15-213

"The course that gives CMU its Zip!"

Machine-Level Programming IV: Data
Sept. 20, 2006

Structured Data
- Arrays
- Structs
- Unions

Data/Control
- Buffer overflow

class07.ppt

Basic Data Types

Integral
- Stored & operated on in general registers
- Signed vs. unsigned depends on instructions used

Intel	GAS	Bytes	C
byte	b	1	[unsigned] char
word	w	2	[unsigned] short
double word	l	4	[unsigned] int
quad word	q	8	[unsigned] long int (x86-64)

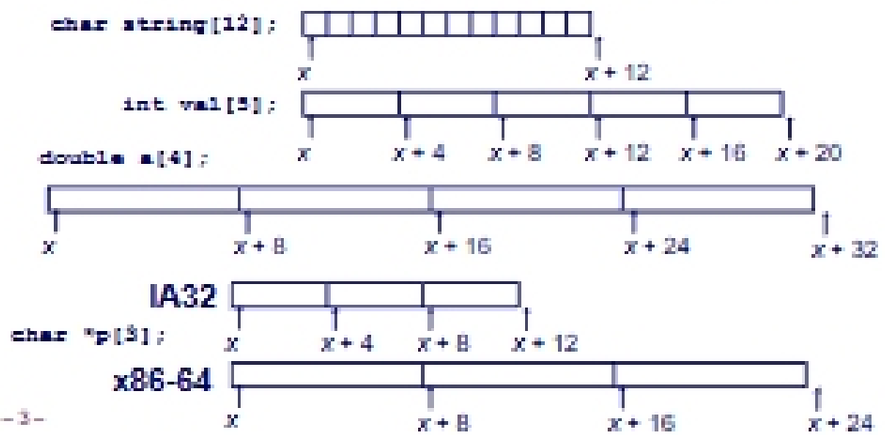
Floating Point
- Stored & operated on in floating point registers

Intel	GAS	Bytes	C
Single	s	4	float
Double	l	8	double
Extended	t	10/12/16	long double

Array Allocation

Basic Principle

`T A[L];`
- Array of data type *T* and length *L*
- Contiguously allocated region of *L* * `sizeof(T)` bytes



Array Access

Basic Principle

`T A[L];`
- Array of data type *T* and length *L*
- Identifier `A` can be used as a pointer to array element 0
  - Type `T*`

Reference	Type	Value
`val[4]`	`int`	3
`val`	`int *`	*x*
`val+1`	`int *`	*x* + 4
`&val[2]`	`int *`	*x* + 8
`val[5]`	`int`	??
`*(val+1)`	`int`	5
`val + i`	`int *`	*x* + 4*i*

Array Example

```
typedef int zip_dig[5];

zip_dig cmu = { 1, 5, 2, 1, 3 };
zip_dig mit = { 0, 2, 1, 3, 9 };
zip_dig ucb = { 9, 4, 7, 2, 0 };
```

Notes

- Declaration "`zip_dig cmu`" equivalent to "`int cmu[5]`"
- Example arrays were allocated in successive 20 byte blocks
  - Not guaranteed to happen in general

Array Accessing Example

Computation

- Register `%edx` contains starting address of array
- Register `%eax` contains array index
- Desired digit at `4*%eax + %edx`
- Use memory reference (`%edx,%eax,4`)

```
int get_digit
  (zip_dig z, int dig)
{
  return z[dig];
}
```

IA32 Memory Reference Code

```
# %edx = z
# %eax = dig
movl (%edx,%eax,4),%eax # z[dig]
```


Referencing Examples

Code Does Not Do Any Bounds Checking!

Reference	Address	Value	Guaranteed?
`mit[3]`	`36 + 4* 3 = 48`	3	Yes
`mit[5]`	`36 + 4* 5 = 56`	9	No
`mit[-1]`	`36 + 4*-1 = 32`	3	No
`cmu[15]`	`16 + 4*15 = 76`	??	No

- Out of range behavior implementation-dependent
  - No guaranteed relative allocation of different arrays

Array Loop Example

Original Source

```
int sd2int(zip_dig z)
{
  int i;
  int zi = 0;
  for (i = 0; i < 5; i++) {
    zi = 10 * zi + z[i];
  }
  return zi;
}
```

Transformed Version

- As generated by GCC
- Eliminate loop variable `i`
- Convert array code to pointer code
- Express in do-while form
  - No need to test at entrance

```
int sd2int(zip_dig z)
{
  int zi = 0;
  int *zend = z + 4;
  do {
    zi = 10 * zi + *z;
    z++;
  } while (z <= zend);
  return zi;
}
```

Array Loop Implementation (IA32)

Registers

- %ecx z
- %eax zi
- %ebx zend

Computations

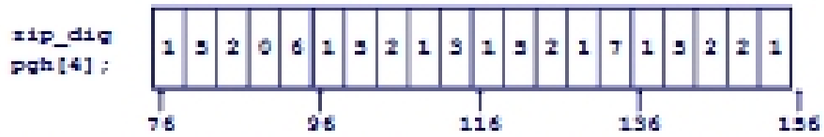
- 10*zi + *z implemented as *z + 2*(zi+4*zi)
- z++ increments by 4

```
int zd2int(zip_dig z)
{
  int zi = 0;
  int *zend = z + 4;
  do {
    zi = 10 * zi + *z;
    z++;
  } while(z <= zend);
  return zi;
}
```

```
  # %ecx = z
  xorl %eax,%eax          # zi = 0
  leal 16(%ecx),%ebx      # zend  = z+4
.L59:
  leal (%eax,%eax,4),%edx # 5*zi
  movl (%ecx),%eax        # *z
  addl $4,%ecx            # z++
  leal (%eax,%edx,2),%eax # zi = *z + 2*(5*zi)
  cmpl %ebx,%ecx          # z : zend
  jle .L59                # if <= goto loop
```


Nested Array Example

```
#define PCOUNT 4
zip_dig pgh[PCOUNT] =
  {{1, 5, 2, 0, 6},
   {1, 5, 2, 1, 3 },
   {1, 5, 2, 1, 7 },
   {1, 5, 2, 2, 1 }};
```



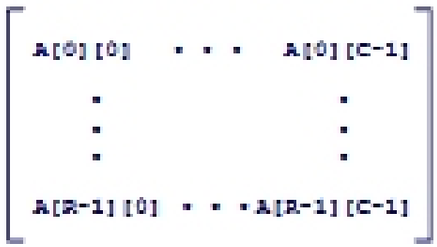
- Declaration "zip_dig pgh[4]" equivalent to "int pgh[4][5]"
  - Variable pgh denotes array of 4 elements
    - Allocated contiguously
  - Each element is an array of 6 int's
    - Allocated contiguously
- "Row-Major" ordering of all elements guaranteed

Viewing as Multidimensional Array

Declaration

T A[*R*][*C*];

- 2D array of data type *T*
- *R* rows, *C* columns
- Type *T* element requires *K* bytes



Array Size

- *R* * *C* * *K* bytes

Arrangement

- Row-Major Ordering

int A[R][C];

Nested Array Row Access

Row Vectors

- A[1] is array of *C* elements
- Each element of type *T*
- Starting address A + *i* * (*C* * *K*)

int A[R][C];