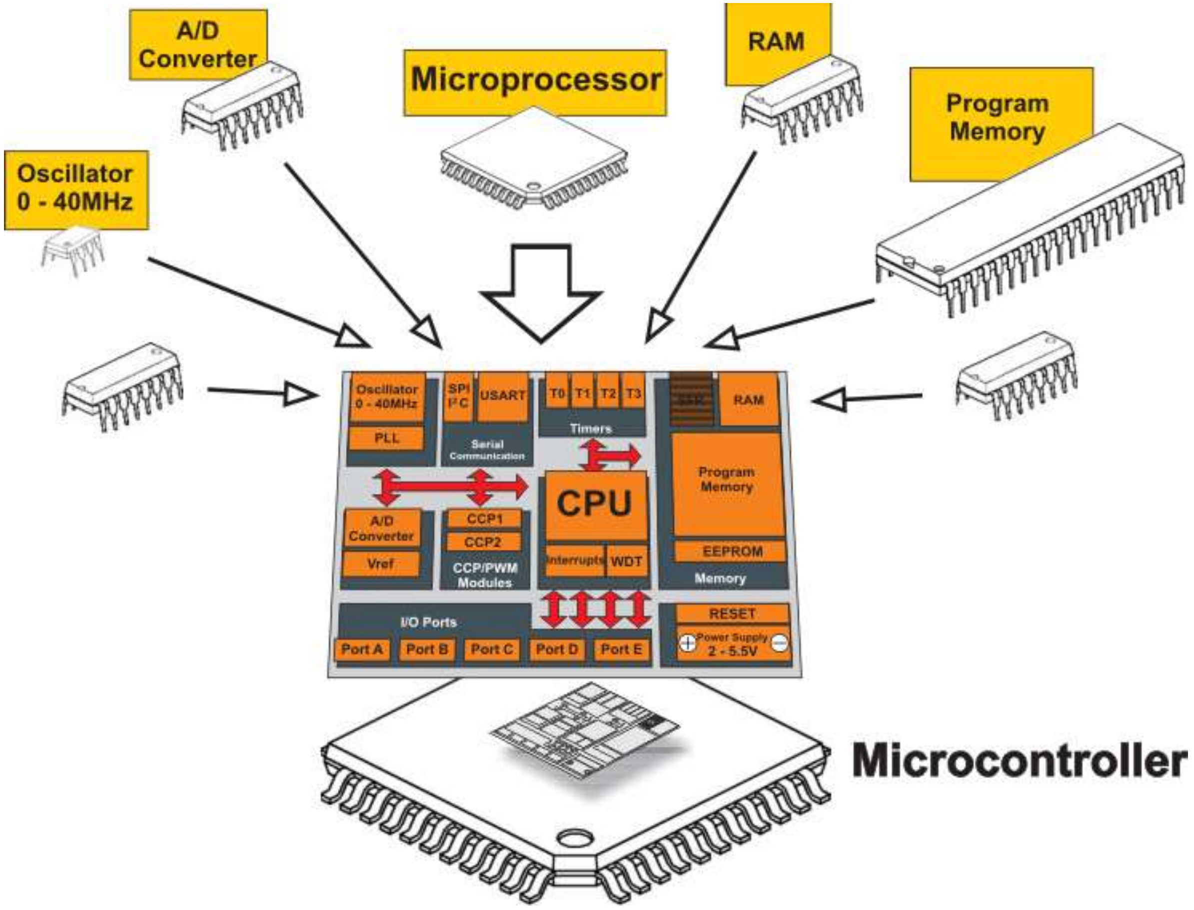
- Some recent flavor of Windows (XP/Vista/7/10)
- IAR Embedded Workbench for ARM
  - Downloaded from IAR website
  - Can use other toolchains*
- Terminal program
  - TeraTermPro is recommended
- PC to run toolchain
  - USB port with good power

Week 1
Introduction

Week 1 – Overview

- Introduction and background topics
  - Introduction to computer hardware
    - Microprocessor vs. Microcontroller
    - Machine organization
    - Processor data path
    - Memory architecture
    - Privilege modes
  - Number systems
    - Number types
    - Endianness
    - Converting between bases
    - Computer math

Microprocessors vs. Microcontrollers



Source: Google Images

Embedded Computer System

- "Embedded" means hidden away
- An embedded computer system:
  - Forms part of a larger system
    - Not generally considered as a computer
  - Performs a dedicated function
    - Does not run general-purpose software
    - Often has a small or no operating system (barebones)
  - Specialized input
    - Limited to sensors
  - Specialized output
    - Limited to actuators, or small displays

Embedded Systems

- > Industrial:
  - – Welders, telescope mount controller, lighting controllers
- > Consumer:
  - – Microwave, washing machines
- > Communications:
  - – Routers, switches, telephones
- > Transportation:
  - – Instrument panels
  - – Seat controllers
  - – “Infotainment system”

Machine Organization

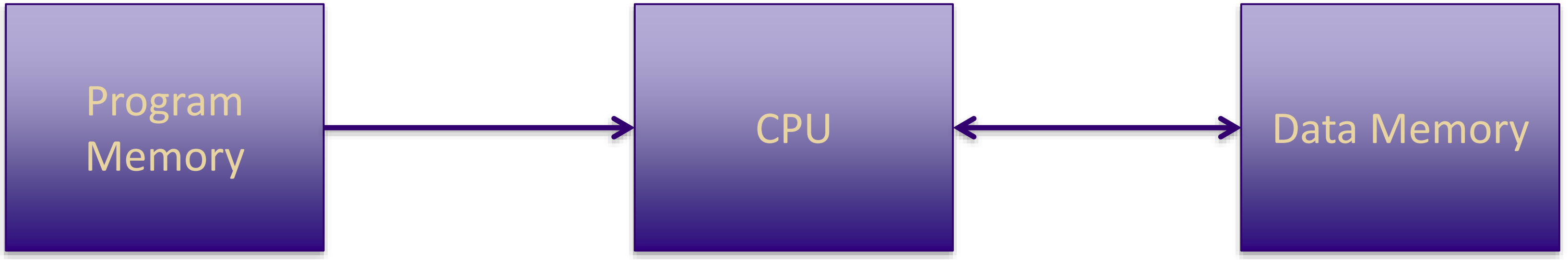
- Processor
  - Control logic controls the processor's guts (decision making)
    - Instructions tell the control logic what to do
  - Data path moves and operates on the data (number crunching)
    - Arithmetic (add, subtract, ect), logical (and, or, xor, ect)
    - Memory (load, store, move)
- Memory
  - Stores machine state
    - Instructions (C program statements and functions)
    - Data (C programs variables)
  - Hierarchical in size and speed
    - On-chip cache
    - On-board ram
    - Hard drive, network storage

Machine Organization – Continued

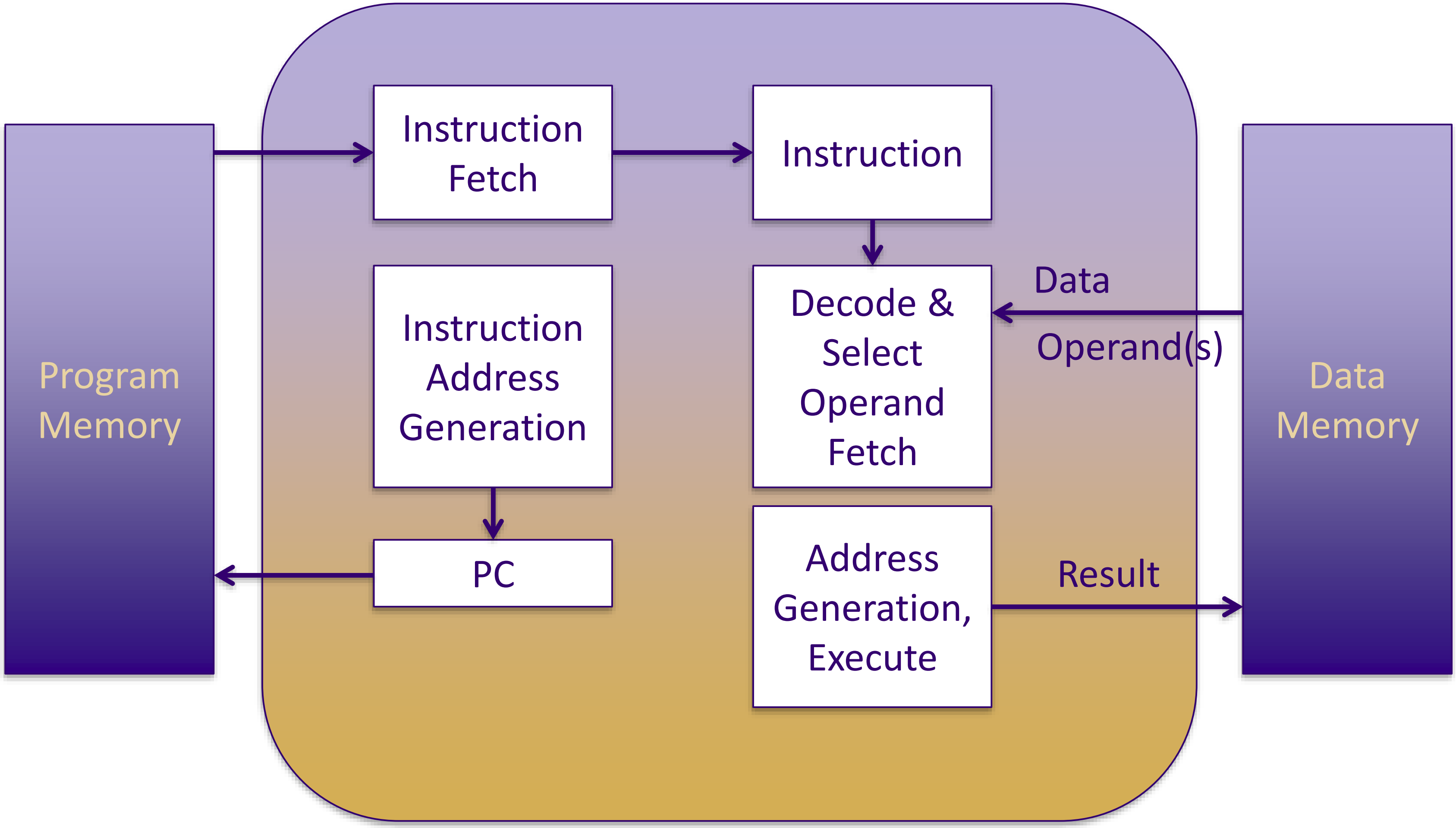
- I/O
  - Computing machine view of the outside world
  - Peripheral device
    - Display, mouse, keyboard, joystick
- Busses connect everything together
  - Simply, just a set of wires that let pieces talk to each other
  - Processor and memory bus
  - Memory and I/O bus
  - I/O bus
    - FireWire, USB, I2C, UART

Processor: Data path

- Instructions tell the control logic how to control the data path
- Data path does a lot of work



Simple CPU Core



Memory Architectures

- > Distinction between program and data memory
- > This is a Harvard architecture proposed by Howard Aiken
  - – Separate busses allow custom optimization
- > The alternate is the Von Neumann architecture proposed by John Von Nuemenn.
  - – Von Nuemann treats instructions and data the same
    - > Instruction fetch and data memory read/writing fight for the same bus.
    - > Memory bandwidth can become a bottleneck

Instruction Set Architectures (ISAs)

- Interface abstraction between hardware and low level software
- Allows code to run on different processors as long as they have the same ISA. (example: x86)
- Can prevent innovation (example: x86)
- Many different ISA's
  - X86, ARM, Thumb, MIPS, ColdFire, 68K, PPC, SHx, Alpha
  - ISA's often have versions
    - ARMv4T – ARM7TDMI
    - ARMv7E – Cortex-M4

Mini Introduction to ARM

- ARM started as a project to replace the 6502
  - Originally meant Acorn RISC Machine
  - In 1990, Acorn Spun off the design team into a new company named Advanced RISC Machines, Ltd.
- ARM Holdings (in 1998) primary business is selling IP cores. This is used to create microcontrollers from various vendors.

Mini Introduction to ARM

- ARM architecture has evolved over time.
  - Three architecture profiles
    - A – Application
      - Raspberry PI
    - R – Real Time
      - Medical devices
      - Robots
      - Avionics
    - M – Microcontroller
      - Embedded systems

Mini Introduction to ARM

- The different Cortex-M cores
  - M7
    - Maximum performance and control and DSP
  - M4
    - Mainstream control and DSP
  - M3
    - Performance and efficiency
  - M0+
    - Highest energy efficiency
  - M0
    - Lowest cost, low power

Mini Introduction to ARM

- Cortex-M cores
  - High performance and efficiency
  - Ease of software development
    - All processors are fully C programmable
  - Superior code density
    - C complier will use 16-bit instructions for code density
    - Unless the operation can be carried out more efficiently using a 32-bit instruction
  - Supports 8-bit, 16-bit, and 32-bit data transfers
- Licensed over 200 ARM partners
  - Vast ecosystem with third-party tools, RTOS and middleware support

Number Systems

Alternate Number Systems

- Humans use base 10.
- Computers use 1’s and 0’s… base 2 or binary
- Binary is cumbersome

Number Systems Summary

Number System	Base/Radix (# of digits)
Decimal	Base 10 (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
Binary	Base 2 (0, 1)
Octal	Base 8 (0, 1, 2, 3, 4, 5, 6, 7)
Hexadecimal	Base 16 (0-9, A, B, C, D, E, F)

Decimal Number System

> All number systems work the same
> Each digit multiplied by the base raised to a power (10^x)
> Decimal example: $65536 = 2^{16}$

$$
\begin{array}{r}
6 * 10^4 = 60000 \\
5 * 10^3 = 5000 \\
5 * 10^2 = 500 \\
3 * 10^1 = 30 \\
\underline{6 * 10^0 = 6} \\
65536
\end{array}
$$

Binary Number System

> Each binary digit multiplied by the base raised to a power (2^x)

> Example: 10101100

$$
\begin{array}{r}
1 * 2^7 = 128 \\
0 * 2^6 = 0 \\
1 * 2^5 = 32 \\
0 * 2^4 = 0 \\
1 * 2^3 = 8 \\
1 * 2^2 = 4 \\
0 * 2^1 = 0 \\
0 * 2^0 = 0 \\
\hline
172
\end{array}
$$

Octal Number System

> Each octal digit multiplied by the base raised to a power (8^x)

> Example: 4275

$4 * 8^3 = 4 * 512 = 2048$

$2 * 8^2 = 2 * 64 = 128$

$7 * 8^1 = 7 * 8 = 56$

$6 * 8^0 = 5 * 1 = 5$

2237

Hexadecimal Number System

- Each hexadecimal digit multiplied by the base raised to a power (16^x)
- Example: 0x123C0DE

$1 * 16^6$ =	1	* 16777216 =	16777216
$2 * 16^5$ =	2	* 1048576 =	2097152
$3 * 16^4$ =	3	* 65536 =	196608
$C * 16^3$ =	12	* 4096 =	49152
$0 * 16^2$ =	0	* 256 =	0
$D * 16^1$ =	13	* 16 =	208
$E * 16^0$ =	14	* 1 =	14
			19120350

Binary and Hexadecimal Unit Relations

- 1 Binary digit = 1 bit
- 4 Bits = 1 Hex digit = 1 Nibble
- 8 Bits = 2 Hex digits = 2 Nibbles = 1 Byte
- 16 Bits = 4 Hex digits = 2 Bytes = 1 Word*
- 32 Bits = 8 Hex digits = 4 Bytes = 1 Long*

- *Sizes > byte are called different things in different ISA's. Beware!

Word Sizes

- ARM and MIPS
  - Half-word = 2 bytes
  - Word = 4 bytes
- ColdFire
  - Word = 2 bytes
  - Long = 4 bytes
- x86
  - Word = 2 bytes
  - Double Word = 4 bytes
- Lessons
  - Be careful
  - Don't assume word size
  - Spills into the programming environment
  - How big are C's ints, shorts, and longs??

Composing Half-Words and Words

- 16-bit half words are made of 2 bytes
  - Most Significant Byte, or MSB
  - Least Significant Byte, or LSB
  - [MSB][LSB]
- 32-bit words are made of 4 bytes
  - [MSB][   ][   ][LSB]

Byte Order

- > 16 and 32 bit numbers are composed of multiple bytes
- > Order of bytes in 16-bit and 32-bit memory blocks is arbitrary
- > Accessing a 16-bit location @ 0:
- > [Byte 0][Byte 1] or [Byte 1][Byte 0]
- > [MSB][LSB]   or [MSB][LSB]

Byte Order – Continued

> Accessing a 32-bit @ 0:
> [Byte 0][Byte 1][Byte 2][Byte 3]
> [MSB][ ][ ][LSB]
> OR
> [Byte 3][Byte 2][Byte 1][Byte 0]
> [MSB][ ][ ][LSB]

Endianness

> Big Endian (Motorola 68K/Coldfire)
> [Byte 0][Byte 1][Byte 2][Byte 3]
> [MSB][ ][ ][LSB]

> Little Endian (Intel x86, ARM)
> [Byte 3][Byte 2][Byte 1][Byte 0]
> [MSB][ ][ ][LSB]

Dealing with Endianness

- Endianness considerations when
  - Multi-byte numbers are used interchangeably between big and little
  - Part of a mulit-byte number is referenced in memory
    - Reference one byte of a 32-bit number in memory
- Compliers have switches to change data models
- When programming in higher level languages, byte order assumption can cause portability bugs.

Converting Binary to Hexadecimal

- One of the easiest conversations
  - Hex is just another representation of binary
- Steps to convert binary to hex:
  - Group binary number in to nibbles
  - Convert individual nibbles to hex notation
  - Change prefix from 0b to 0x or 0X
- Example
  - Group nibbles: 1001 1100 0000 1010
  - Convert nibbles: 9 C 0 A
  - Change the prefix: 0x9C0A

Converting Binary to Decimal

- Done by adding the weighted values of the bits
  - Format: $n = n_3 * 2^3 + n_2 * 2^2 + n_1 * 2^1 + n_0 * 2^0$
- Example: Convert 0b1101 to decimal
  - $n = 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0$
  - $= 8 + 4 + 0 + 1$
  - $= 13$
- Tip:
  - If the LSB = 0, then number is even
  - If the LSB = 1, then the number is odd

Converting From Decimal

Another way…

- Convert decimal number ‘d’ to base ‘b’
  - Find the largest value of ‘b’ raised to an integer power ‘x’ that is smaller than the number ‘d’
  - The digit at place ‘x’ is d / b^x
  - Calculate the remainder $d \% b^x$ and feed this number in as ‘d’ in the second step after decrementing ‘x’
  - Done when you hit $x = 0$

Decimal to Hexadecimal Example

- Convert d = **8363** to hexadecimal (b = 16)
- 16^4 is to big (65536)
- 16^3 can divide 8363 by 2: 0x2---
- 8363 % 16^3 = 171
- 16^2 > 171 so 171 % 256 = 0: 0x20--
- 171 / 16^1 = 10 = 0xA: so now we have: 0x20A-
- 171 % 16^1 = 11
- 11 / 16 = 0: 11 = B: answer = **0x20AB**

Practice Converting Numbers

- You can do this with a calculator…
- BUT, numbers are the vocabulary of the machine
  - Best to be able to do much of the math by hand
    - After a while, you'll be doing it in your head!
  - Using a calculator to help read a hex memory dump could make reading the contents take a long time.

Computer Math

- Binary addition
- Signed numbers
- Addition overflow
- Subtraction
- Multiplication by 2
- Division by 2
- Floating Point math

Binary Addition

> Binary addition is very similar to math in base 10.
  - Add digits in the same place, and carry values greater than the digits range
  - Example: 9 + 2 = 11. We “carried the one”

> In binary
  - As in base 10, we always consider the carry (aka ‘c’ bit)
  - Example: 0 + 0 = 0, c = 0 (no carry)
  - 0 + 1 = 1, c = 0
  - 1 + 0 = 1, c = 0
  - 1 + 1 = 0, c = 1
  - All examples of carry addition

Binary Addition and Subtraction

- Binary addition extends to any word size
  - Add each pair of bits, starting with LSB.
  - The carry ripples forward towards the MSB with each addition.
- Operationally the same as decimal
  - Carry and borrow 1's

Addition

1 1
0000 0101
0000 0011
0000 1000

Subtraction

1 1
0 1 1
0000 ~~1~~000
0000 0011
0000 0101

Signed Numbers

- Examples showed unsigned numbers
- Same groupings can represent signed values
- Most significant bit becomes the “sign” bit
  - Loses one bit in value
- Signed byte
  - Bit 7 becomes sign bit
    - 1 = negative number
    - 0 = positive number
  - Therefore, the range is -128 to 127 (instead of 0 to 255)
  - -128 = 0b1000 0000
  - 127 = 0b0111 1111

Signed Numbers - Continued

- Computers use 2's complement to represent negative numbers
- To form 2's completment of a number:
  - Represent scalar value in binary
  - Invert all bits (1 becomes 0, and 0 becomes 1)
  - Add 1
- Example: Represent -12 as a signed byte
  - Binary: 0000 1100
  - Invert: 1111 0011
  - And add 1: 1111 0100

Signed Numbers – Continued (2)

- Finding the scalar (positive) value of a negative binary number is simular
  - Subtract 1
  - Invert bits
- Example: Find scalar value of 0b1111 1100
  - Subtract 1: 0b1111 1011
  - Invert the bits: 0b0000 0100

Signed Numbers – Continue (3)

- > Binary addition of unsigned numbers can’t change sign
  - 127 + 1 = 0b0111 1111 + 0b0000 0001 = ?
  - Unsigned: 0b0111 1111 + 0b0000 0001 = 0b1000 0000 = 128
- > Binary addition of signed numbers *can* change sign
  - Signed: 0b0111 1111 + 0b0000 0001 = 0b1000 0000 = -128
  - Because 128 exceeds the range of a signed byte, an overflow has occurred, which changes the sign and give an incorrect result
- > The is a danger of writing software
  - Programmer must always be aware of the range of variables

Truncation (Data Loss)

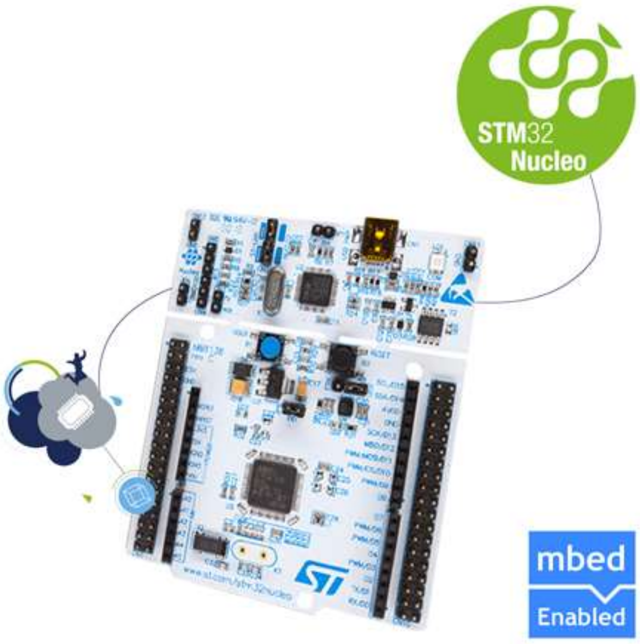
- Truncation
  - The loss of bits of data
  - Occurs when a value is to large for the data type
- Example: Assigning 256 to an unsigned byte
  - Unsigned byte has eight bits
    - Range is 0 to $2^8 - 1 = 255$
    - 256 = 0b1 0000 0000
      - Has to many bits to bit into a byte (9 instead of 8)
      - Exceeds the range of a byte
    - The lower 8 bits are preserved, the rest are lost
    - An unsigned byte of 256, results in 0b0000 0000 = 0
    - In this case, 256 actually equals 0!

The Nucleo Development Kit

UNIVERSITY *of* WASHINGTON

Development Kit Contents

- > Nucleo STM32F401
- > 2.8” TFT Touch Shield
- > “Music Maker” Shield
- > Stacking headers
- > USB 2.0A to Mini-B Cable
- > X-Nucleo-IKS01A1
  - – Maybe?



Development Board Intro

> Used for class programming
> Off the shelf
> Embedded development prototype/hobby board
> Not built like a consumer device
  – Open circuit board
  – Fragile connectors
  – Static sensitive
  – Similar to PC upgrades (video cards)

STM32F401xE Nucleo

- STM32 microcontroller – Cortex M4 with FPU
- On-board ST-Link debugger/programmer
- Three LED’s
  - USB Communication
  - User LED
  - Power LED
- Two push buttons
  - User & Reset
- Expandable extensions
  - Arduino Uno Revision 3 connectivity
  - STM Morpho extension pin headers
- mbed support

STM32F401xE Nucleo

Parts

STM32F103 ST-Link

SWD connector

RESET button

USER button

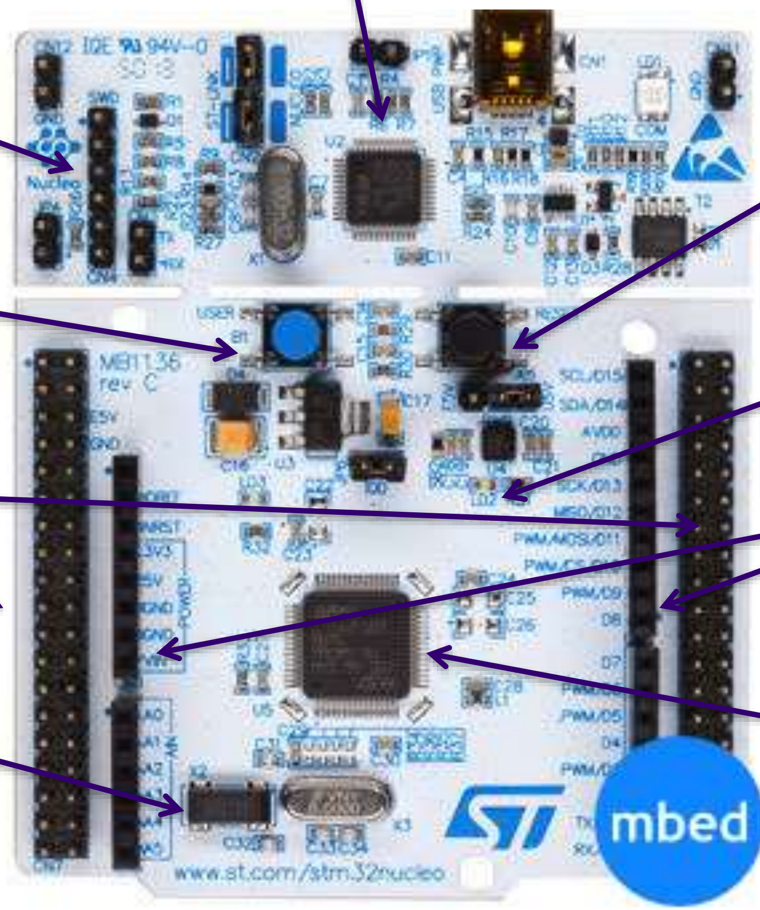
LD2 User LED (Green)

ST Morpho connector

Arduino connector

32Khz crystal (1)

STM32F401



Development Board Handling

- Treat it with care
- Don’t bend it
- Be gentle when connecting cables and pins
- Don’t spill liquids on it
- Don’t let it get too dirty (including dust)
- Keep conductive materials away.
  - Staples, paper clips, ect
- If you use a wrist strap, don’t forget you have it on.

Electrostatic Discharge (ESD)

> Static electricity is destructive to most electronics
> You don't need to feel or hear the zap for damage to occur
> You don't need to touch the component for damage to occur.
  – Just getting close could be enough

Symptoms of ESD Damage

> Everything stops working
> Certain parts stop working
> Part works intermittently
> A lighter wallet…

Preventing ESD Damage

- > Ground yourself (such as a wrist strap)
- > Ground the board
  - – Keep the USB cable connected to the PC
- > Beware of clothes, rugs, and chairs that generate static
- > Keep the board in a anti-static bag or clam shell when stored or transporting
- > Minimize contact with the board

IAR Embedded Workbench for ARM

> Quick IAR guided tour

Week 1 Wrap-up

- Homework
  - Reading assignment
    - PDF: DM00105928_UM1727: Getting started with Nucleo
  - Quiz 1: Math and number conversions
    - Can use a calculator, but I challenge you to try doing them by hand and then confirming your results.
    - Can only be taken once.
  - Discussion 1:
    - Part 1: Introduce yourself. What interests you about embedded systems? Do you work with embedded systems now? What are you hoping to get out of this set of courses?
    - Part 2: Identify at least one embedded system. Describe its dedicated purpose, and it’s interface to the world.

Week 1 Wrap-up

- Homework
  - Install the Development Tools
    - Instructions provided
    - Code skeleton will be provided
      - Compile the code
      - Turn in a screen capture from TeraTerm