C Compared to Assembler

Everything we do in C can be done in assembler

C and Assembler are mostly Isomorphic

- The compiler reduces C code to assembler
- It's a mostly one-way trip – decompilers make good thesis / dissertation projects
- We can make better sense out of assembler by looking at how aspects of C translate into assembler
  - This slide deck gives the parallels / closest equivalents to be expanded in later lectures
  - You will be expected to convert single lines of C to assembler on the final exam

Comparison Chart

C	Assembler
Variables with names & addresses	A few reusable registers with fixed names Stack memory Heap memory
Operators	Opcodes, many opcodes
Pointers of various types	Memory addresses (without type)
Types	Sizes (number of bytes)
Constants	Immediate data
Statements terminated by ;	A single opcode and zero or more operands on one line
Preprocessor directives	Assembler directives
No direct control over the stack	Full visibility into the mechanics of the stack
Functions	Functions
If-then, loops, switch	Conditional branching with labels
goto	Unconditional jump to a label

Before we leap into the details

- If this is your first time
  - Don't expect to understand every detail of every slide that follows today – each slide is something like a week of lecture material
  - Do use it as a guide to things you need to master
  - Don't panic

- Been here before?
  - You are on the hook for every detail implied by these slides
  - Be able to look at any bit of C code and be able to turn it into assembler

Storage

- Registers

```
xorq %rax, %rax              # longvar = 0; /* register */
```

- Stack via %rsp or %rbp

```
movl $0, -4(%rbp)  # int i = 0; /* auto */
```

- Heap via a fixed address

```
movq $free_count, %rax  # p = &free_count;
movl $15, (%rax)        # *p = 15;[free_count = 15]
                        # /* static int free_count; */
```

Pointers

- Registers

```
xorq %rax, %rax              # longvar = 0;
```

- Stack via %rsp or %rbp

```
movl $0, -4(%rbp)  # int i = 0;
```

- Heap via a fixed address

```
movq $free_count, %rax  # p = &free_count;
movl $15, (%rax)        # *p = 15;[free_count = 15]
                        # /* static int free_count; */
```

Types and Length

- Registers

```
xorq %rax, %rax              # long longvar = 0;
```

- Stack via %rsp or %rbp

```
movl $0, -4(%rbp)  # int i = 0;
```

- Heap via a fixed address

```
movq $free_count, %rax  # p = &free_count; (int *p;)
movl $15, (%rax)        # *p = 15;
                        #/* static int free_count; */
```

Constants & Immediate

- Registers

```
xorq %rax, %rax              # longvar = 0;
```

- Stack via %rsp or %rbp

```
movl $0, -4(%rbp)  # int i = 0;
```

- Heap via a fixed address

```
movq $free_count, %rax  # p = &free_count;
movl $15, (%rax)        # *p = 15;[free_count = 15]
                        # /* static int free_count; */
```

Conditional branching

```
decq %rax          # j--;
jnz loop_top       # if (j!= 0) goto loop_top;
```

- From this we make loops and all other control flow decisions other than function calls
- No else; just drops through if it doesn't take the jump
- We don't have blocks in assembler – but we can use unconditional branch

Labels

- Are a signpost for an address in the code
- Should use descriptive names such as “loop_top”
- Avoid machine-generated labels!
  - You will see many examples, such as **.LC0**
  - Do not write any similar labels in your code
  - Points will be deducted
  - You might wind up in front of CoAM

Functions & Stack Frame

```
call bit_count          # set up params first!

#set up stack frame - same as enter, but faster
pushq %rbp       #rbp is callee-saved
movq %rsp, %rbp # now %rbp points at the bottom of
                        # our stack frame

leave#tears down stack frame - same as:
     # movq %rbp, %rsp
     # popq %rbp

ret  # %rsp had better point to the return address
```

It’s safe to breathe now

- On something like a weekly basis return to this deck and go over your progress
  - Which of the details now make sense?
  - Which of the details do you now have mastery over?