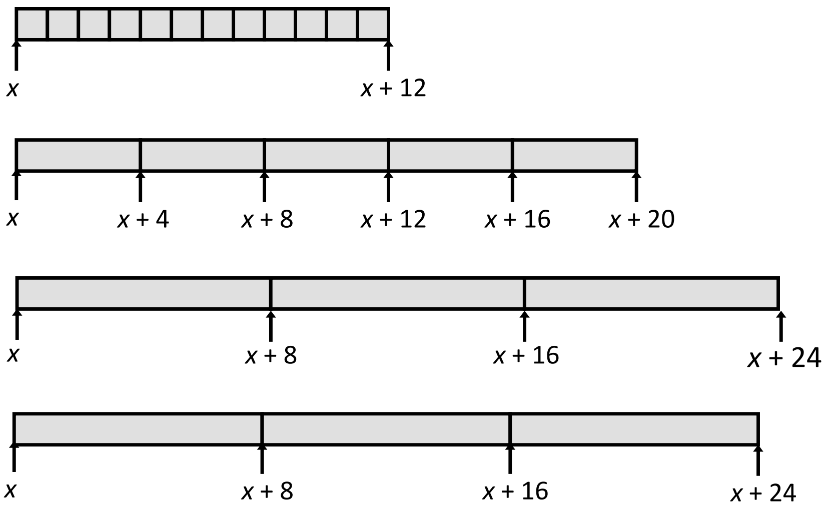
# Array Allocation



`char string[12];`

x

x + 12

`int val[5];`

x

x + 4

x + 8

x + 12

x + 16

x + 20

`double a[3];`

x

x + 8

x + 16

x + 24

`char* p[3];`

x

x + 8

x + 16

x + 24

# Array Access

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

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

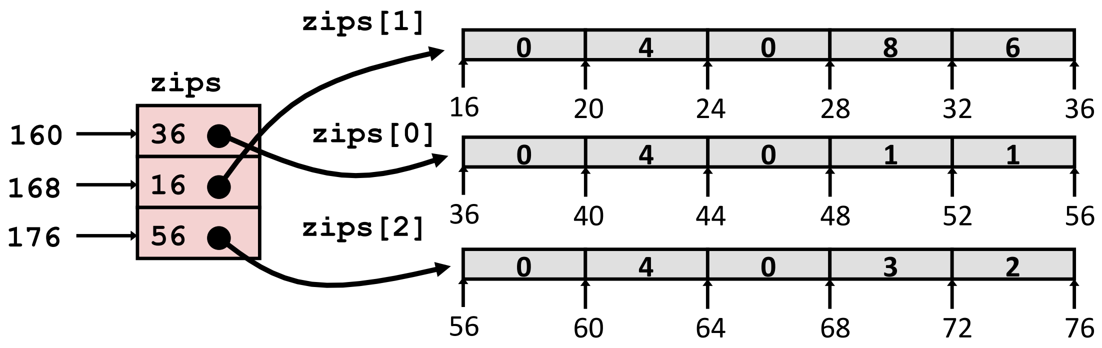
# Array Looping

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

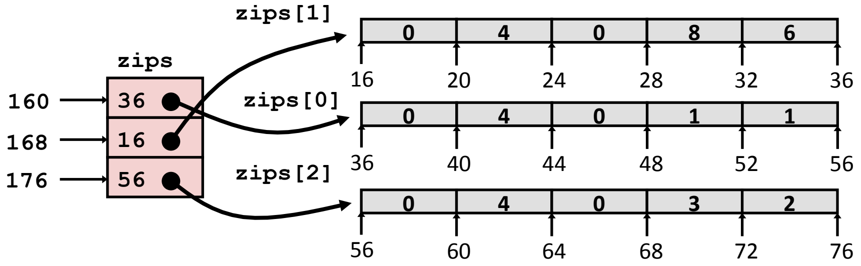
```
  # %rdi = a
  movl     $0, %eax           #   i = 0
  jmp      .L3                #   goto middle
.L4:                          # loop:
  addl     $1, (%rdi,%rax,4)  #   a[i]++
  addq     $1, %rax           #   i++
.L3:                          # middle
  cmpq     $4, %rax           #   i:4
  jbe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

# Multi-Level Array Example

```
int* zips[3];
zips[0] = (int*) malloc(sizeof(int)*5);
...
```



# Multi-Level Array Example

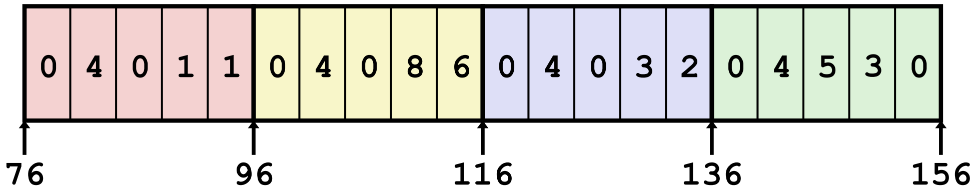


```
int get_zip_digit
  (size_t index, size_t digit)
{
  return zips[index][digit];
}
```

```
salq    $2, %rsi               # 4*digit
addq    160(,%rdi,8), %rsi     # p = zips[index] + 4*digit
movl    (%rsi), %eax           # return *p
ret
```

# Nested Array Example

```
int zips[4][5];
```

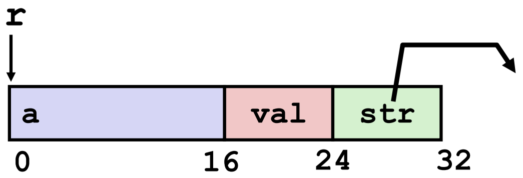


```
int* get_zip(int index)
{
  return zips[index];
}
```

```
# %rdi = index
 leaq (%rdi,%rdi,4),%rax # 5 * index
 leaq 76(,%rax,4),%rax   # zips + (20 * index)
```

# Structs

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



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

x.val = 5;
x.a[1] = 2;
x.str = “hello”;

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

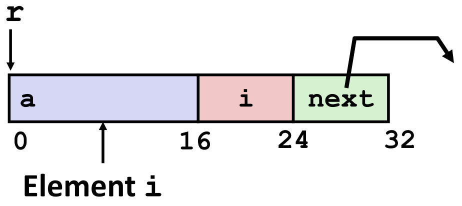
```
struct thing* p; // 8 bytes
p = malloc(sizeof(struct thing));

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

// form 2 (preferred)
p->val = 7;

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

# typedef

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


// defines a type “struct thing” with alias “thing”
// T is “struct thing { ... }”, U is “thing”
typedef struct thing {
  ...
} thing;


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


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

# Linked List Example

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



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

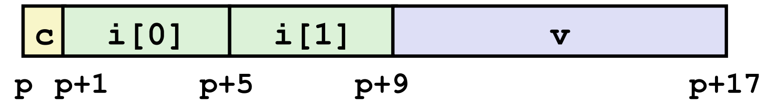
| Register | Value |
|---|---|
| %rdi | n |
| %rsi | val |

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

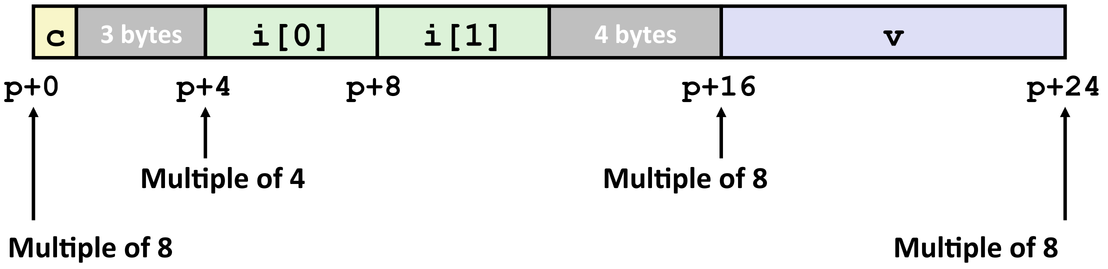
# Data Alignment

```
struct S1 {
  char c;
  int i[2];
  double v;
} *p;
```

## Unaligned Data

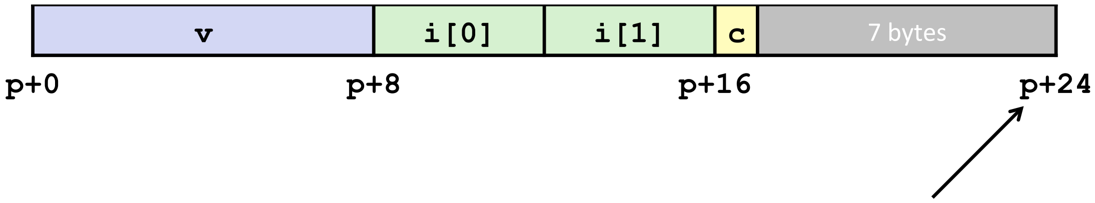


## Aligned Data



# Struct Data Alignment

```
struct S2 {
  double v;
  int i[2];
  char c;
} *p;
```



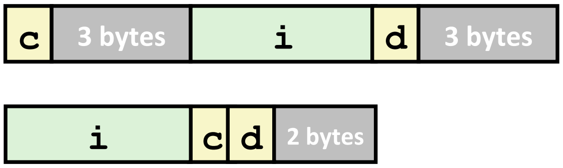
Multiple of 8 (largest alignment in struct)

# Saving Space

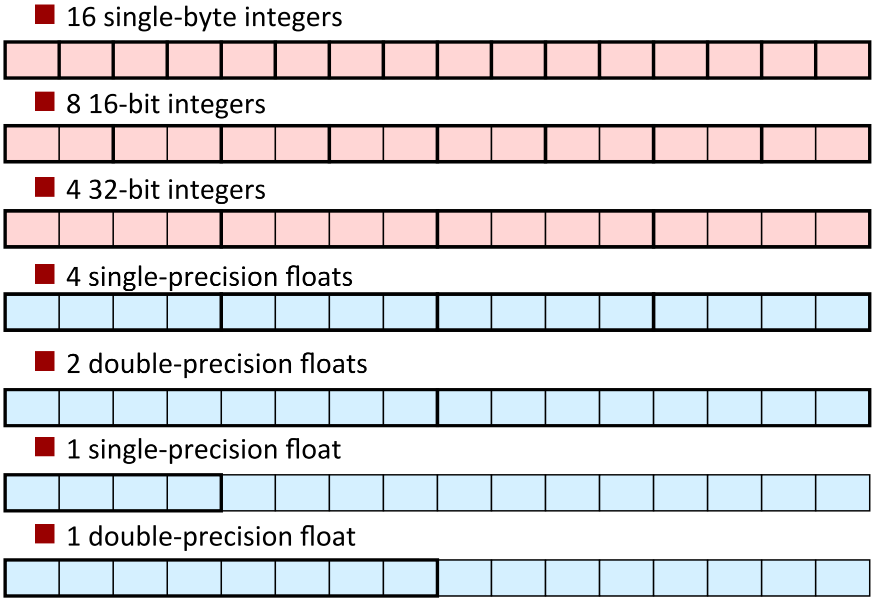
## Put large data types first

```
struct S4 {
  char c;
  int i;
  char d;
} *p;
```

```
struct S5 {
  int i;
  char c;
  char d;
} *p;
```



# 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