Machine Program: Data

Jinyang Li

based on Tiger Wang's slides

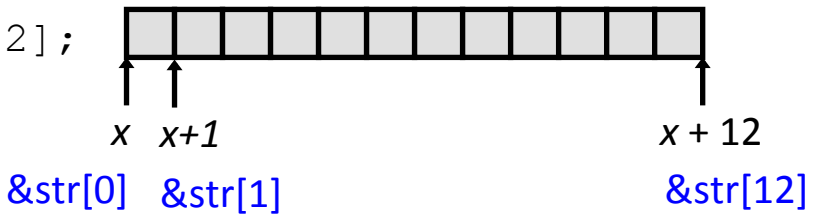
How hardware stores program data

- Variables of primitive types
  - Might correspond to registers or 1,2,4,8-byte memory.
- Arrays
  - Stored in contiguous memory
- Structures
  - Stored in contiguous memory with alignment

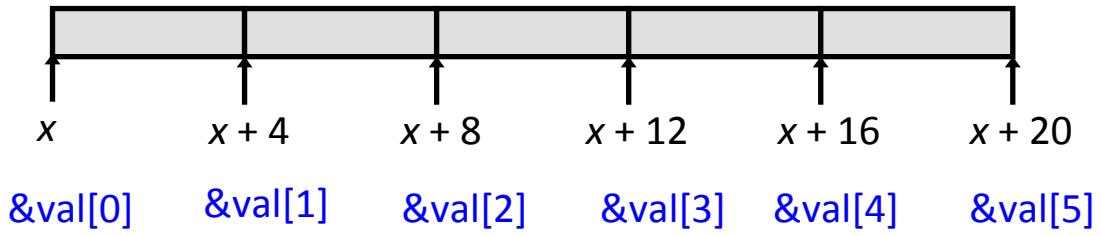
Array Allocation

- Array is stored contiguously in memory.

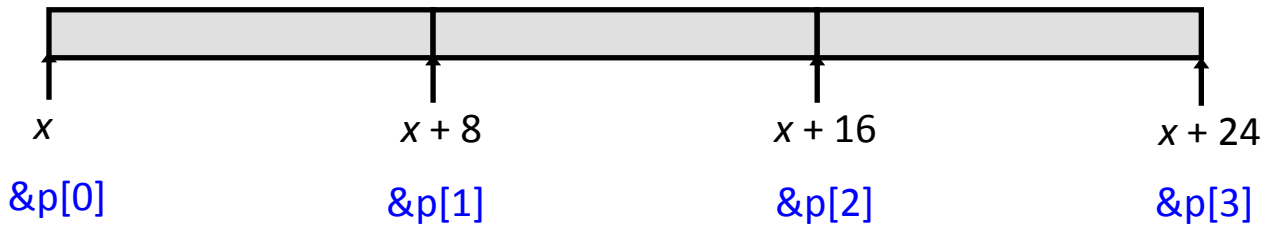
```
char str[12];
```




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

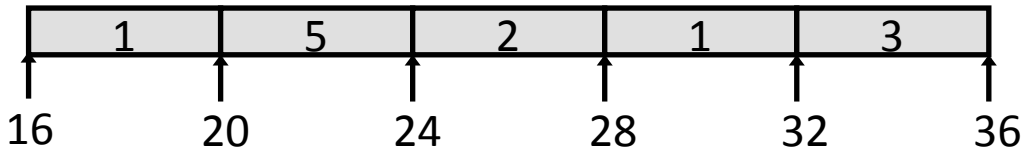



```
char *p[3];
```



Array Accessing Example

`int arr[5];`



C code

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

Suppose

`%rdi` contains starting address of array

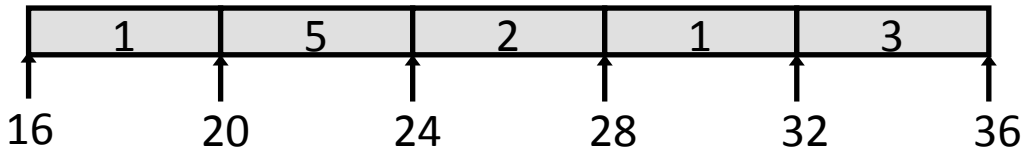
%rsi contains the index i

Assembly code

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

Array Accessing Example

```
int arr[5];
```




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

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

Binary Puzzle

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

```
  movq $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) {
    ???
}
```

```
  movq $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) {
    ???
}
```

```
  movq $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) {
    ???
}
```

```
  movq $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

What is the type of a?

Binary Puzzle

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

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

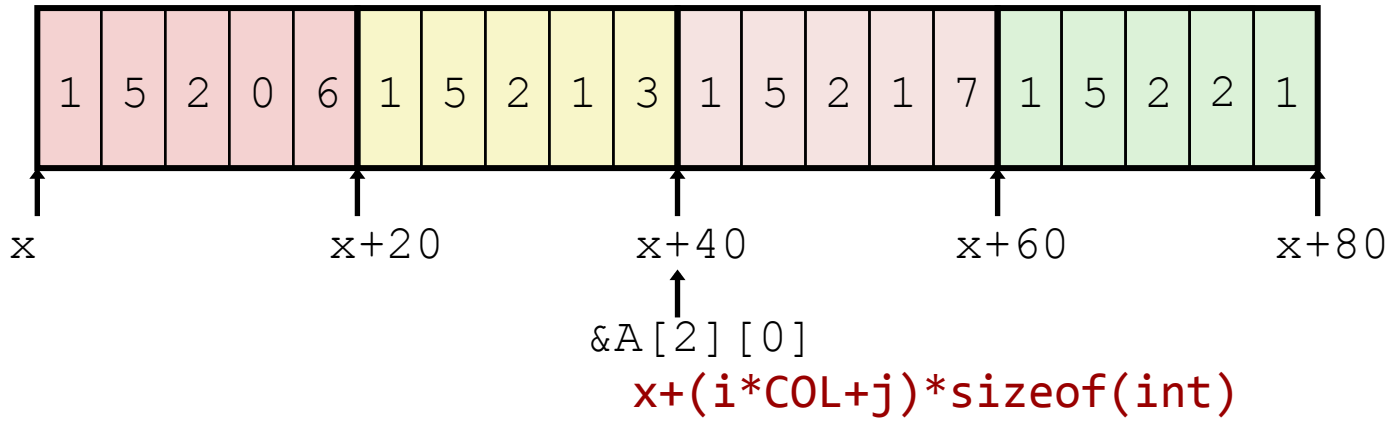
```
  a = 0;
  goto .L3
.L4
  arr[a] = arr[a] + 1
  a++
.L3:
  if a <= 4
     goto .L4
  return
```

`rdi` has the value of arr

2D 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(long long i, long long j)
{
  return A[i][j];
}
```

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

```
???
```

x+(i*5+j)*sizeof(int)

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
```

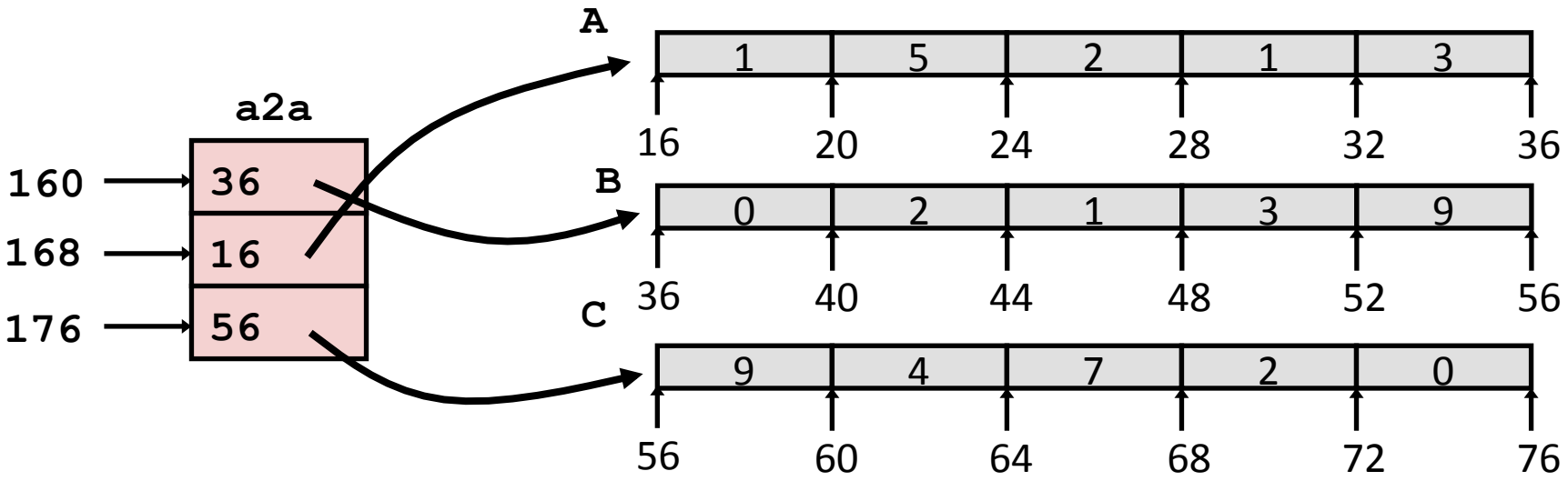
```
leaq   (%rdi,%rdi,4), %rax          # 5*i
addq   %rax, %rsi                   # 5*i+j
movl   0x890d0d(,%rsi,4), %eax      # Memory[A + 4*(5*i+j)]
```

x+(i*5+j)*sizeof(int)

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

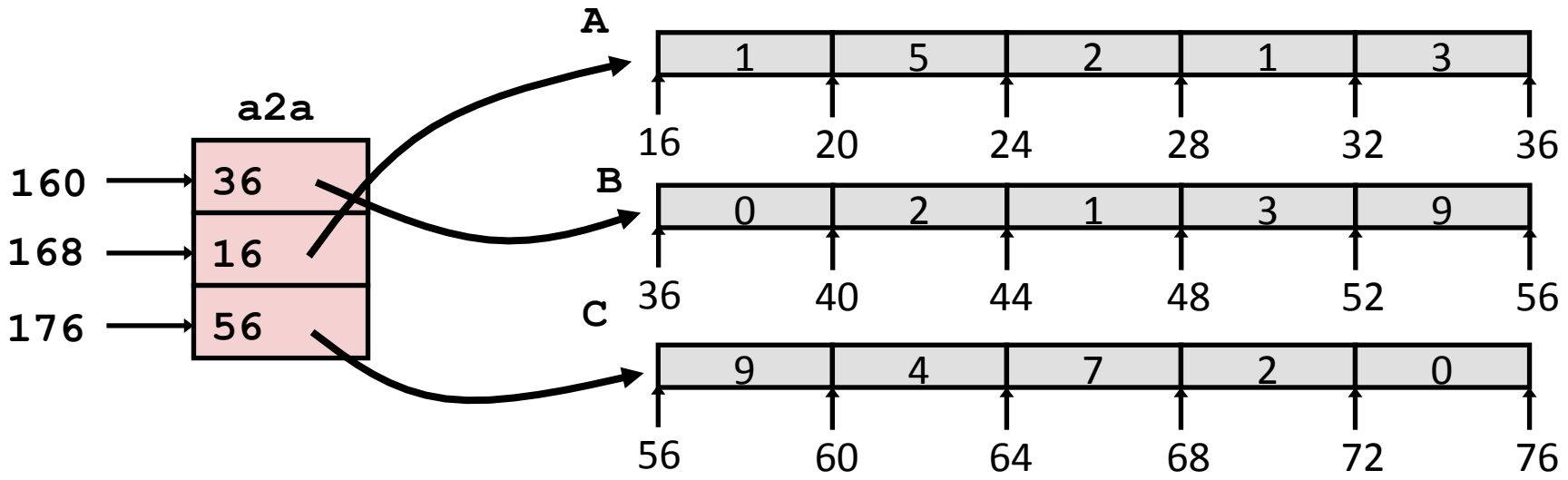
```
int A[5] = { 1, 5, 2, 1, 3 };
int B[5] = { 0, 2, 1, 3, 9 };
int C[5] = { 9, 4, 7, 2, 0 };
```

```
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) {

      ???
}
```

s is kept in %rdi

```
    movl    $0x0,%eax
    jmp     L1.
L2.
    addl    $0x1,%eax
L1.
    movslq %eax,%rdx            # move sign-extended double word
    cmpb   $0x0,(%rdi,%rdx,1)
    jne    L2.
    ret
```

Identify the mystery function

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

      ???
}
```

s is kept in %rdi

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

Identify the mystery function

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

      ???
}
```

s is kept in %rdi

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

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.
    ret
```

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

Identify the mystery function

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

      ???
}
```

s is kept in %rdi

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

```
    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.
    ret
```

```
    int a = 0;
    goto L1;


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

Identify the mystery function

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

      ???
}
```

s is kept in %rdi

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

```
    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.
    ret
```

```
    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;
}
```

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

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

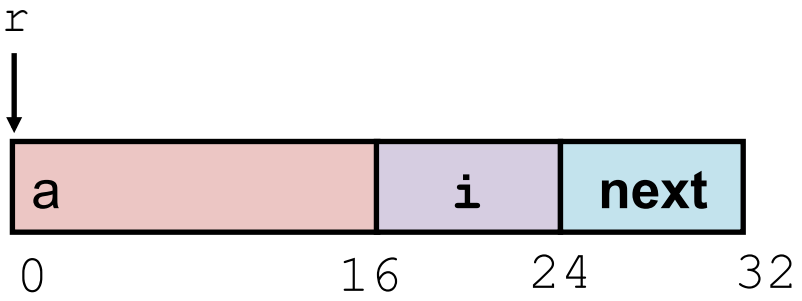
s is kept in %rdi

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;
};
```

Register	Value
%rdi	r
%rsi	val

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

Structure

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

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

Register	Value
%rdi	r
%esi	val

```
.L11:                           # loop:
  movslq  16(%rdi), %rax        #   i = M[r+16]
  movl    %esi, (%rdi,%rax,4)   #   M[r+4*i] = val
  movq    24(%rdi), %rdi        #   r = M[r+24]
  testq   %rdi, %rdi            #   test r
  jne     .L11                  #   if !=0 goto loop
```