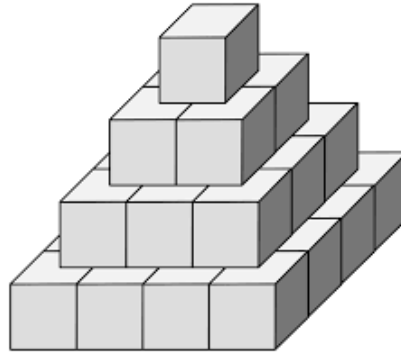
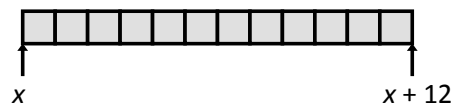


Machine Code: Structures

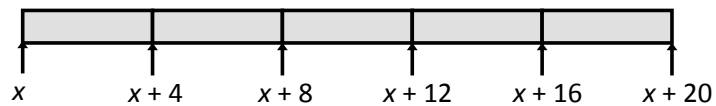


Arrays

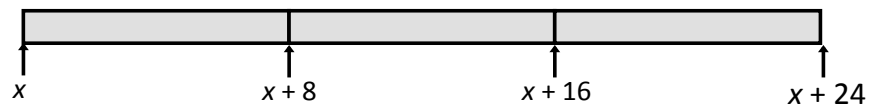
```
char string[12];
```



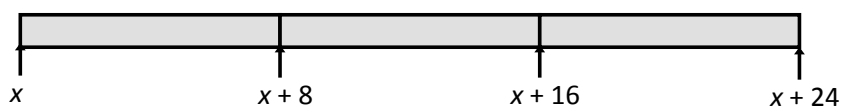
```
int val[5];
```



```
double a[3];
```



```
char* p[3];
```



Array Access

```
int get_val(int a[], int i) {  
    return a[i];  
}
```

```
# %rdi = a  
# %rsi = i  
movl (%rdi,%rsi,4), %eax # a[i]
```

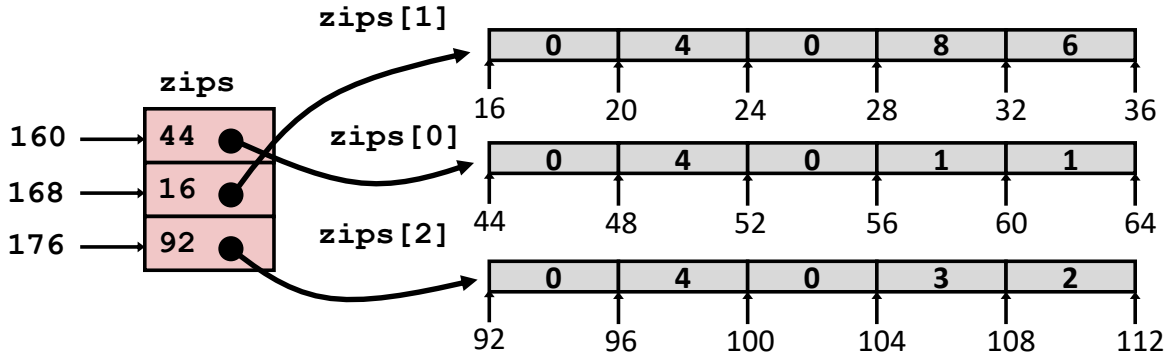
Array Loop

```
void inc5(int a[]) {  
    int i;  
    for (i = 0; i < 5; i++) {  
        a[i]++;  
    }  
}
```

```
# %rdi = a  
movl $0, %eax # i = 0  
jmp .L3 # goto middle  
.L4: # loop:  
addl $1, (%rdi,%rax,4) # a[i]++  
addq $1, %rax # i++  
.L3: # middle:  
cmpq $4, %rax # i:4  
jbe .L4 # if <=, goto loop  
ret
```

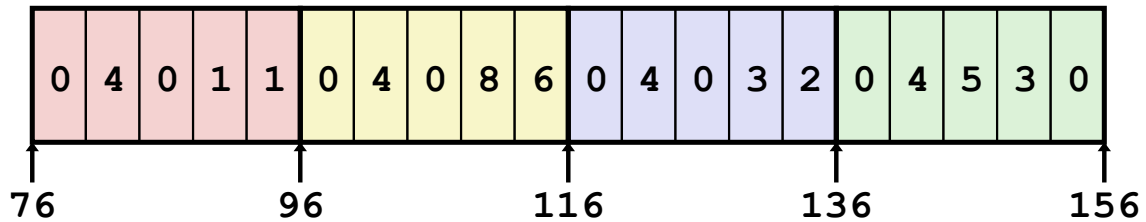
Multi-Level Array

```
int* zips[3]; // stored at address 160
zips[0] = (int*) malloc(sizeof(int) * 5);
zips[0][1] = 4;
...
```



Nested Array

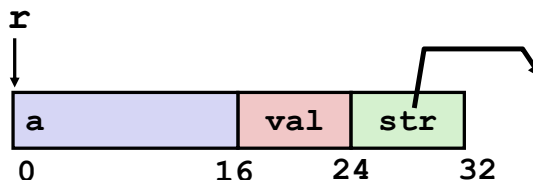
```
int zips[4][5]; // stored at address 76
zips[0][1] = 4;
...
```



Structs

New 32-byte type!

```
struct thing {  
    int a[4];  
    long val;  
    char* str;  
};
```



```
struct thing x; // 32 bytes  
struct thing y; // 32 bytes
```

```
x.val = 5;  
x.a[1] = 2;  
x.str = "hello";
```

```
y = x; // copy full struct
```

```
struct thing* p; // 8 bytes  
p = malloc(sizeof(struct thing));
```

```
// form 1  
(*p).val = 7; // NOT p.val = 7
```

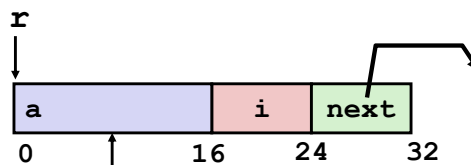
```
// form 2  
p->val = 7; // preferred to above
```

```
struct thing* p2;  
p2 = p; // just a pointer copy
```

Linked List Example

```
struct list_el {  
    int a[4];  
    long i;  
    struct list_el* next;  
};
```

```
void set_val  
(struct list_el* n, int val) {  
    while (n) {  
        long i = n->i;  
        n->a[i] = val;  
        n = n->next;  
    }  
}
```



Element i

Register	Value
%rdi	n
%rsi	val

```
.L1:  
    movq    16(%rdi), %rax    # loop:  
    movl    %esi, (%rdi,%rax,4) # i = M[n+16]  
    movq    24(%rdi), %rdi    # M[n+4*i] = val  
    testq   %rdi, %rdi       # n = M[n+24]  
    jne     .L1              # Test n  
                                # if !=0 goto loop
```

Struct Naming: typedef

```
// give type T another name: U
typedef T U;

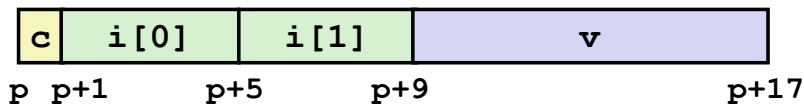
// idiomatic struct usage:
// defines a type "struct thing" with alias "thing"
// T is "struct thing { ... }", U is "thing"
typedef struct thing {
    ...
} thing;

thing x; // now can omit "struct" from type name
x.i = 5;

thing* p = malloc(sizeof(thing)); // easier!
p->i = 3;
```

Data Alignment

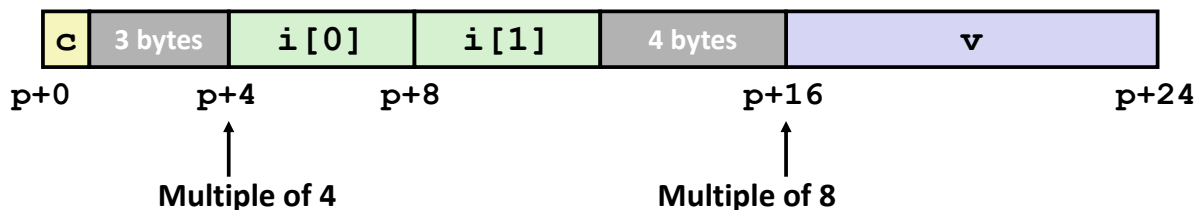
Unaligned Data



```
struct S {
    char c;
    int i[2];
    double v;
};
```

Alignment Rule: addr of each field must be a multiple of the field size

Aligned Data

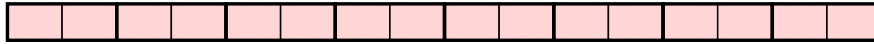


Floating Point: YMM/XMM Registers

■ 16 single-byte integers



■ 8 16-bit integers



■ 4 32-bit integers



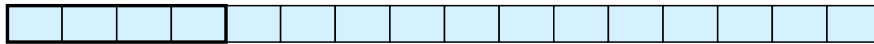
■ 4 single-precision floats



■ 2 double-precision floats



■ 1 single-precision float



■ 1 double-precision float



Different FP instruction sets! (SSE, AVX, ...) **vcvttssd2siq**