CS 354 - Machine Organization & Programming
Thursday, November 14, 2019

Project p5 (4.5%): DUE at 10 pm on Wednesday, December 2nd
Project p6 (4.5%): Assigned on Tuesday, November 26th

Homework hw6 (1.5%): DUE at 10 pm on Wednesday, November 20th

Last Time

Register Usage Conventions
Function Call-Return Example
Recursion
Stack Allocated Arrays in C
Stack Allocated Arrays in Assembly

Today

Stack Allocated Arrays in Assembly (from last time)
Stack Allocated Multidimensional Arrays
Stack Allocated Structs
Alignment
Alignment Practice
Unions
Exams Returned

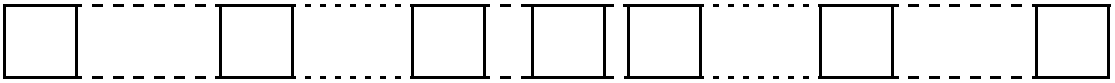
Next Time

Pointers in Assembly, and Stack Smashing
Read: B&O 3.10, 3.12

Stack Allocated Multidimensional Arrays

Recall 2D Array Basics

T `A[`*R*`][`*C*`];` where *T* is the element datatype of size *L* bytes,
R is the number of rows and *C* is the number of columns



❋ *Recall that 2D arrays are stored on the stack*

```
int A[5][3];        typedef int row_t[3];
                    row_t A[5];
```

Accessing 2D Arrays in Assembly

```
&A[i][j]
```

Given array `A` as declared above, if x_A in %eax, i in %ecx, j in %edx
then `A[i][j]` in assembly is:

```
leal (%ecx, %ecx, 2), %ecx
sall $2, %edx
addl %eax, %edx
movl (%edx, %ecx, 4), %eax
```

Compiler Optimizations

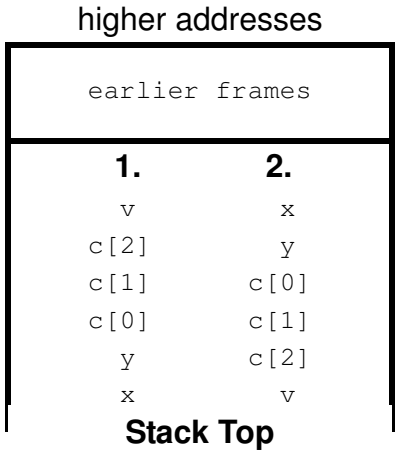
- If only accessing part of array

- If taking a fixed stride through the array

Stack Allocated Structures

Structures on the Stack

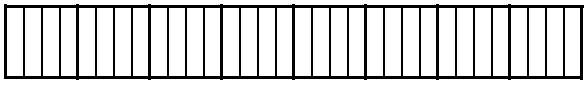
```
struct iCell {
   int x;
   int y;
   int c[3];
   int *v;
};
```



→ How is a structure laid out on the stack? Option 1 or 2:

The compiler

-



-

❋ *The first data member of a structure*

Accessing Structures in Assembly

Given:

```
struct iCell ic = //assume ic is initialized

void function(iCell *ip) {
```

→ Assume `ic` is at the top of the stack, %edx stores `ip` and %esi stores `i`.
Determine for each the assembly instruction to move the C code's value into %eax:

C code	assembly
1. `ic.v`	
2. `ic.c[i]`	
3. `ip->x`	
4. `ip->y`	
5. `&ip->c[i]`	

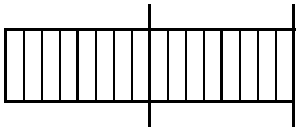
❋ *Assembly code to access a structure*

Alignment

What?

Why?

Example: Assume cpu reads 8 byte words
f is a misaligned float



Restrictions

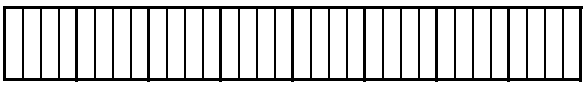
Linux: short
int, float, pointer, double

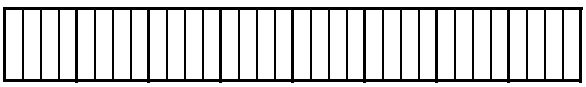
Windows: same as Linux except
double

Implications

Structure Example

```
struct s1 {
   int i;
   char c;
   int j;
};
```



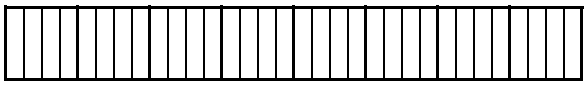


❋ *The total size of a structure*

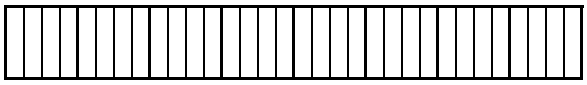
Alignment Practice

→ For each structure below, complete the memory layout and determine the total bytes allocated.

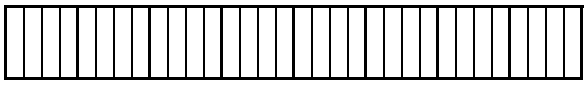
1)
```
struct sA {
   int i;
   int j;
   char c;
};
```



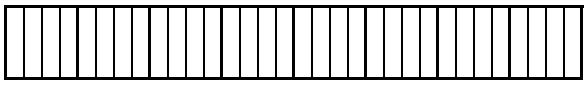
2)
```
struct sB {
   char a;
   char b;
   char c;
};
```



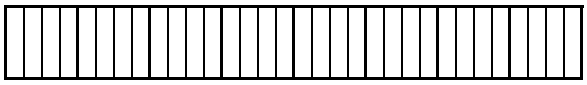
3)
```
struct sC {
   char c;
   short s;
   int i;
   char d;
};
```



4)
```
struct sD {
   short s;
   int i;
   char c;
};
```



5)
```
struct sE {
   int i;
   short s;
   char c;
};
```



❋ *The order that a structure's data members are listed*

Unions

What? A *union* is

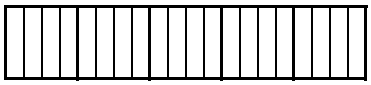
-
-

Why?

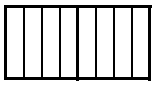
-
-
-

How?

```
struct s {
   char c;
   int i[2];
   double d;
};
```



```
union u {
   char c;
   int i[2];
   double d;
};
```



Example

```
typedef union {
   unsigned char cntrlrByte;
   struct {
      unsigned char playbutn  : 1;
      unsigned char pausebutn : 1;
      unsigned char ctrlbutn  : 1;
      unsigned char fire1butn : 1;
      unsigned char fire2butn : 1;
      unsigned char direction : 3;
   } bits;
} CntrlrReg;
```