

x86-64 Programming IV

CSE 351 Autumn 2023

Instructor:

Justin Hsia

Teaching Assistants:

Afifah Kashif

Malak Zaki

Bhavik Soni

Naama Amiel

Cassandra Lam

Nayha Auradkar

Connie Chen

Nikolas McNamee

David Dai

Pedro Amarante

Dawit Hailu

Renee Ruan

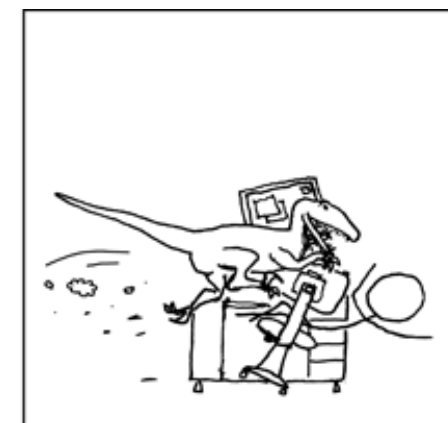
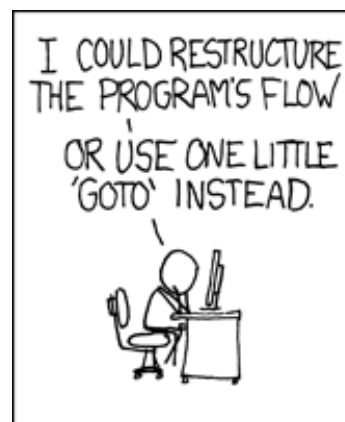
Ellis Haker

Simran Bagaria

Eyoel Gebre

Will Robertson

Joshua Tan



<http://xkcd.com/571/>

Relevant Course Information

- ❖ Lab 1a regrade requests open on Gradescope
- ❖ Lab 1b submissions close tonight
- ❖ Lab 2 due next Friday (10/27)
 - Section tomorrow on to help prep you for Lab 2 – use the midterm reference sheet & bring your laptop!
 - Optional GDB Tutorial in Ed Lessons
- ❖ Midterm (take home, 11/2–11/4)
 - Make notes and use the [midterm reference sheet](#)
 - Form study groups and look at past exams!

A detailed, colorful micrograph of a microchip die, showing a complex grid of circuitry and various colored regions. The text "x86-64 Programming IV" is overlaid in the center.

x86-64 Programming IV

Lesson Summary (1/2)

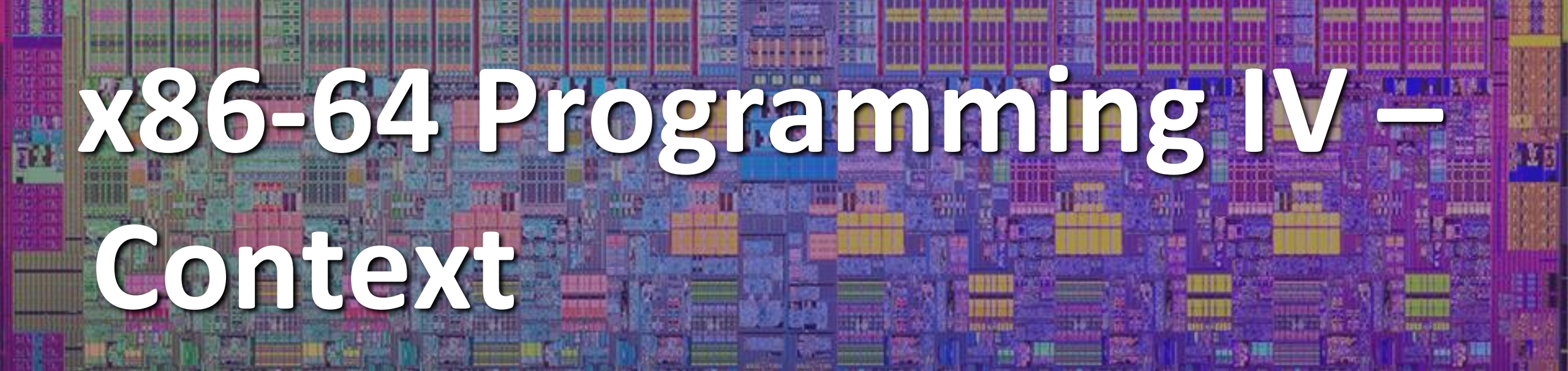
- ❖ Most control flow constructs (*e.g.*, if-else, for-loop, while-loop) can be implemented in assembly using combinations of conditional and unconditional jumps
 - Differences come from placement of jumps and whether they jump forward or backwards in code
- ❖ Switch statements can be implemented using jump tables and indirect jump instructions
 - Jump tables are arrays of pointers to code blocks
 - Indirect jump jumps to address stored somewhere in memory instead of target specified in instruction

Lesson Summary (2/2)

- ❖ Terminology:
 - Jump table, indirect jump

- ❖ Learning Objectives:
 - Without executing, describe the overall purpose of snippets of x86-64 assembly code containing arithmetic, if-else statements, and/or loops.

- ❖ What lingering questions do you have from the lesson?

A detailed, colorful micrograph of a microchip die, showing intricate circuit patterns in shades of purple, blue, yellow, and green. The text is overlaid on this background.

x86-64 Programming IV – Context

Labels & Jumps in C (goto)

- ❖ C allows goto as means of transferring control (jump)
 - Closer to assembly programming style
 - Generally considered bad coding style

```
long absdiff(long x, long y) {
    long result;
    if (x > y)
        result = x-y;
    else
        result = y-x;
    return result;
}
```

```
long absdiff_j(long x, long y) {
    long result;
    int ntest = (x <= y);
    if (ntest) goto Else;
    result = x-y;
    goto Done;
Else:
    result = y-x;
Done:
    return result;
}
```

Labels & Jumps in C (goto)

- ❖ C allows goto as means of transferring control (jump)
 - Closer to assembly programming style
 - Generally considered bad coding style... listen to Kernighan & Ritchie:

3.8 Goto and Labels

C provides the infinitely-abusable `goto` statement, and labels to branch to. Formally, the `goto` is never necessary, and in practice it is almost always easy to write code without it. We have not used `goto` in this book.

Nevertheless, there are a few situations where `gotos` may find a place. The most common is to abandon processing in some deeply nested structure, such as breaking out of two or more loops at once. The `break` statement cannot be used directly since it only exits from the innermost loop. Thus:

Mainstream ISAs, Revisited



x86

Designer	Intel, AMD
Bits	16-bit, 32-bit and 64-bit
Introduced	1978 (16-bit), 1985 (32-bit), 2003 (64-bit)
Design	CISC
Type	Register-memory
Encoding	Variable (1 to 15 bytes)
Branching	Condition code
Endianness	Little

Macbooks & PCs
(Core i3, i5, i7, M)
[x86-64 Instruction Set](#)



ARM

Designer	Arm Holdings
Bits	32-bit, 64-bit
Introduced	1985
Design	RISC
Type	Register-Register
Encoding	AArch64/A64 and AArch32/A32 use 32-bit instructions, T32 (Thumb-2) uses mixed 16- and 32-bit instructions; ARMv7 user-space compatibility. ^[1]
Branching	Condition code, compare and branch
Endianness	Bi (little as default)

Smartphone-like devices
(iPhone, iPad, Raspberry Pi)
[ARM Instruction Set](#)



RISC-V

Designer	University of California, Berkeley
Bits	32 · 64 · 128
Introduced	2010
Design	RISC
Type	Load-store
Encoding	Variable
Endianness	Little ^{[1][3]}

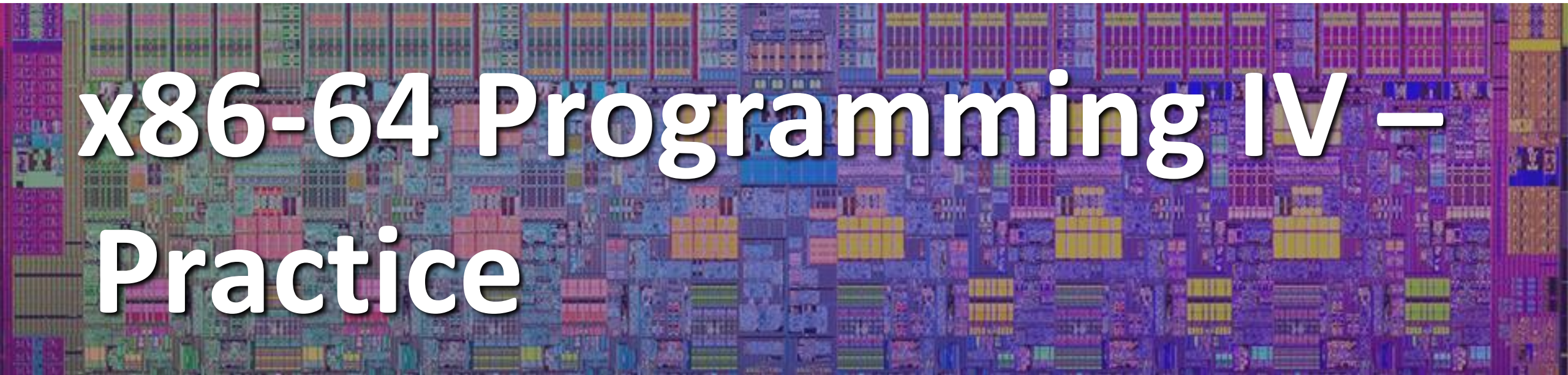
Mostly research
(some traction in embedded)
[RISC-V Instruction Set](#)

Discussion Question

- ❖ Discuss the following question(s) in groups of 3-4 students
 - I will call on a few groups afterwards so please be prepared to share out
 - Be respectful of others' opinions and experiences

- ❖ We taught you assembly using x86-64; you didn't have a choice
 - What are some of the advantages and drawbacks of this choice?

 - What are some possible assumptions we are making about our students or values we are forcing on our students with this choice?

A detailed, colorful micrograph of a microchip die, showing intricate circuit patterns in shades of purple, blue, yellow, and red. The text is overlaid on this background.

x86-64 Programming IV – Practice

Group Work Time

- ❖ During this time, you are encouraged to work on the following:
 - 1) If desired, continue your discussion
 - 2) Work on the lesson problems (solutions at the end of class)
 - 3) Work on the homework problems

- ❖ Resources:
 - You can revisit the lesson material
 - Work together in groups and help each other out
 - Course staff will circle around to provide support

Practice Question

- ❖ The following is assembly code for a for-loop; identify the corresponding parts (Init, Test, Update)
 - $i \rightarrow \%eax$, $x \rightarrow \%rdi$, $y \rightarrow \%esi$

```
Line
 1      movl    $0, %eax
 2      .L2:  cmpl    %esi, %eax
 3              jge     .L4
 4      movslq  %eax, %rdx
 5      leaq   (%rdi,%rdx,4), %rcx
 6      movl   (%rcx), %edx
 7      addl   $1, %edx
 8      movl   %edx, (%rcx)
 9      addl   $1, %eax
10      jmp    .L2
11      .L4:
```

for(_____ ; _____ ; _____)