Machine Program: Data

Zhaoguo Wang

Today

Arrays

- One-dimensional
- Multi-dimensional (nested)
- Multi-level

Structures

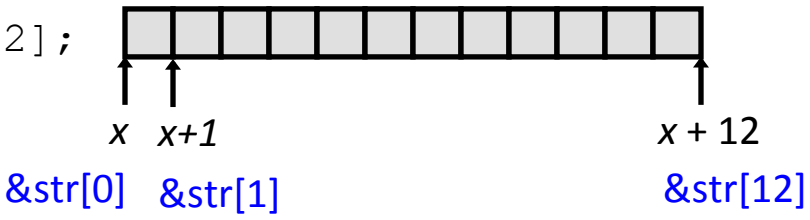
- Allocation
- Access
- Alignment

Array Allocation

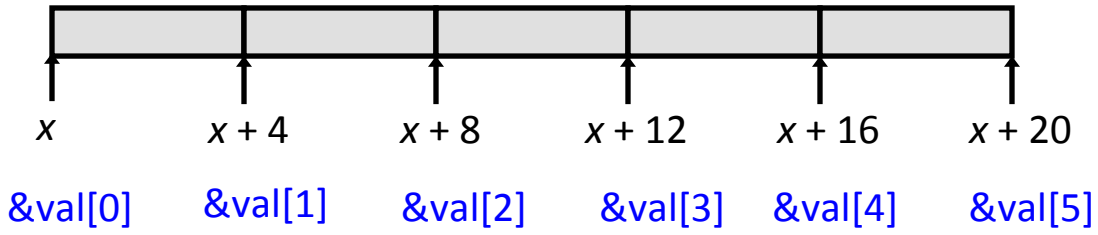
T **`A[`***L***`];`**

– Contiguously allocated region of *L* * **`sizeof`**`(`*T*`)` bytes in memory

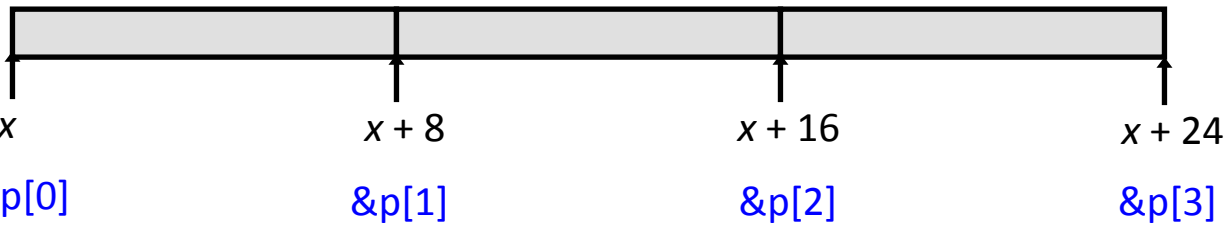
`char str[12];`



`int val[5];`



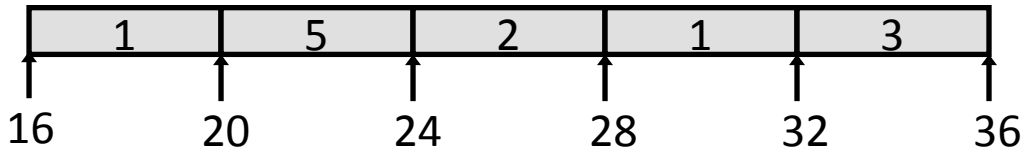
`char *p[3];`



Array Accessing Example

```
#define LEN 5

int arr[LEN];
```



C code

```
int
get_digit(int *arr, int i)
{
  return arr[i];
}
```

%rdi contains starting address of array

%rsi contains the index i

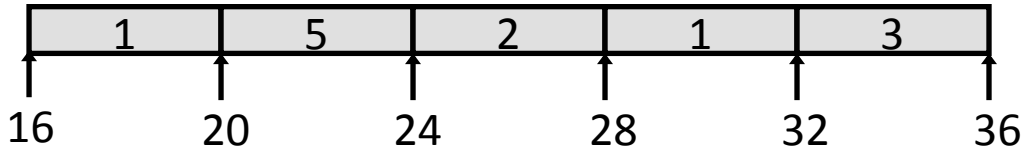
Assembly code

```
??? ???
```

Array Accessing Example

```
#define LEN 5

int arr[LEN];
```




```
int
get_digit(int *arr, int i)
{
  return arr[i];
}
```

%rdi contains starting address of array

%rsi contains the index i

```
  # %rdi = arr
  # %rsi = i
movl (%rdi,%rsi,4), %eax   # z[ind]
```

Binary Puzzle

```
void mystery(int *arr) {
    ???
}
```

```
  movl $0, %rax
  jmp  .L3
.L4:
  addl $1, (%rdi,%rax,4)
  addq $1, %rax
.L3:
  cmpq $4, %rax
  jbe  .L4
  ret
```

`rdi` has the value of arr

Binary Puzzle

```
void mystery(int *arr) {
    ???
}
```

```
  movl $0, %rax
  jmp  .L3
.L4:
  addl $1, (%rdi,%rax,4)
  addq $1, %rax
.L3:
  cmpq $4, %rax
  jbe  .L4
  ret
```

```
a = 0;
goto .L3
```

`rdi` has the value of arr

Binary Puzzle

```
void mystery(int *arr) {
    ???
}
```

```
  movl $0, %rax
  jmp  .L3
.L4:
  addl $1, (%rdi,%rax,4)
  addq $1, %rax
.L3:
  cmpq $4, %rax
  jbe  .L4
  ret
```

```
  a = 0;
  goto .L3



.L3:
  if a <= 4
     goto .L4
  return
```

`rdi` has the value of arr

Binary Puzzle

```
void mystery(int *arr) {
   ???
}
```

```
  movl $0, %rax
  jmp  .L3
.L4:
  addl $1, (%rdi,%rax,4)
  addq $1, %rax
.L3:
  cmpq $4, %rax
  jbe  .L4
  ret
```

```
  a = 0;
  goto .L3
.L4
  arr[a] = arr[a] + 1
  a++
.L3:
  if a <= 4
     goto .L4
  return
```

`rdi` has the value of arr

Binary Puzzle

```
void mystery(int *arr) {
   for(int a = 0; a <= 4; a++)
   {
     arr[a] = arr[a] + 1;
   }
}
```

```
  movl $0, %rax
  jmp  .L3
.L4:
  addl $1, (%rdi,%rax,4)
  addq $1, %rax
.L3:
  cmpq $4, %rax
  jbe  .L4
  ret
```

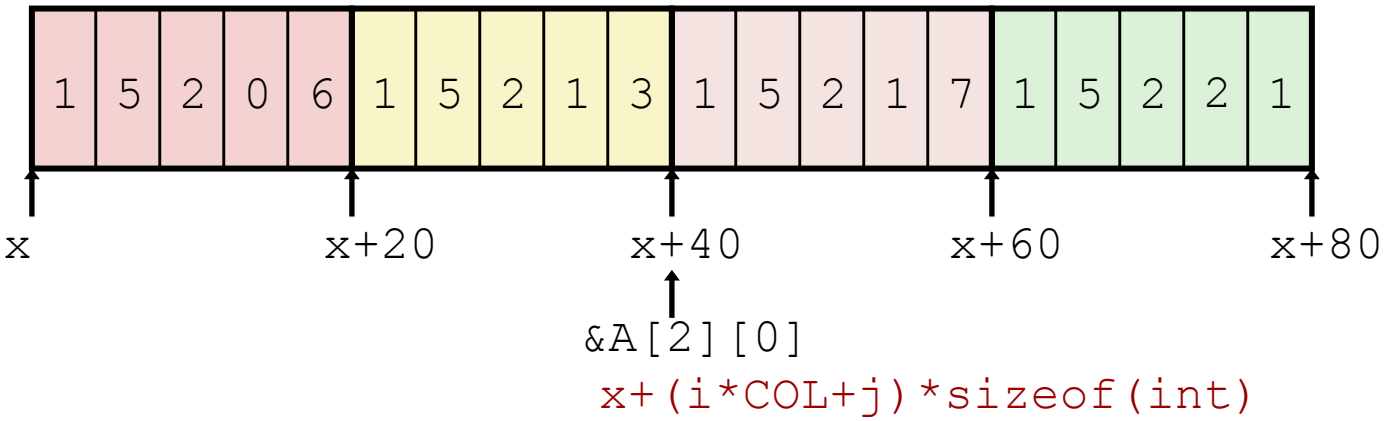
```
  a = 0;
  goto .L3
.L4
  arr[a] = arr[a] + 1
  a++
.L3:
  if a <= 4
     goto .L4
  return
```

`rdi` has the value of arr

Multi-dimensional arrays

```
#define ROW 4
#define COL 5
int A[ROW][COL] =
  {{1, 5, 2, 0, 6},
   {1, 5, 2, 1, 3 },
   {1, 5, 2, 1, 7 },
   {1, 5, 2, 2, 1 }};
```

- "Row-Major" ordering of all elements in memory



2D Array Element Access

```
int A[4][5]

int
get_digit(int i, int j)
{
  return A[i][j];
}
```

```
i:              %rdi
j:              %rsi
return value:   %rax
&A[0][0]:       0x890d0d
```

???

2D Array Element Access

```
int A[4][5]

int
get_digit(int i, int j)
{
  return A[i][j];
}
```

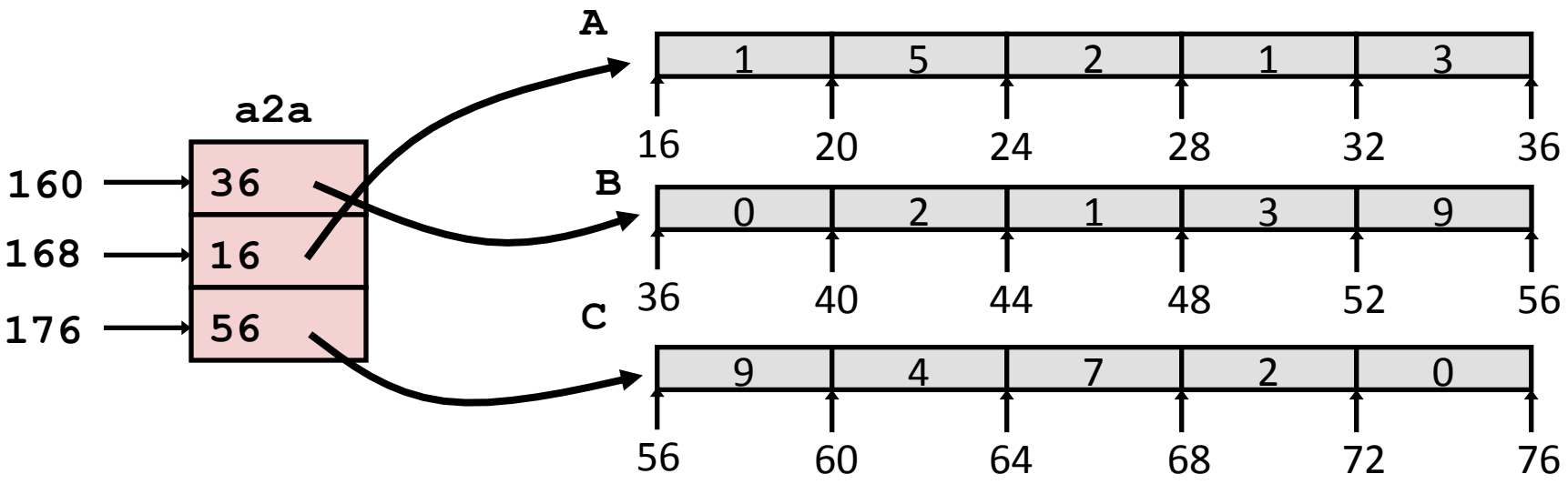
```
i:                %rdi
j:                %rsi
return value:     %rax
&A[0][0]:         0x890d0d
```

```
leaq   (%rdi,%rdi,4), %rax        # 5*i
addl   %rax, %rsi                 # 5*i+j
movl   0x890d0d(,%rsi,4), %eax    # Memory[A + 4*(5*i+j)]
```

Multi-Level Array Example

```
int A[5] = { 1, 5, 2, 1, 3 };
int B[5] = { 0, 2, 1, 3, 9 };
int C[5] = { 9, 4, 7, 2, 0 };
```

```
int *a2a[3] = {B, A, C};
```



Multi-Level Array Example

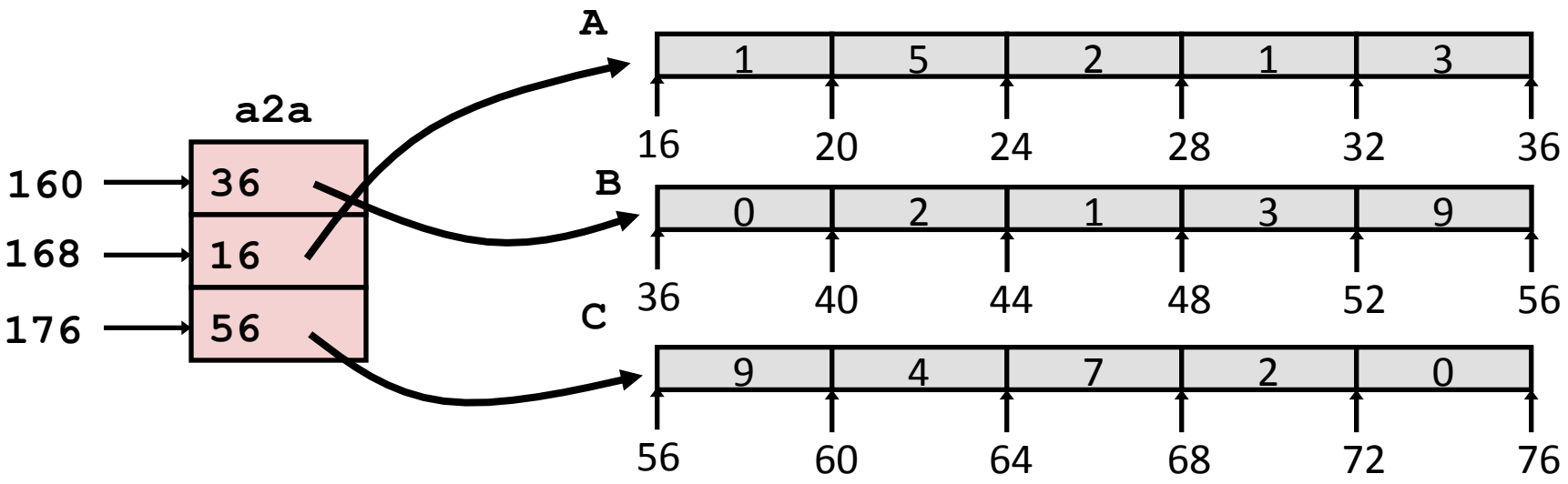
```
int A[5] = { 1, 5, 2, 1, 3 };
int B[5] = { 0, 2, 1, 3, 9 };
int C[5] = { 9, 4, 7, 2, 0 };
```

```
int *a2a[3] = {B, A, C};
```

```
int *a2a[3] = {A, B, C};

int
get_digit(long i, long j)
{
  return a2a[i][j];
}
```

a2a address is 0x8eaf
%rsi has j, %rdi has i



Multi-Level Array Example

```
int A[5] = { 1, 5, 2, 1, 3 };
int B[5] = { 0, 2, 1, 3, 9 };
int C[5] = { 9, 4, 7, 2, 0 };
```

```
int *a2a[3] = {B, A, C};
```

```
int *a2a[3] = {A, B, C};

int
get_digit(long i, long j)
{
  return a2a[i][j];
}
```

a2a address is 0x8eaf
%rsi has j, %rdi has i

```
  salq  $2, %rsi                 # 4*j
  addq  0x8eaf(,%rdi,8), %rsi    # p = a2a[i] + 4*j
  movl  (%rsi), %eax             # return *p
  ret
```

Multi-Level Array Example

```
int A[5] = { 1, 5, 2, 1, 3 };
int B[5] = { 0, 2, 1, 3, 9 };
int C[5] = { 9, 4, 7, 2, 0 };
```

```
int *a2a[3] = {B, A, C};
```

```
int *a2a[3] = {A, B, C};

int
get_digit(long i, long j)
{
  return a2a[i][j];
}
```

a2a address is 0x8eaf
%rsi has j, %rdi has i

```
  salq  $2, %rsi                  # 4*j
  addq  0x8eaf(,%rdi,8), %rsi     # p = a2a[i] + 4*j
  movl  (%rsi), %eax              # return *p
  ret
```

How does this differ from accessing a 2D array?

`Mem[A+4*(5*i+j)]`

`Mem[Mem[aofa+8*i]+4*j]`

Identify the mystery function

```
?? mystery(char *s) {

      ???
}
```

```
    mov    $0x0,%eax
    jmp    L1.
L2.
    add    $0x1,%eax
L1.
    movslq %eax,%rdx           # move sign-extended double word
    cmpb   $0x0,(%rdi,%rdx,1)
    jne    400588 L2.
    repz retq
```

Identify the mystery function

```
?? mystery(char *s) {

      ???
}
```

```
    movl    $0x0,%eax
    jmp     L1.
L2.
    addl    $0x1,%eax
L1.
    movslq %eax,%rdx
    cmpb   $0x0,(%rdi,%rdx,1)
    jne     L2.
    repz retq
```

Identify the mystery function

```
?? mystery(char *s) {

      ???
}
```

```
    movl    $0x0,%eax
    jmp     L1.
L2.
    addl    $0x1,%eax
L1.
    movslq %eax,%rdx
    cmpb   $0x0,(%rdi,%rdx,1)
    jne     L2.
    repz retq
```

```
int a = 0;
```

Identify the mystery function

```
?? mystery(char *s) {

      ???
}
```

```
    movl    $0x0,%eax
    jmp     L1.
L2.
    addl    $0x1,%eax
L1.
    movslq %eax,%rdx
    cmpb   $0x0,(%rdi,%rdx,1)
    jne     L2.
    repz retq
```

```
int a = 0;
goto L1;
```

Identify the mystery function

```
?? mystery(char *s) {

      ???
}
```

```
    movl    $0x0,%eax
    jmp     L1.
L2.
    addl    $0x1,%eax
L1.
    movslq %eax,%rdx
    cmpb   $0x0,(%rdi,%rdx,1)
    jne     L2.
    repz retq
```

```
    int a = 0;
    goto L1;


L1.
    long d = a;
```

Identify the mystery function

```
?? mystery(char *s) {

      ???
}
```

```
    movl    $0x0,%eax
    jmp     L1.
L2.
    addl    $0x1,%eax
L1.
    movslq %eax,%rdx
    cmpb   $0x0,(%rdi,%rdx,1)
    jne     L2.
    repz retq
```

```
    int a = 0;
    goto L1;


L1.
    long d = a;
    if(0 != s[d])
```

Identify the mystery function

```
?? mystery(char *s) {

      ???
}
```

```
    movl    $0x0,%eax
    jmp     L1.
L2.
    addl    $0x1,%eax
L1.
    movslq %eax,%rdx
    cmpb   $0x0,(%rdi,%rdx,1)
    jne     L2.
    repz retq
```

```
    int a = 0;
    goto L1;


L1.
    long d = a;
    if(0 != s[d]) {
       goto L2;
    }
```

Identify the mystery function

```
?? mystery(char *s) {

      ???
}
```

```
    movl    $0x0,%eax
    jmp     L1.
L2.
    addl    $0x1,%eax
L1.
    movslq %eax,%rdx
    cmpb   $0x0,(%rdi,%rdx,1)
    jne     L2.
    repz retq
```

```
    int a = 0;
    goto L1;
L2.
    a = a  + 1;
L1.
    long d = a;
    if(0 != s[d]) {
       goto L2;
    }
```

Identify the mystery function

```
int mystery(char *s) {

    int a = 0;
    long d = a;
    while(0 != s[d]) {
       a = a + 1;
       d = a;
    }
    return a;
}
```

```
        int a = 0;
        goto L1;
    L2.
        a = a  + 1;
    L1.
        long d = a;
        if(0 != s[d]) {
           goto L2;
        }
        ret;
```

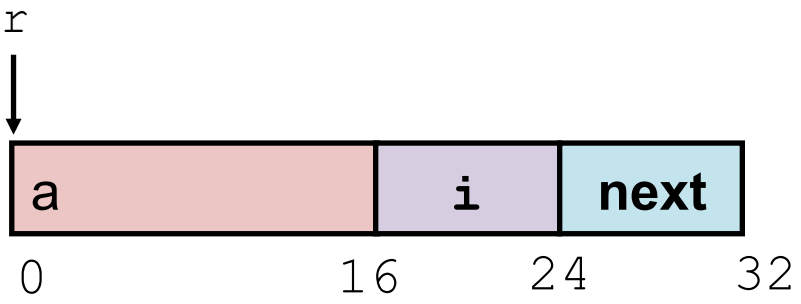
```
        movl    $0x0,%eax
        jmp     L1.
    L2.
        addl    $0x1,%eax
    L1.
        movslq %eax,%rdx
        cmpb   $0x0,(%rdi,%rdx,1)
        jne     L2.
        repz retq
```

Structure

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

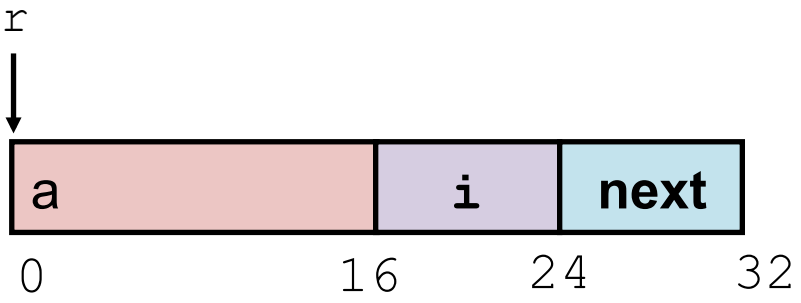
Structure

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



Structure

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




```
void func
  (struct node *r, int val)
{
   ???
}
```

Register	Value
%rdi	r
%rsi	val

```
.L11:
  movslq  16(%rdi), %rax
  movl    %esi, (%rdi,%rax,4)
  movq    24(%rdi), %rdi
  testq   %rdi, %rdi
  jne     .L11
```

Structure

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

```
void func
  (struct node *r, int val)
{
   ???
}
```

Register	Value
%rdi	r
%rsi	val

```
.L11:
  movslq  16(%rdi), %rax
  movl    %esi, (%rdi,%rax,4)
  movq    24(%rdi), %rdi
  testq   %rdi, %rdi
  jne     .L11
```

Structure

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

```
void func
  (struct node *r, int val)
{
   int a = r->i;

}
```

Register	Value
%rdi	r
%rsi	val

```
.L11:
  movslq  16(%rdi), %rax
  movl    %esi, (%rdi,%rax,4)
  movq    24(%rdi), %rdi
  testq   %rdi, %rdi
  jne     .L11
```

Structure

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

```
void func
  (struct node *r, int val)
{
   int a = r->i;
   r->a[a] = val;
}
```

Register	Value
%rdi	r
%rsi	val

```
.L11:
  movslq  16(%rdi), %rax
  movl    %esi, (%rdi,%rax,4)
  movq    24(%rdi), %rdi
  testq   %rdi, %rdi
  jne     .L11
```

Structure

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

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

Register	Value
%rdi	r
%rsi	val

```
.L11:
  movslq  16(%rdi), %rax
  movl    %esi, (%rdi,%rax,4)
  movq    24(%rdi), %rdi
  testq   %rdi, %rdi
  jne     .L11
```

Structure

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

Register	Value
%rdi	r
%rsi	val

```
void func
  (struct node *r, int val)
{
 beg:
    int a = r->i;
    r->a[a] = val;
    r = r->next;
    if (r == 0) {
      goto beg;
    }
}
```

```
.L11:
  movslq  16(%rdi), %rax
  movl    %esi, (%rdi,%rax,4)
  movq    24(%rdi), %rdi
  testq   %rdi, %rdi
  jne     .L11
```

Structure

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

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

Register	Value
%rdi	r
%rsi	val

```
.L11:
  movslq  16(%rdi), %rax
  movl    %esi, (%rdi,%rax,4)
  movq    24(%rdi), %rdi
  testq   %rdi, %rdi
  jne     .L11
```