x86-64 Programming III

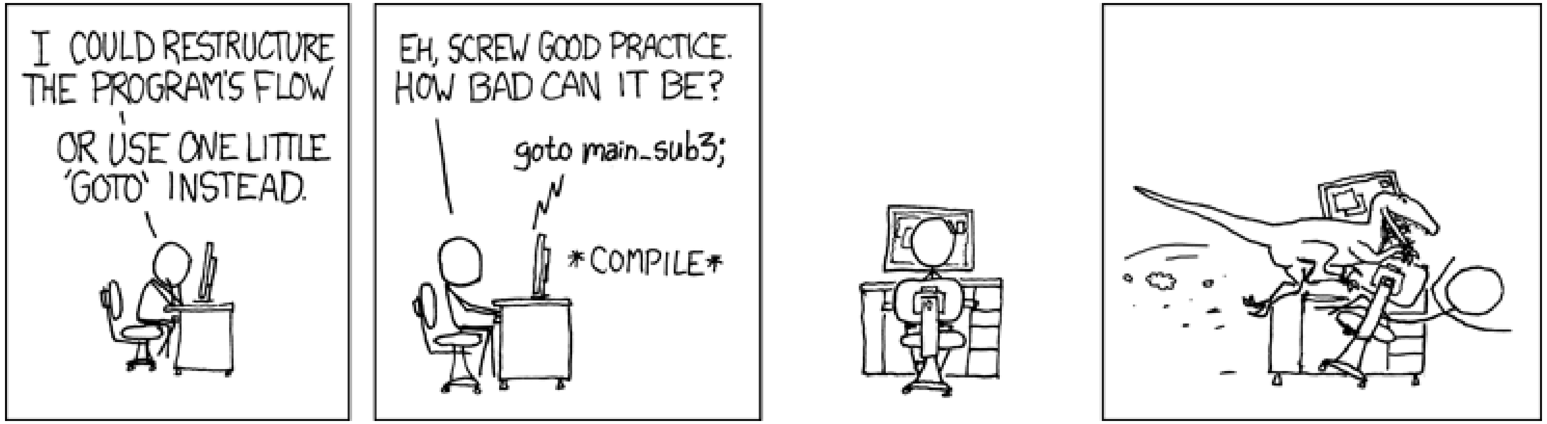
CSE 351 Winter 2024

Instructor:

Justin Hsia

Teaching Assistants:

Adithi Raghavan
Aman Mohammed
Connie Chen
Eyoel Gebre
Jiawei Huang
Malak Zaki
Naama Amiel
Nathan Khuat
Nikolas McNamee
Pedro Amarante
Will Robertson


I COULD RESTRUCTURE THE PROGRAM'S FLOW
OR USE ONE LITTLE 'GOTO' INSTEAD.
EH, SCREW GOOD PRACTICE. HOW BAD CAN IT BE?
goto main_sub3;
COMPILE

http://xkcd.com/571/

Relevant Course Information

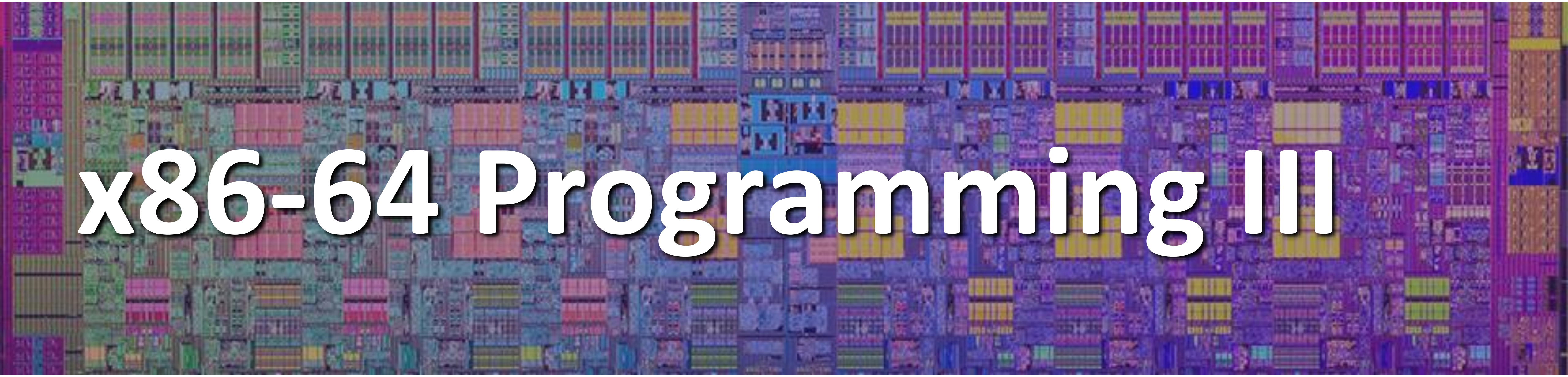
- Lab 1a regrade requests open on Gradescope
- Lab 1b submissions close tonight
- Lab 2 due next Friday (2/2)
  - Section tomorrow on to help prep you for Lab 2 – use the midterm reference sheet & bring your laptop!
  - Optional GDB Tutorial in Ed Lessons

- Midterm (take home, 2/8–2/10)
  - Make notes and use the midterm reference sheet
  - Form study groups and look at past exams!

Extra Credit

- All labs starting with Lab 2 have extra credit portions
  - These are meant to be fun extensions to the labs

- Extra credit points *don't* affect your lab grades
  - From the course policies: "they will be accumulated over the course and will be used to bump up borderline grades at the end of the quarter."
  - Make sure you finish the rest of the lab before attempting any extra credit



x86-64 Programming III

Lesson Summary (1/3)

- **Labels** (*e.g.*, `main`, `.L0`) refer to an instruction address and used as jump targets in assembly

- Control flow in x86 determined by Condition Codes
  - **Set instructions** (`set*`) read out flag values (0/1)
  - **Jump instructions** (`j*`) use flag values to determine next instruction to execute
  - Result of 1st instruction gets compared against 0 in a way determined by 2nd instruction:

			`(op) s, d`	`cmp a, b`	`test a, b`
`je`	`sete`	"Equal"	`d (op) s == 0`	`b-a == 0`	`b&a == 0`
`jne`	`setne`	"Not equal"	`d (op) s != 0`	`b-a != 0`	`b&a != 0`
`js`	`sets`	"Signed" (negative)	`d (op) s < 0`	`b-a < 0`	`b&a < 0`
`jns`	`setns`	"Not signed" (nonnegative)	`d (op) s >= 0`	`b-a >= 0`	`b&a >= 0`
`jg`	`setg`	"Greater"	`d (op) s > 0`	`b-a > 0`	`b&a > 0`
`jge`	`setge`	"Greater or equal"	`d (op) s >= 0`	`b-a >= 0`	`b&a >= 0`
`jl`	`setl`	"Less"	`d (op) s < 0`	`b-a < 0`	`b&a < 0`
`jle`	`setle`	"Less or equal"	`d (op) s <= 0`	`b-a < 0`	`b&a <= 0`
`ja`	`seta`	"Above" (unsigned >)	`d (op) s > 0U`	$b >_U a$	`b&a > 0U`
`jb`	`setb`	"Below" (unsigned <)	`d (op) s < 0U`	$b <_U a$	`b&a < 0U`

Lesson Summary (2/3)

- 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

```
if (<test>)
  <then>
else
  <else>
```

```
     <!test>
     j*' .L2
     <then>
     jmp done
.L2:
     <else>
done:
```

```
     <test>
     j* .L2
     <else>
     jmp done
.L2:
     <then>
done:
```

```
do {
    <body>
} while (<test>)
```

```
top:   <body>
       <test>
       j* top
done:
```

```
while (<test>) {
    <body>
}
```

```
top:   <!test>
       j*' done
       <body>
       jmp top
done:
```

```
       <!test>
       j*' done
top:   <body>
       <test>
       j* top
done:
```

```
<init>
while (<test>) {
    <loop body>
    <update>
}
```

Lesson Summary (3/3)

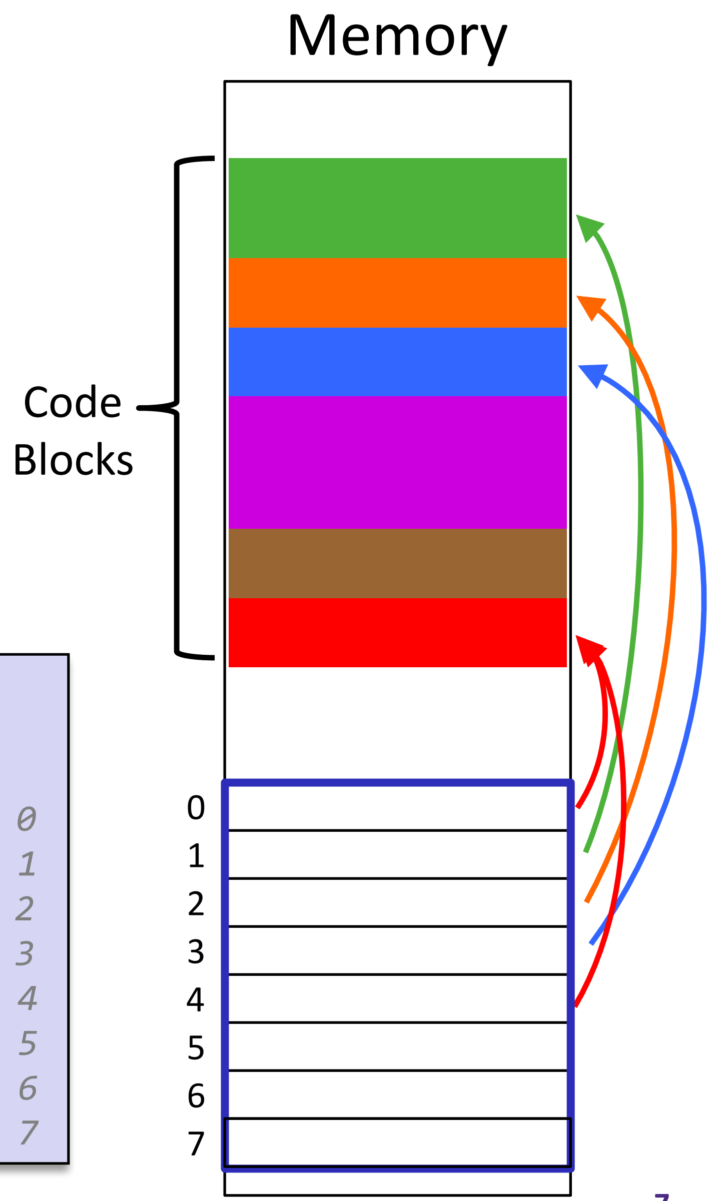
- Switch statements can be implemented using *jump tables* and *indirect jump instructions*
  - Jump tables are arrays of pointers to code blocks
  - Indirect jump jumps to address stored somewhere in memory instead of target specified in instruction

```
switch (x) {
  case 1: <code> break;
  case 2: <code>
  case 3: <code> break;
  case 5:
  case 6: <code> break;
  case 7: <code> break;
  default: <code>
}
```

```
cmpq $7, %rdi
ja   .L9   # default
jmp  *.L4(,%rdi,8)
```

Jump table

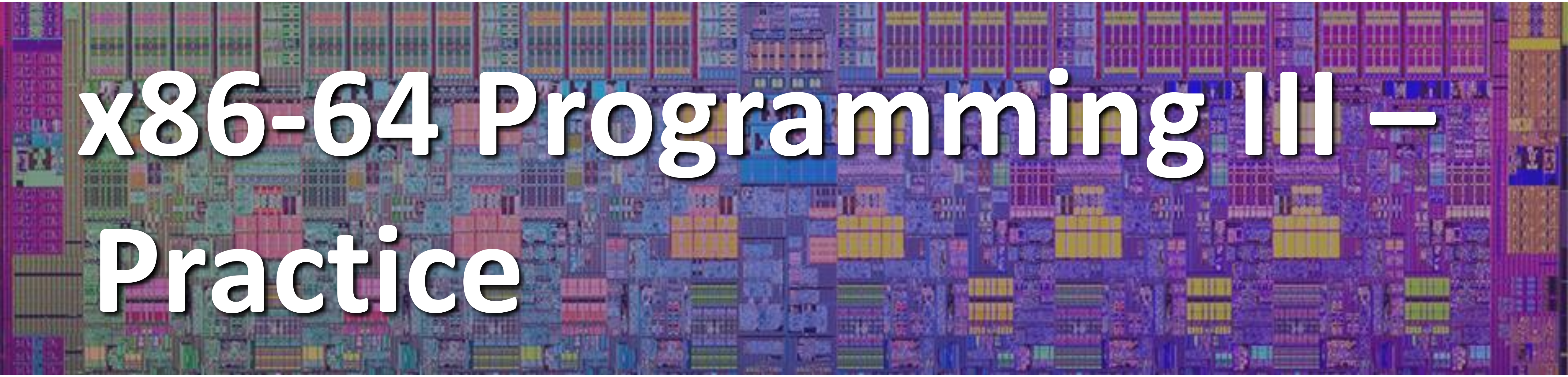
```
.section    .rodata
  .align 8
.L4:
   .quad    .L9    # x = 0
   .quad    .L8    # x = 1
   .quad    .L7    # x = 2
   .quad    .L10   # x = 3
   .quad    .L9    # x = 4
   .quad    .L5    # x = 5
   .quad    .L5    # x = 6
   .quad    .L3    # x = 7
```



Lesson Q&A

- 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?
  - Chat with your neighbors about the lesson for a few minutes to come up with questions



x86-64 Programming III – Practice

Polling Question (1/2)

Register	Use(s)
%rdi	1st argument (x)
%rsi	2nd argument (y)
%rax	return value

[✓ A. cmpq %rsi, %rdi
jle .L4]
y x
x-y

B. cmpq %rsi, %rdi
jg .L4
x-y

C. ~~testq~~ %rsi, %rdi
jle .L4
x&y

D. ~~testq~~ %rsi, %rdi
jg .L4
x&y

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

```
absdiff:

    ______________________

    ______________________
                                # x > y:
    movq     %rdi, %rax
    subq     %rsi, %rax
    ret
.L4:                            # x <= y:
    movq     %rsi, %rax
    subq     %rdi, %rax
    ret
```

x-y <= 0
↑
less than or equal to
(le)

Polling Question (2/2)

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

Line

```
       movl    $0, %eax
.L2:   cmpl    %esi, %eax
       jge     .L4
       movslq  %eax, %rdx
       leaq    (%rdi,%rdx,4), %rcx
       movl    (%rcx), %edx
       addl    $1, %edx
       movl    %edx, (%rcx)
       addl    $1, %eax
       jmp     .L2
.L4:
```

y

i

← Init

} !Test → i-y >= 0

i >= y

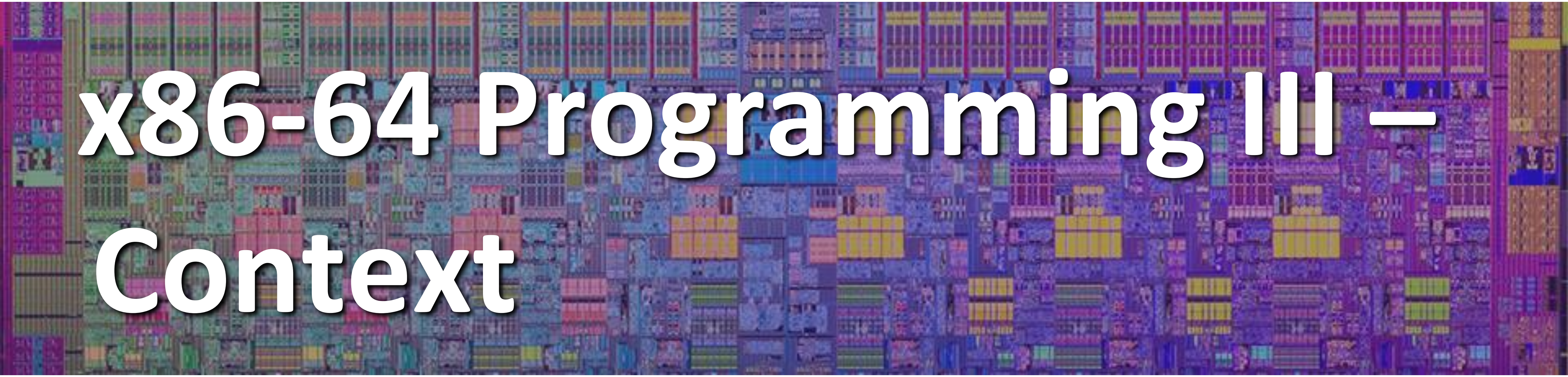
exit

← Update

loop

for(int i = 0 ; i < y ; i++)

Init Test Update



x86-64 Programming III – 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);          cmp
    if (ntest) goto Else;          jle
    result = x-y;
    goto Done;                     jmp
Else:
    result = y-x;
Done:
    return result;
}
```

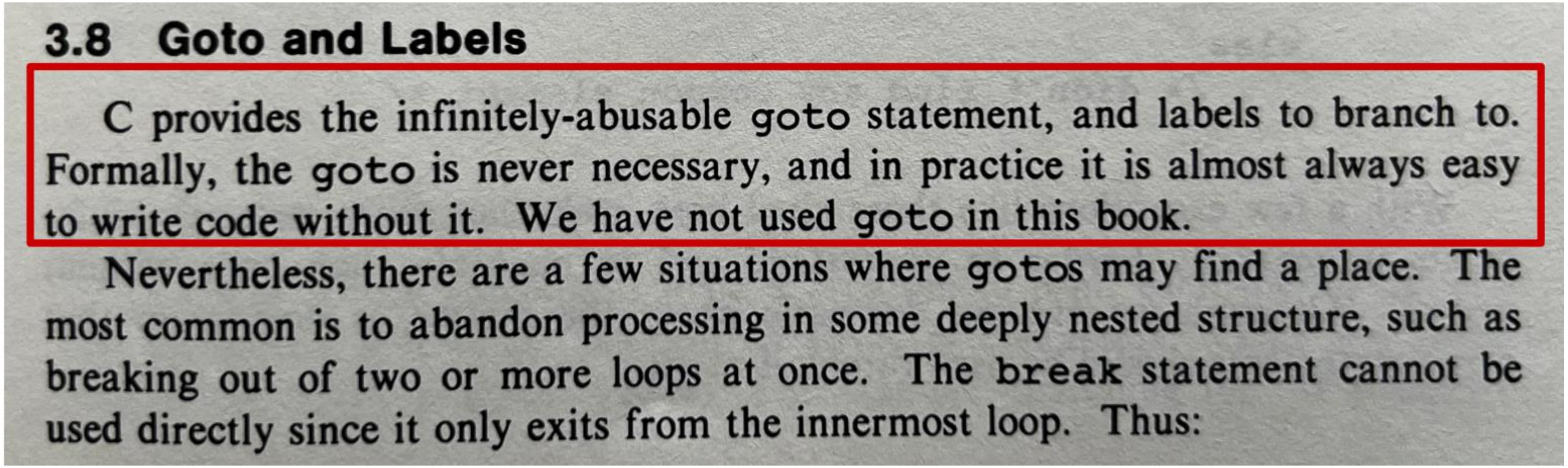
conditional jump

unconditional jump →

labels (addresses)

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:

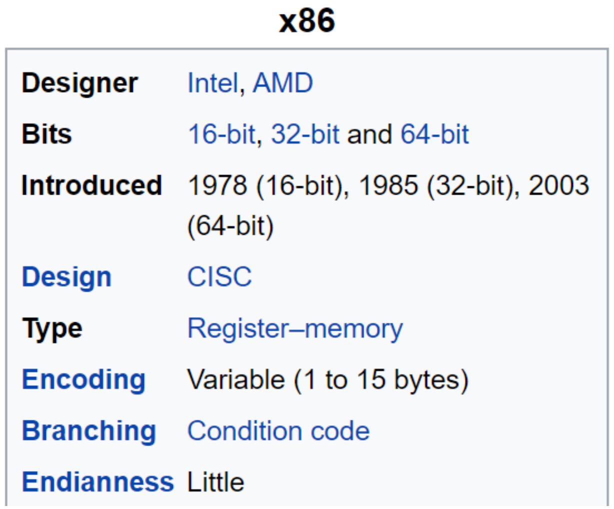


3.8 Goto and Labels

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:

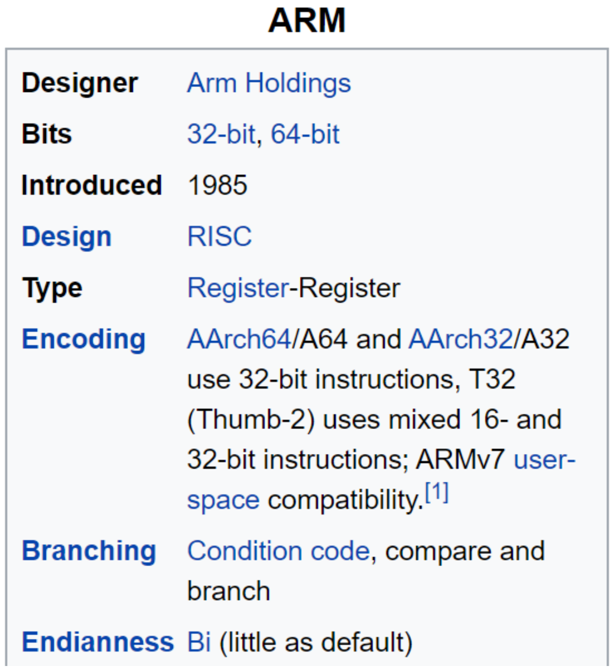
Mainstream ISAs, Revisited



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

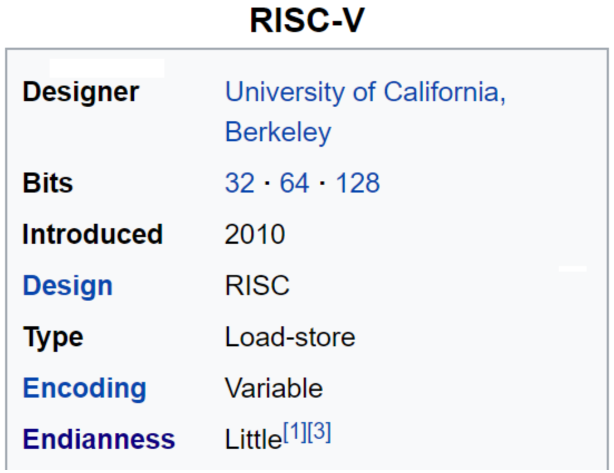
Macbooks & PCs
(Core i3, i5, i7, M)
x86-64 Instruction Set



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?

Group Work Time

- During this time, you are encouraged to work on the following:
  1) If desired, continue your discussion
  2) Work on the homework problems
  3) Work on the lab (if applicable)

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