Last Lecture

- Recursion
  - Handled without special consideration using ...
    - Stack frames
    - x86-64 Function call and Register saving conventions
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
1D Array

\[ T \ A[n]; \]
- Array of data type \( T \) and length \( n \)
- Contiguously allocated region of \( n \times L \) bytes in memory where \( L = \text{sizeof}(T) \)

- \( \text{char x[12]}; \)
  - \( x \) to \( x + 12 \)

- \( \text{int x[5]}; \)
  - \( x \) to \( x + 8 \)
  - \( x + 8 \) to \( x + 16 \)

- \( \text{long x[3]}; \)
  - \( x \) to \( x + 16 \)
  - \( x + 16 \) to \( x + 24 \)

- \( \text{char *x[3];} \)
  - \( x \) to \( x + 8 \)
  - \( x + 8 \) to \( x + 16 \)
  - \( x + 16 \) to \( x + 24 \)
Accessing 1D Array

- Address of \( A[i] = \) base address + \( i \times L \)
- \( A \) can be used as a pointer to array element 0
- Can increment a pointer \( A \) by adding \( L \) to the address

```
int x[5];
```

<table>
<thead>
<tr>
<th>Reference</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x[4] )</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>( x )</td>
<td>int *</td>
<td></td>
</tr>
<tr>
<td>( x + 1 )</td>
<td>int *</td>
<td></td>
</tr>
<tr>
<td>&amp;( x[2] )</td>
<td>int *</td>
<td></td>
</tr>
<tr>
<td>( x[5] )</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>*( x+1 )</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>( x + i )</td>
<td>int *</td>
<td></td>
</tr>
</tbody>
</table>
Manipulating 1D array – Example - `main.c`

```c
#include <stdio.h>
#include <stdlib.h>

char sumChar(char *, int);
short sumShort(short *, int);
int sumInt(int *, int);
long sumLong(long *, int);

#define N 6
char AC[N] = {-58, 22, 101, -15, 72, 27}; // sum = -107
short AS[N] = {-58, 22, 101, -15, 72, 27}; // sum = 149
int AI[N] = {258, 522, 1010, -15, -3372, 27}; // sum = -1570
long AL[N] = {258, 522, 1010, -15, -3372, 27}; // sum = -1570

void main () {
    printf("The total of AC is %d.\n", sumChar(AC,N));
    printf("The total of AS is %d.\n", sumShort(AS,N));
    printf("The total of AI is %d.\n", sumInt(AI,N));
    printf("The total of AL is %ld.\n", sumLong(AL,N));
    return;
}
```

Test cases

<table>
<thead>
<tr>
<th>Array</th>
<th>Elements</th>
<th>Expected sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>AC[N] = {-58, 22, 101, -15, 72, 27}</td>
<td>-107</td>
</tr>
<tr>
<td>short</td>
<td>AS[N] = {-58, 22, 101, -15, 72, 27}</td>
<td>149</td>
</tr>
<tr>
<td>int</td>
<td>AI[N] = {258, 522, 1010, -15, -3372, 27}</td>
<td>-1570</td>
</tr>
<tr>
<td>long</td>
<td>AL[N] = {258, 522, 1010, -15, -3372, 27}</td>
<td>-1570</td>
</tr>
</tbody>
</table>
Manipulating 1D array – Example - sum.s

Part 1

- Register `%rdi` contains starting address of array
- Register `%rcx` contains array index

```
.globl sumChar
sumChar:
  movl $0, %eax
  movl $0, %ecx
loopChar:
  cmpl %ecx, %esi
  jle endlooppl
  addb (%rdi, %rcx, 1), %al
  incl %ecx
  jmp loopChar
endlooppl:
  ret
```

```
.globl sumShort
sumShort:
  xorl %eax, %eax
  xorl %ecx, %ecx
  jmp cond1
loopShort:
  addw (%rdi, %rcx, 2), %ax
  incl %ecx
cond1:
  cmpl %esi, %ecx
  jne loopShort
ret
```
Manipulating 1D array – Example – sum.s

Part 2

- Register %rdi contains starting address of array
- Register %rcx contains array index

```assembly
.globl sumInt
sumInt:
xorl %eax, %eax
xorl %ecx, %ecx
jmp cond2
loopInt:
  addl (%rdi, %rcx, 4), %eax
  incl %ecx
cond2:
  cmpl %esi, %ecx
  jne loopInt
ret
```

```assembly
.globl sumLong
sumLong:
  movq $0, %rax
  movl $0, %ecx
loopLong:
  cmpl %ecx, %esi
  jle endloop
  addq (%rdi, %rcx, 8), %rax
  incl %ecx
  jmp loopLong
endloop:
  ret
```
Summary

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