Last Lecture

- Arrays
  - Elements packed into contiguous region of memory
  - Use index arithmetic to locate individual elements
  - 1D array: address of $A[i] = A + i \times L$
Today’s Menu

- Introduction
  - C program -> assembly code -> machine level code
- Assembly language basics: data, move operation
  - Memory addressing modes
- Operation leaq and Arithmetic & logical operations
- Conditional Statement – Condition Code + cmovX
- Loops
- Function call – Stack – Recursion
  - Overview of Function Call
  - Memory Layout and Stack - x86-64 instructions and registers
  - Passing control
  - Passing data – Calling Conventions
  - Managing local data
  - Recursion
- Array
  - Floating-point operations
  - x86-64 - Buffer Overflow
2D Array

\[ T \ A[R][C]; \]
- 2D array of data type \( T \)
- \( R \) rows, \( C \) columns
- Type \( T \) element requires \( L \) bytes ( \text{sizeof}(T) \)

- Array Size
  - Contiguously allocated region of \( R \times C \times L \) bytes in memory

- Arrangement
  - Row-Major Ordering – Example: int \( A[R][C] \);

\[ \begin{bmatrix}
A[0][0] & \cdots & A[0][C-1] \\
\vdots & & \vdots \\
A[R-1][0] & \cdots & A[R-1][C-1]
\end{bmatrix} \]
Accessing a row of 2D array

\[ T \ \text{A}[R][C]; \]

- \( \text{A}[i] \) is array of \( C \) elements
- Each element of type \( T \) requires \( L \) bytes
- Address of each row: \( \text{A} + i \times (C \times L) \)
- Example: \( \text{int} \ \text{A}[R][C]; \)
Accessing an element of 2D array

\[
T \ A[R][C];
\]

- \(A[i][j]\) is element of type \(T\), which requires \(L\) bytes
- Address of each element \(A[i][j]\):
  \[
  A + i * (C * L) + j * L = A + (i * C + j) * L
  \]
- Example: \(\text{int } A[R][C];\)

\[
\begin{align*}
A[0][0] & \quad \ldots \quad A[0][C-1] \quad \ldots \quad \ldots \\
A[i][0] & \quad \ldots \quad A[i][j] \\
A[R-1][0] & \quad \ldots \quad A[R-1][C-1]
\end{align*}
\]

\[
A + (i*C*4) + (j*4) = A + (i*C+j)*4
\]
Accessing an element of 2D array - Example

- See copy function from Assignment 5
Storing Data in Memory

Two Methods

- Fixed memory
  - Dedicated purpose through life of program
- Stack memory - stack frame
  - Temporarily use and recycle
  - Lasts through life of function call
Storing Data in Memory

Fixed Memory

- Declare using a label & a directive for size
  - label: is a shorthand for the **address**
  - size: .byte, .word, .long, .quad
  - initial value

- Example: C: ```
  long x = 6;
  long y = 9;
  void main {
    ...
  }
``` x86-64:
  ```
  x: .quad 6 # 0x0000000000000006
  y: .quad 9 # 0x0000000000000009
  ```
Storing Data in Memory
– Fixed Memory – Example

#define N 6

int A[N] = {12, 34, 56, 78, -90, 1};

void main () {
    printf("The total is %d.\n", sum_array(A,N));
    return;
}

A:

.long 12  # or .long 12, 34, 56, 78, -90, 1
.long 34
.long 56
.long 78
.long -90
.long 1

.ident "GCC: (Ubuntu 7.3.0-2iubuntu1~16.04 7.3.0) 7.3.0"

.section .note.GNU-stack,"",@progbits
Storing Data in Memory – Stack Memory – Example

```c
void main(int argc, char * argv) {
    int A[] = {12, 34, 56, 78, -90, 1};
    printf("The total is %d.\n", sum_array(A, N));
    return;
}
```

```
main:
.LFB38:
    .cfi_startproc
    subq  $40, %rsp
    .cfi_def_cfa_offset 48
    movl  $6, %esi
    movq  %fs:40, %rax
    movq  %rax, 24(%rsp)
    xorl  %eax, %eax
    movabsq $146028888076, %rax
    movq  %rsp, %rdi
    movq  %rax, (%rsp)
    movabsq $335007449144, %rax
    movq  %rax, 8(%rsp)
    movabsq $8589934502, %rax
    movq  %rax, 16(%rsp)
    call  sum array
    movl  $.LC0, %esi
    movl  %eax, %edx
    movl  $1, %edi
    xorl  %eax, %eax
    call  __printf_chk
    movq  24(%rsp), %rax
    xorq  %fs:40, %rax
    jne  .L5
    addq  $40, %rsp
    .cfi_remember_state
    .cfi_def_cfa_offset 8
    ret
```

11
Spilling to Stack

- When manipulating data (like array elements), it is time efficient to make use of registers.
- If all registers are used OR we want to reuse a register like `rax`, then we use the stack -> called *spill* to stack.
- Example: Perform dot product of two char matrices A and B (2x2) -> Here, only showing: \( A_{00} \cdot B_{00} + A_{01} \cdot B_{10} \)

```assembly
# A in %rdi, B in %rsi, dot product in %rax
movq (%rdi), %rax    # store A_{00} in %rax
imul (%rsi), %rax    # A_{00} \cdot B_{00}
pushq %rax           # store partial result on stack
movq 1(%rdi), %rax   # overwrite %rax with A_{01}
imul 2(%rsi), %rax   # A_{01} \cdot B_{10}
addq (%rsp), %rax    # A_{01} \cdot B_{10} + A_{00} \cdot B_{00}
leal 8(%rsp), %rsp   # restore %rsp (same as addq $8, %rsp)
# note: cannot "popq %rax" - why?
```
Summary

- Arrays
  - Elements packed into contiguous region of memory
  - Use index arithmetic to locate individual elements
  - 2D array: address of
    \[ A[i][j] = A + i \times (C \times L) + j \times L \]
- Two Methods to save arrays (and data in general)
  - Fixed memory
  - Stack memory - stack frame
- Spill to stack
Next Lecture

- Introduction
  - C program -> assembly code -> machine level code
- Assembly language basics: data, move operation
  - Memory addressing modes
- Operation leaq and Arithmetic & logical operations
- Conditional Statement – Condition Code + cmovX
- Loops
- Function call – Stack – Recursion
  - Overview of Function Call
  - Memory Layout and Stack - x86-64 instructions and registers
  - Passing control
  - Passing data – Calling Conventions
  - Managing local data
  - Recursion
- Array
- Floating-point operations
- x86-64 - Buffer Overflow