CMPE-013/L

Introduction to "C" Programming

Max Lichtenstein

MIPS 32

```
struct {
    * Left child      4
    * R child         4
    * m child         4
    char data         1
} Node
```

sizeof(Node)

Roadmap

- Announcements
- Grades Review
- Battleboats Stuff
  - Rules
  - Field Module
  - Demo
- Timer Question How fast can you observe a timer
- Randomness
- BREAK
- Engineering Tips from Max
- Software Design Principles

Max Lichtenstein

Announcements

- Next week:
  - Still required!
  - What to cover?
  - Current "agenda":
    - C dark arts
    - Tour of Board.c/h and other CMPE13 libraries
    - C vs C++, C#, other low-level languages
    - Hierarchical / Parallel State Machines
    - Events and Services: Queues and Priorities
  - Taking requests!

Magic 8 Quiz

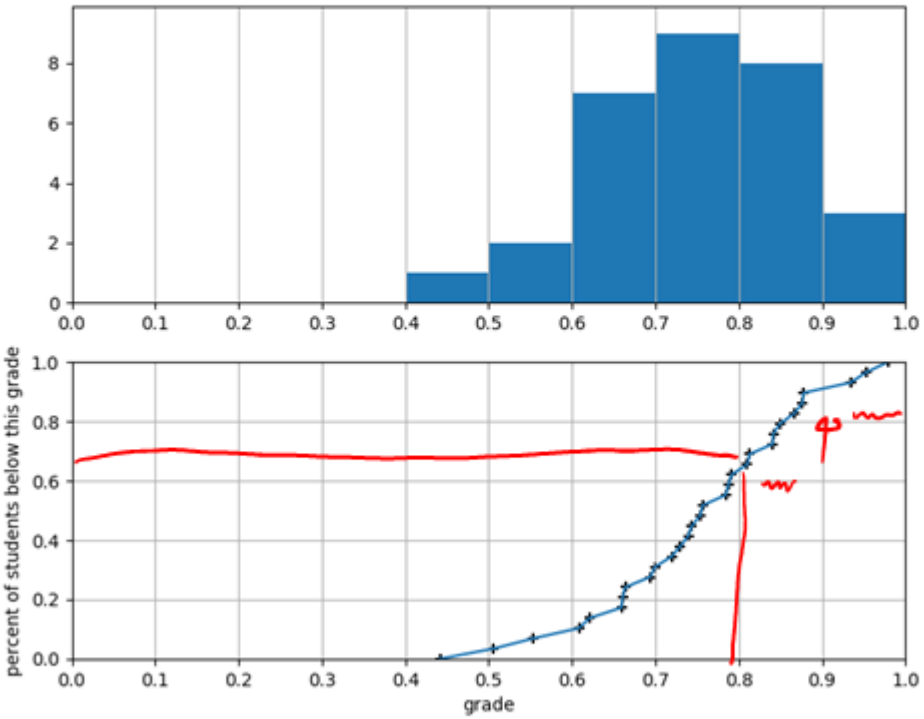
Announcements

- Gitlab still “down”!
  - We have to live with it
  - Try disabling ssl verification in repo config (so much easier!)

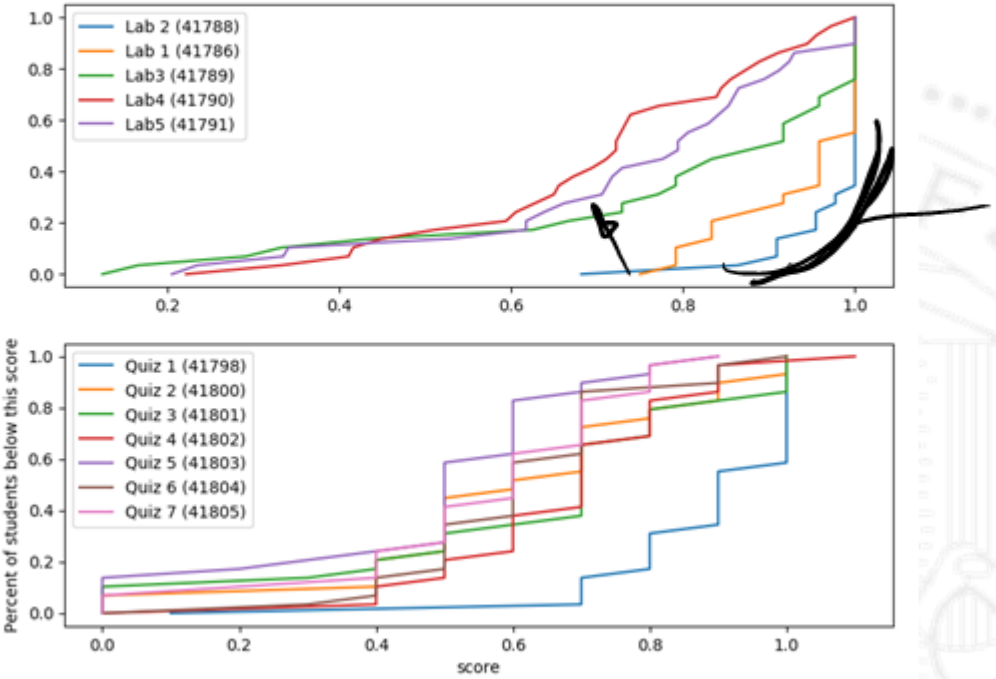
```
git config --global http.sslVerify false
```

Max Lichtenstein

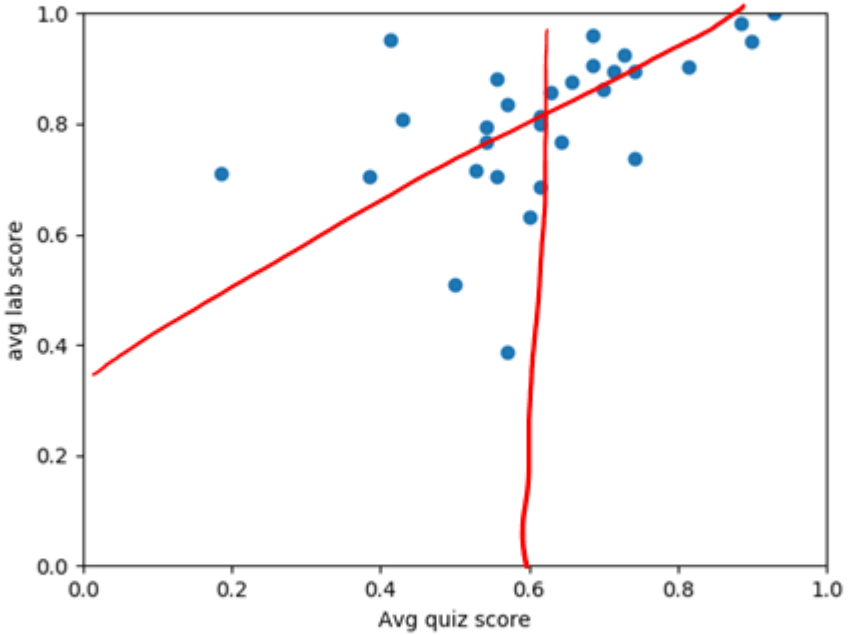
Grade stuff


0
2
4
6
8
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
percent of students below this grade
grade

Grade stuff


Lab 2 (41788)
Lab 1 (41786)
Lab3 (41789)
Lab4 (41790)
Lab5 (41791)
Quiz 1 (41798)
Quiz 2 (41800)
Quiz 3 (41801)
Quiz 4 (41802)
Quiz 5 (41803)
Quiz 6 (41804)
Quiz 7 (41805)
Percent of students below this score
score

Grade stuff



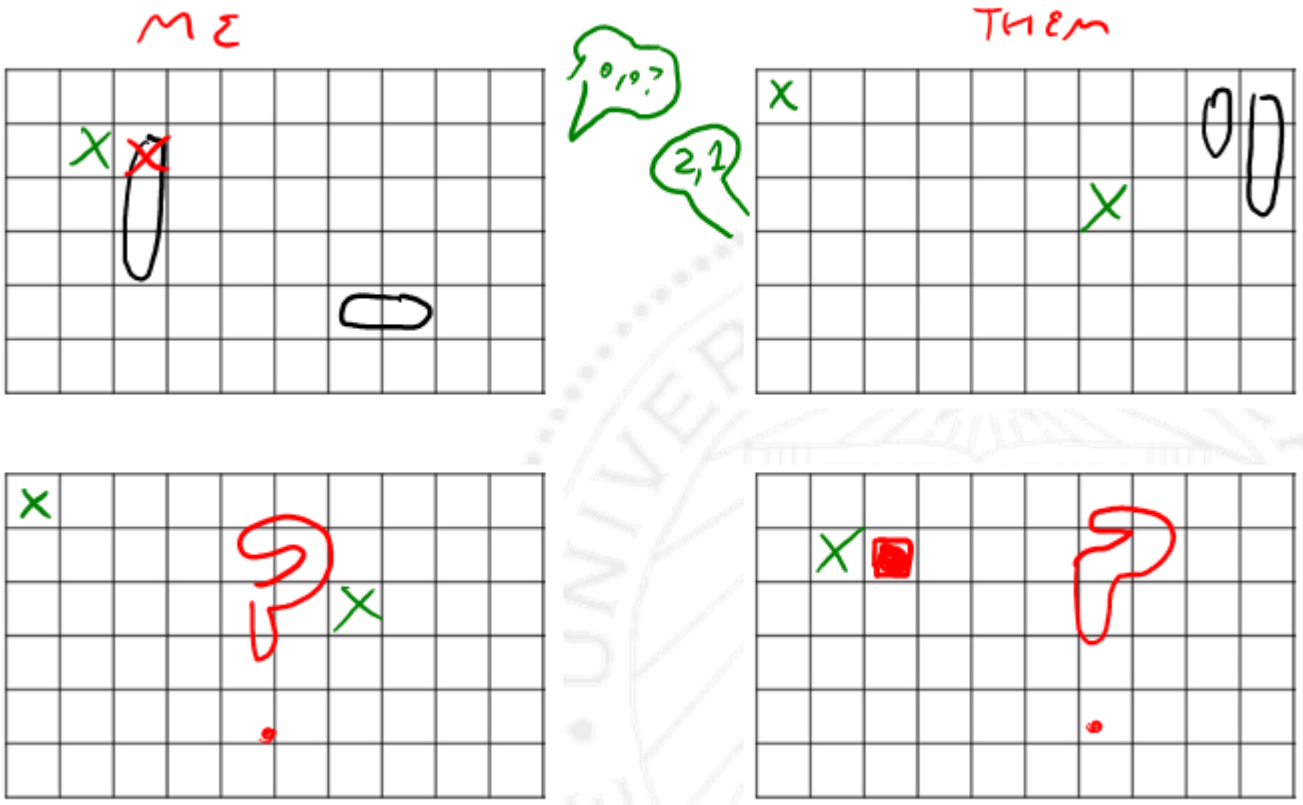
Battleboats Stuff

Battleboats Demo

Battleboats Rules

- A coin flip for first move
- Players place boats
- Each turn:
  - The attacking player makes a guess
  - The defending player describes the result
    - MISS, HIT, SINK a ship
  - BOTH players record the results
- The game is over when one player is out of ships.

Battleboats Rules



Field Module

```
typedef enum {
    /// These denote field pos
    FIELD_SQUARE_EMPTY = 0,
    FIELD_SQUARE_SMALL_BOAT,
    FIELD_SQUARE_MEDIUM_BOAT,
    FIELD_SQUARE_LARGE_BOAT,
    FIELD_SQUARE_HUGE_BOAT,

    /// These denote field pos
    FIELD_SQUARE_UNKNOWN,

    FIELD_SQUARE_HIT,

    ///these statuses may be u
    FIELD_SQUARE_MISS,

    /// This may be useful for
    FIELD_SQUARE_CURSOR,


    /// Occasionally, it may b
    FIELD_SQUARE_INVALID,
} SquareStatus;
```

Max Lichtenstein

Field struct

```
typedef struct {
    uint8_t grid[FIELD_ROWS][FIELD_COLS];
    uint8_t smallBoatLives;
    uint8_t mediumBoatLives;
    uint8_t largeBoatLives;
    uint8_t hugeBoatLives;
} Field;
```

BattleBoats Tips

- Design with visibility in mind

use OLED
Look for tools already there

printf too

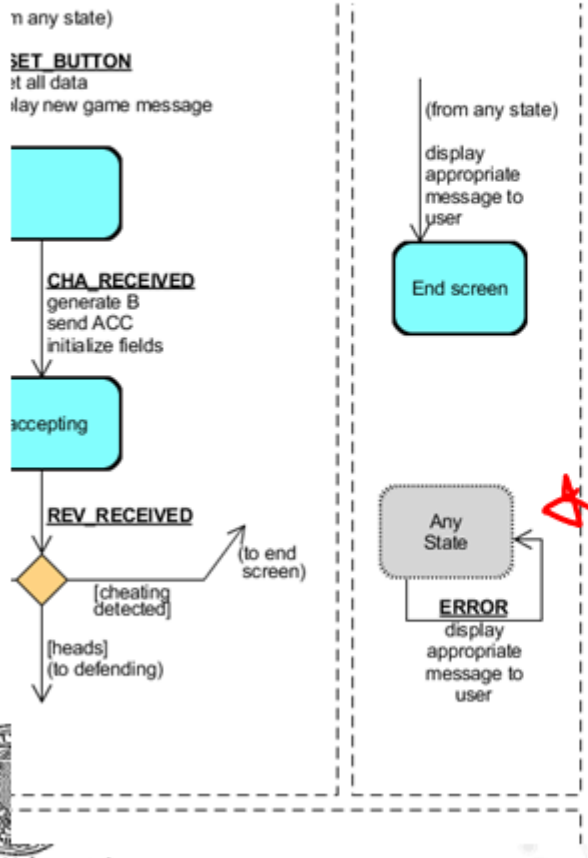
- Use params!

Error Event

//#define printf(...)
#define

```
/**
 * Used to signal different types of errors a
 * of a BattleBoat Error event.  You are not
 * but they can make error checking much more
 *
 */
typedef enum {
    BB_SUCCESS = 0,                          //0
    BB_ERROR_BAD_CHECKSUM,                   //1
    BB_ERROR_PAYLOAD_LEN_EXCEEDED,           //2
    BB_ERROR_CHECKSUM_LEN_EXCEEDED,          //3
    BB_ERROR_INVALID_MESSAGE_TYPE,           //4
    BB_ERROR_MESSAGE_PARSE_FAILURE,
} BB_Error;
```

BattleBoats Announcements



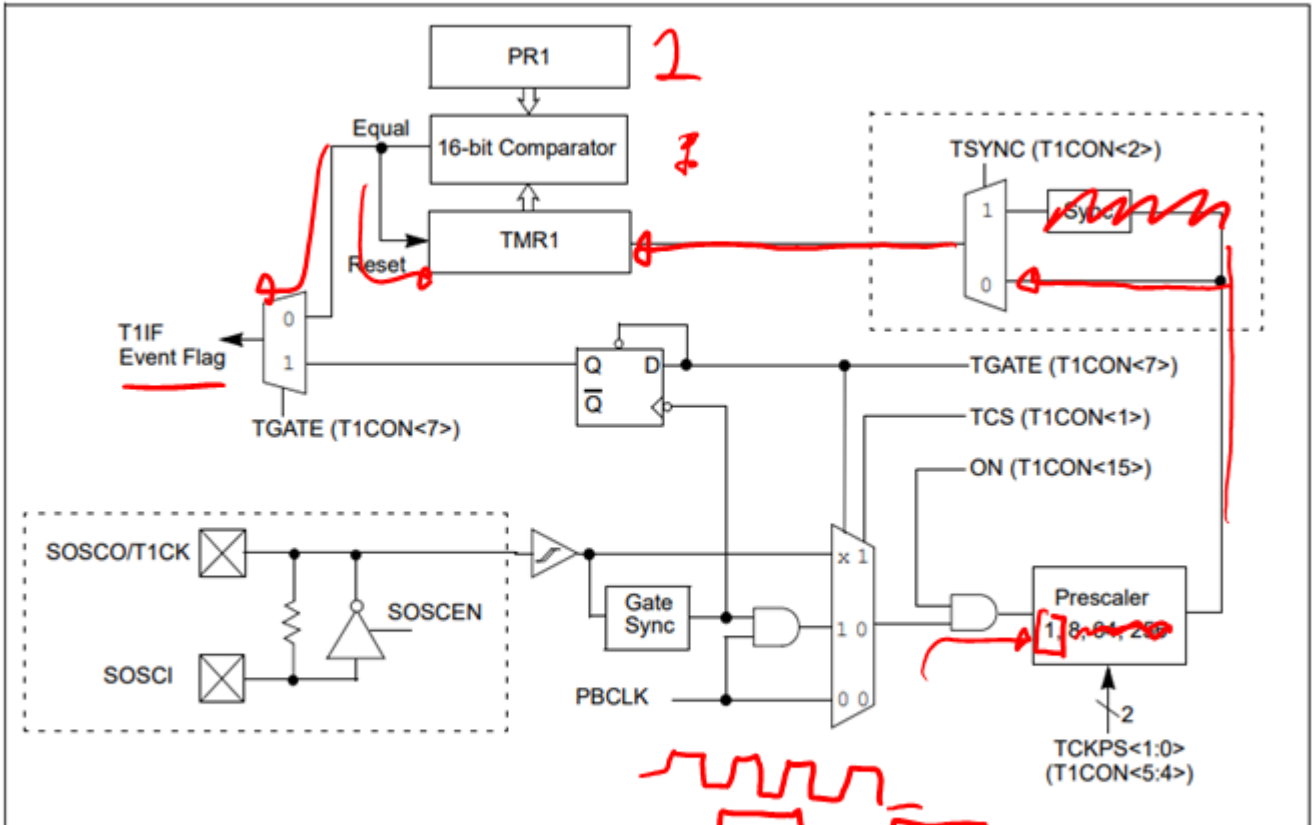
- Small tweak to Lab09 spec:
  - Should help with
  - error investigation

BattleBoats Announcements

- You're *supposed* to work with your partner!

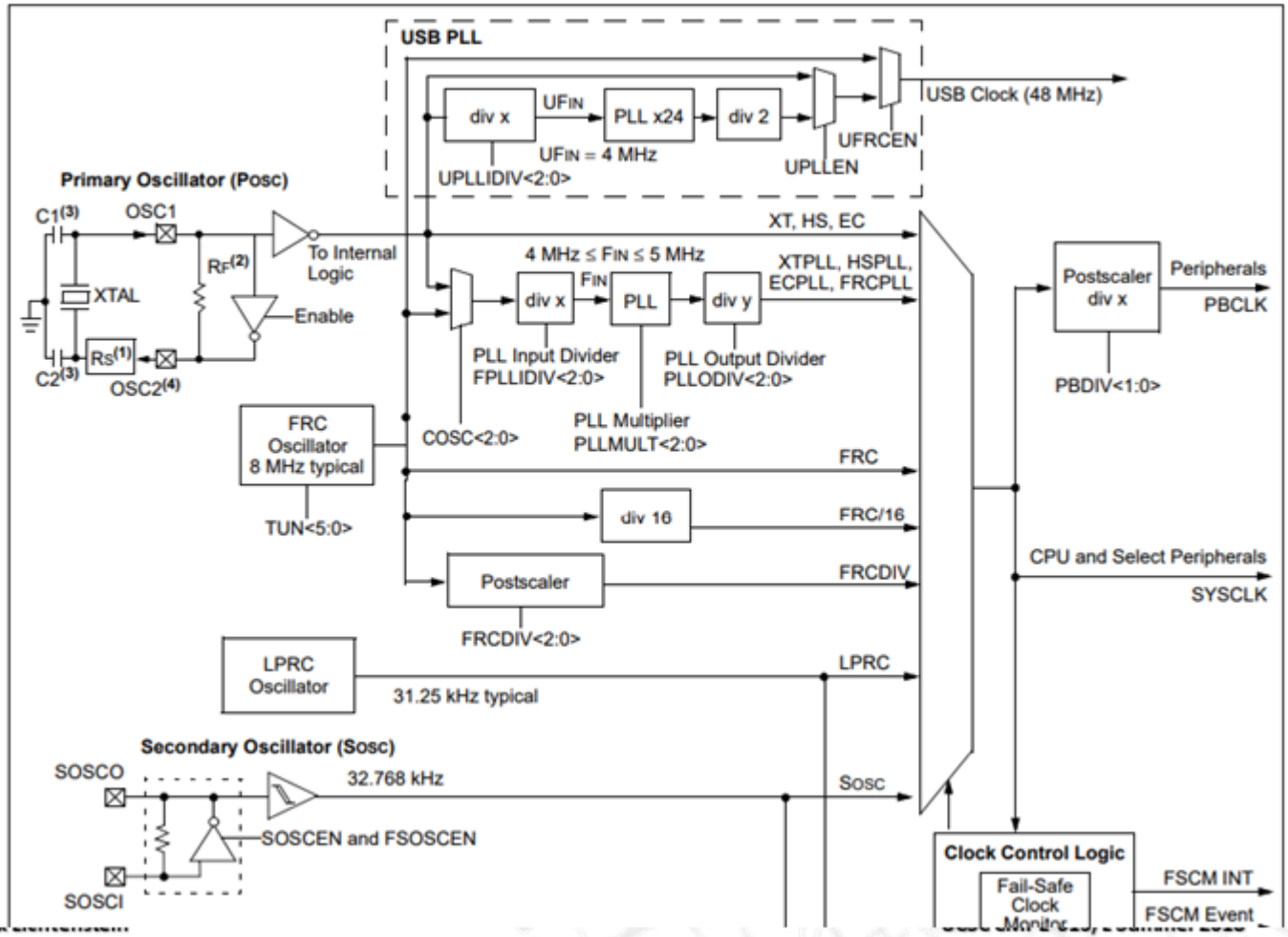
How fast can an ISR go?

FIGURE 13-1: TIMER1 BLOCK DIAGRAM(1)



Note 1: The default state of the SOSCEN (OSCCON<1>) during a device Reset is controlled by the FSOSCEN bit in Configuration Word DEVCFG1.

FIGURE 8-1: PIC32MX3XX/4XX FAMILY CLOCK DIAGRAM



bit 18-16 **PLLMULT<2:0>:** Phase-Locked Loop (PLL) Multiplier bits

The POR default is set by the FPLLMUL<2:0> bits (DEVCFG2<6:4>). Do not change these bits if the PLL is enabled. Refer to the **"Special Features"** chapter in the specific device data sheet for details.

111 = Clock is multiplied by 24
110 = Clock is multiplied by 21
101 = Clock is multiplied by 20
100 = Clock is multiplied by 19
011 = Clock is multiplied by 18
010 = Clock is multiplied by 17
001 = Clock is multiplied by 16
000 = Clock is multiplied by 15

High-Performance 32-bit RISC CPU:

- MIPS32® M4K® 32-bit core with 5-stage pipeline
- 80 MHz maximum frequency
- 1.56 DMIPS/MHz (Dhrystone 2.1) performance at 0 wait state Flash access
- Single-cycle multiply and high-performance divide unit
- MIPS16e® mode for up to 40% smaller code size
- Two sets of 32 core register files (32-bit) to reduce interrupt latency
- Prefetch Cache module to speed execution from Flash

```
void __ISR(_TIMER_1_VECTOR, IPL4AUTO) Timer1Handler(void) {
    // Clear the interrupt flag.
    INTClearFlag(INT_T1);

    // If we've exceeded the timer trigger count, trigger a timer event.
    if (++timerData.value > SWITCH_STATES()) {
        timerData.event = true;
        timerData.value = 0;
    }
}
```

What happens when an interrupt occurs?

```
!void __ISR(_TIMER_1_VECTOR, IPL4AUTO) Timer1Handler(void) {
0x9D0026F8: RDPGPR SP, SP
0x9D0026FC: MFC0 K1, EPC
0x9D002700: MFC0 K0, SRSCtl
0x9D002704: ADDIU SP, SP, -120
0x9D002708: SW K1, 116(SP)
0x9D00270C: MFC0 K1, Status
0x9D002710: SW K0, 108(SP)
0x9D002714: SW K1, 112(SP)
0x9D002718: INS K1, ZERO, 1, 15
0x9D00271C: ORI K1, K1, 4096
0x9D002720: MTC0 K1, Status
0x9D002724: SW V1, 28(SP)
0x9D002728: SW V0, 24(SP)
0x9D00272C: LW V1, 108(SP)
0x9D002730: ANDI V1, V1, 15
0x9D002734: BNE V1, ZERO, 0x9D002780
0x9D002738: NOP
0x9D00273C: SW RA, 92(SP)
0x9D002740: SW S8, 88(SP)
0x9D002744: SW T9, 84(SP)
0x9D002748: SW T8, 80(SP)
0x9D00274C: SW T7, 76(SP)
0x9D002750: SW T6, 72(SP)
0x9D002754: SW T5, 68(SP)
0x9D002758: SW T4, 64(SP)
```

```
0x9D002774: SW A1, 36(SP)
0x9D002778: SW A0, 32(SP)
0x9D00277C: SW AT, 20(SP)
0x9D002780: NOP
0x9D002784: MFLO V0
0x9D002788: SW V0, 100(SP)
0x9D00278C: MFHI V1
0x9D002790: SW V1, 96(SP)
0x9D002794: ADDU S8, SP, ZERO
!    // Clear the interrupt flag.
!    INTClearFlag(INT_T1);
0x9D002798: ADDIU A0, ZERO, 8
0x9D00279C: JAL INTClearFlag
0x9D0027A0: NOP
```

```
$ git commit
$ git tag Lab08_submission_4
$ git push --tag
```

Randomness

rand();

0 6 9 1 2 0 0 1 9 ...

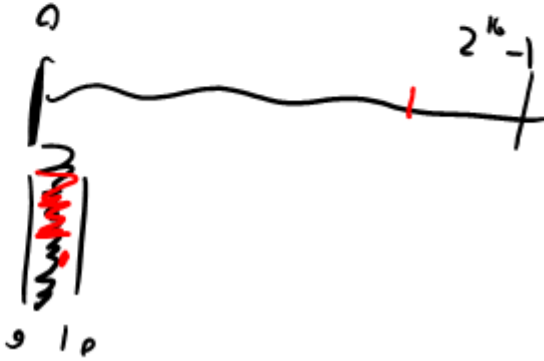
uint16_t ∈ [0 .... 65535]

9

~~rand() >> 12~~

```
int x;
do {
    x = rand() >> 12;
} while (x > 10);
```

rand() % 10



rand() between 0 and 100,000

(16 bits) (17 bits)

16 | 0000 ... 0000 | 1011010111001000

16 | 1010110001011 | 0000 ... 000

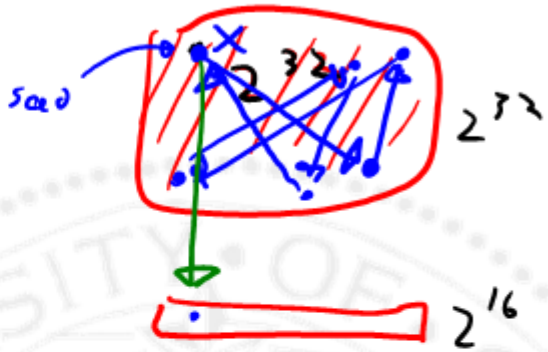
```
uint32_x = rand();
x &= rand()<<16;
x %= 100 000;
```

Randomness

- How to do randomness on a deterministic machine?

Pseudo-randomness

- Start with a big, static variable
- Seed it
- Hash it
  - Usually, a linear Congruential Generator:



close enough for jazz

$$X_{n+1} = (aX_n + c) \mod m$$

const const const

× / 0xBEEF

Max Lichtenstein

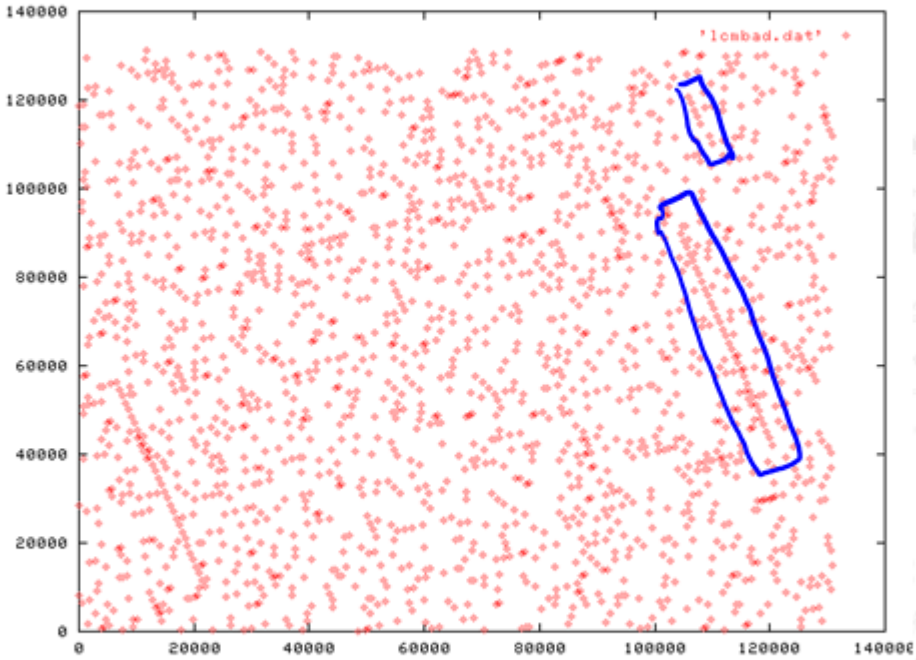
Pseudo-randomness Issues

- It's a cycle!
  - Has a maximum period
  - If eavesdroppers can watch long enough, they can figure out where you are in your cycle
  - it can't shuffle a list very randomly
- Correlations still appear (usually)
  - DieHard battery

100! > 2^{32}

Max Lichtenstein

odd Pseudo-randomness



even

– From https://www.taygeta.com/rwalks/node1.html

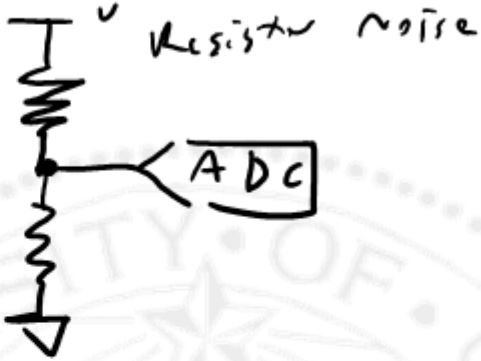
Max Lichtenstein

Real* randomness

- Use (nearly) non-deterministic phenomena
  - Noise
  - Human interaction
  - Timer drift

Randomness from noise

Chaotic



external sensors

Max Lichtenstein

Randomness from humans

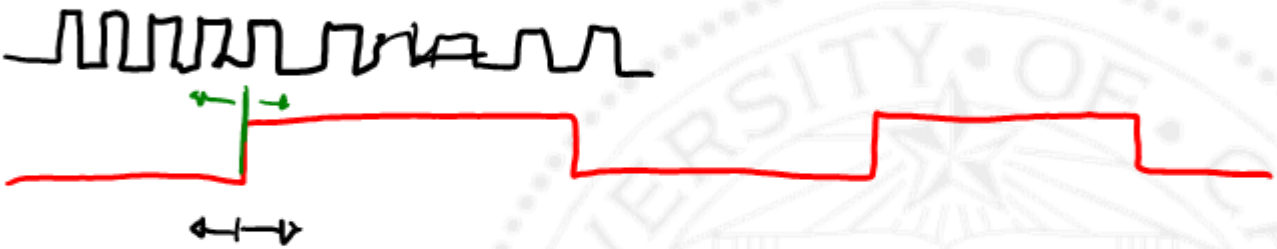
take the time

Key strokes

Max Lichtenstein

Randomness from clock drift

2 clocks



ISR (big timer)

read little timer

Combining randomness

- XORing two random bitstrings
  - ALWAYS increases your entropy
  - Can combine many of these

A	B	OR	AND	XOR
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

```
main(){
    BOARD_Init();
    srand(SWITCH_STATES());
```

BREAK

Engineering Tips from Max

- Naming is important
  - That's why it's hard

top_level_event;

- Functions should do one thing well

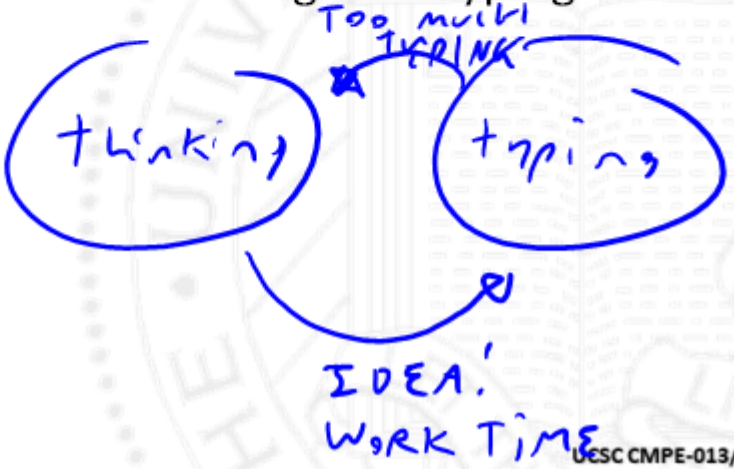
Message Parse
- Validate checksum
- Make message struct

Engineering Tips from Max

- Take breaks!

- sleep
- eat
- shower
- go outside

- Coding is more about thinking than typing



Engineering Tips from Max

- Work on teams where you're the best, work on teams where you're the worst

- Learn new languages

Engineering Tips from Max

- Design systems with debugging in mind

visibility

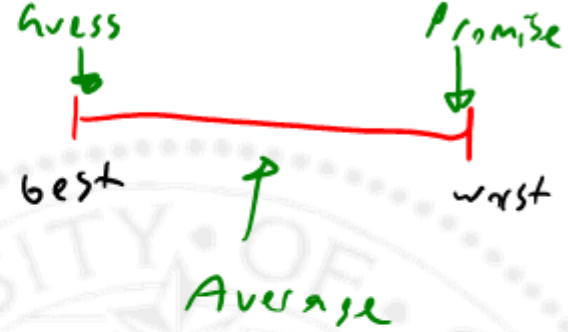
- ~~Learn new languages~~

Max Lichtenstein

Engineering Tips from Max

- Underpromise and overdeliver
  - Multiply by 10

- Do it for love, not money



Engineering Tips from Max

- Help fix the culture
  - Sexist, racist MAYBE
  - Challenge your own biases
  - think about users
  - Engineers are full of themselves

Max Lichtenstein

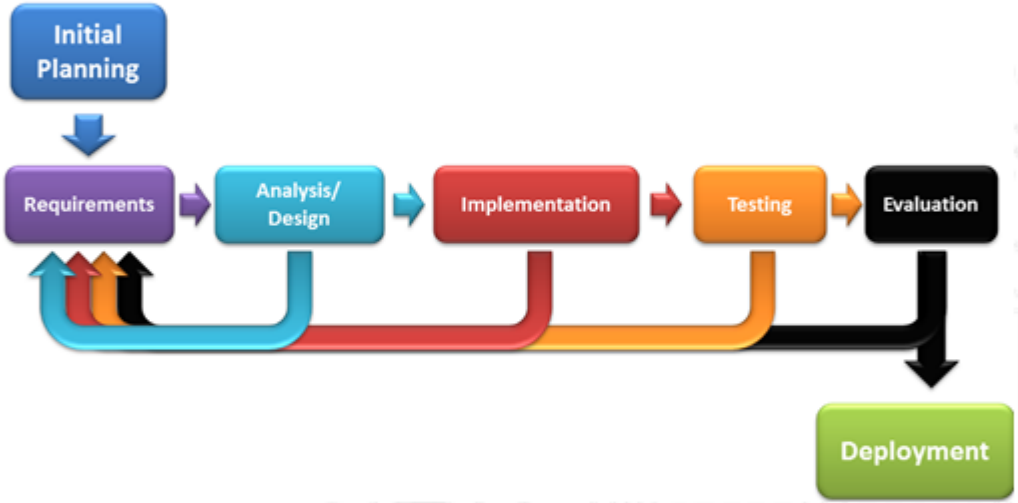
Software Engineering

Design

Build

Software Engineering

Design process



Software Engineering

Principles

- Use consistent styling
- Summary:
  - Utilize whitespace
  - Good variable/function names
  - Comments that describe non-obvious code behavior
    - "How?" and "why?" are good questions to answer in comments

Software Engineering

Principles

- Modularity is important
- Why?
  - Supports code reuse
  - Simplifies changes
  - Allows for testing
- How?
  - Keep functions small
  - Minimize side effects
  - Information hiding/encapsulation

Software Engineering

Principles

- Information hiding/encapsulation
- Summary:
  - Hide unimportant details from the user
  - Protects the user from breaking things
  - Separates backend from frontend

Software Engineering

Mantras

- Keep it simple, stupid
  - KISS
- Summary:
  - Don't solve problems you don't need to
  - Don't introduce unnecessary complexity
  - Prioritize for readability and modularity
  - Don't be clever and/or cute
  - Applies to code architecture and specific code constructs

Software Engineering

Mantras

- Don't repeat yourself
  - DRY
- Summary:
  - Write code only once
  - Simplifies refactoring/incremental development
  - Avoids copy/paste errors

Software Engineering

Mantras

- You aren't gonna need it
  - YAGNI
- Summary:
  - Don't introduce features that are unnecessary
  - Don't write more code then you have to
  - Start small and build from there

Software Engineering

Principles

- Principle of Least Astonishment
- Summary:
  - Be consistent with user's expectations
  - Build on user's intuition
  - Applies to users and developers
    - so both the code and library/program functionality
  - Lowers learning curve

Software Engineering

Principle of Least Astonishment

- Functions/variables should have clear names
  - That should match their functionality!
  - Same for comments
- Functions should not do more than you would think
  - Minimize side effects
- Code should be grouped logically
- Functionality should follow precedence if any exists

Software Engineering

Principles

- Garbage in, garbage out
- Summary:
  - "A system's output quality usually cannot be better than the input quality"
  - So bad input results in garbage output
    - Instead of an error condition
  - Can propagate through the system
  - Can be mitigated by checking the input data

Software Engineering

Principles

- Fault tolerant design
- Summary:
  - Plan for operating failures
    - Running out of memory
    - Data being corrupted
  - Provide fallback modes
  - Important for complex software where minor errors can be common
  - Part of defensive programming

Software Engineering

Principles

- Error tolerant design
- Summary:
  - Plan for user errors
    - "Fault tolerant design" applied to the human component
  - Primarily invalid user input
  - Important for complex software where minor errors can be common
  - Part of defensive programming

Software Engineering

Writing fault/error tolerant code

- Check return values for errors!
  - Many functions have special return values when there are errors, these should usually be checked
  - File accesses
  - scanf()
  - malloc()
- Your code should have special error values
  - LinkedList library
- Program should also return error if failure

Software Engineering

Principles

- Eating your own dogfood
- Summary:
  - When engineers use their own creations, they're generally better
  - More likely that bugs are fixed, features are added because they directly impact the developers
  - In use by all of industry
  - I do it

Software Engineering

Pitfalls

- Premature Optimization
  - "root of all evil"
- Summary:
  - Optimizing code before performance is a critical factor
  - Optimizing reduces readability & modularity
  - Optimization not required for a lot of code
    - See Amdahl's Law
  - See KISS

Software Engineering

Teamwork

- Working as a group is **the** most challenging engineering practice
- Requires:
  - Good communication
- That's it!

Software Engineering

Teamwork

- Pair programming
- Summary:
  - Two developers work side by side: one driving, the other navigating
  - Just like driving:
    - Driver writes code
    - Navigator plans ahead, thinks of edge cases, double-checks driver
  - Requires frequent role switching to be effective!

Software Engineering

Teamwork

- Division of labor
- Summary:
  - Divide work into tasks that can be split between team members
  - Requires coordination to not step on each other's toes
  - Documentation is very important!
  - Can be useful to split testing and development between different people