Machine Program: Data

Jinyang Li

What we've learnt so far

- Hardware organization (CPU + memory)
- x86 instructions
  - moving data: mov
  - arithmetic: add, sub, imul, shl
  - control: EFLAGS, cmp, test, setX, jmpX

How are data stored and manipulated?

- C's primitive data type and pointer
  - char, short, int, long, long long, char *
  - stored in memory.
    - sometimes local ints or pointers are only stored in registers.

- Today's lesson plan:
  - Arrays
  - Structs

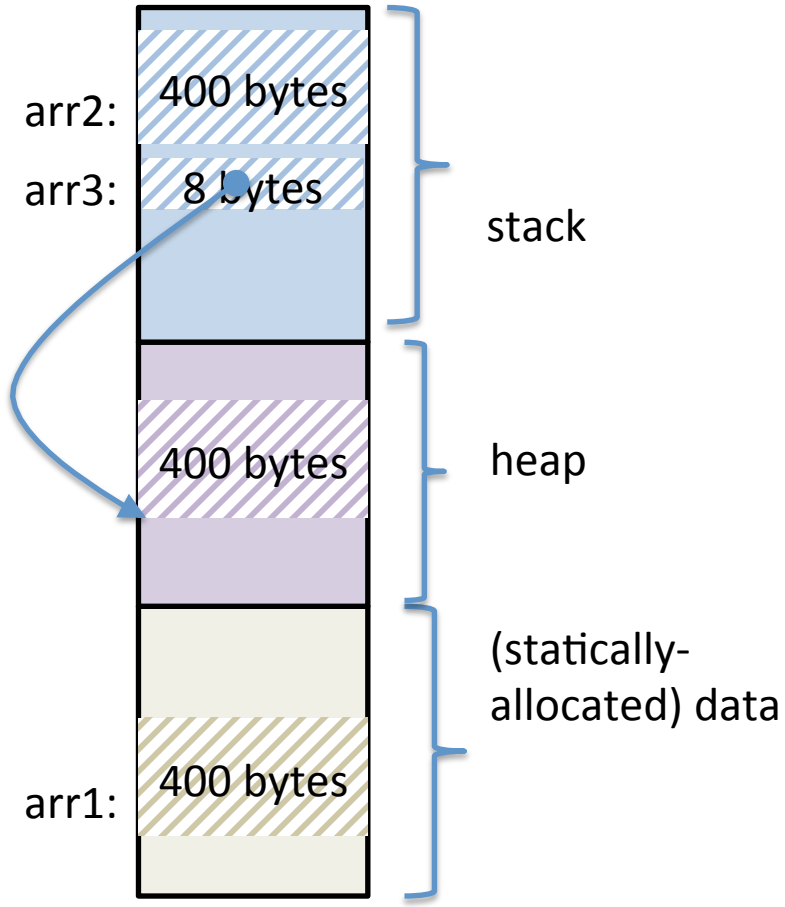
Data allocation

- Recall: a running program's memory is (conceptually) separately into 3 regions

```
int arr1[100];

void main() {

    int arr2[100];
    int *arr3;
    arr3 = malloc(sizeof(int)*100);
}
```



Data allocation

```
int arr1[100];
void main() {
   int arr2[100];
   int *arr3;
   arr3 = malloc(sizeof(int)*100);
}
```

```
(gdb) r
Starting program: /oldhome/jinyang/classes/cso/a.out

Breakpoint 1, main () at mytest.c:11
11              printf("finished\n");
(gdb) info proc map
process 30042
Mapped address spaces:

          Start Addr           End Addr       Size     Offset objfile
            0x400000           0x401000     0x1000        0x0 /oldhome/jinyang/classes/cso/a.out
            0x600000           0x601000     0x1000        0x0 /oldhome/jinyang/classes/cso/a.out
            0x601000           0x602000     0x1000     0x1000 /oldhome/jinyang/classes/cso/a.out
            0x602000           0x623000    0x21000        0x0 [heap]
      0x7ffff7a0d000     0x7ffff7bcd000   0x1c0000        0x0 /lib/x86_64-linux-gnu/libc-2.23.so
      0x7ffff7bcd000     0x7ffff7dcd000   0x200000   0x1c0000 /lib/x86_64-linux-gnu/libc-2.23.so
      0x7ffff7dcd000     0x7ffff7dd1000     0x4000   0x1c0000 /lib/x86_64-linux-gnu/libc-2.23.so
      0x7ffff7dd1000     0x7ffff7dd3000     0x2000   0x1c4000 /lib/x86_64-linux-gnu/libc-2.23.so
      0x7ffff7dd3000     0x7ffff7dd7000     0x4000        0x0
      0x7ffff7dd7000     0x7ffff7dfd000    0x26000        0x0 /lib/x86_64-linux-gnu/ld-2.23.so
      0x7ffff7fce000     0x7ffff7fd1000     0x3000        0x0
      0x7ffff7ff6000     0x7ffff7ff8000     0x2000        0x0
      0x7ffff7ff8000     0x7ffff7ffa000     0x2000        0x0 [vvar]
      0x7ffff7ffa000     0x7ffff7ffc000     0x2000        0x0 [vdso]
      0x7ffff7ffc000     0x7ffff7ffd000     0x1000    0x25000 /lib/x86_64-linux-gnu/ld-2.23.so
      0x7ffff7ffd000     0x7ffff7ffe000     0x1000    0x26000 /lib/x86_64-linux-gnu/ld-2.23.so
      0x7ffff7ffe000     0x7ffff7fff000     0x1000        0x0
      0x7ffffffde000     0x7ffffffff000    0x21000        0x0 [stack]
  0xffffffffff600000 0xffffffffff601000     0x1000        0x0 [vsyscall]
```

(gdb) p &arr1[0]
(int *) 0x601080

(gdb) p &arr2[0]
(int *) 0x7fffffffe120

Data allocation

```
int arr1[100];
void main() {
   int arr2[100];
   int *arr3;
   arr3 = malloc(sizeof(int)*100);
}
```

```
(gdb) r
Starting program: /oldhome/jinyang/classes/cso/a.out

Breakpoint 1, main () at mytest.c:11
11              printf("finished\n");
(gdb) info proc map
process 30042
Mapped address spaces:

          Start Addr           End Addr       Size     Offset objfile
            0x400000           0x401000     0x1000        0x0 /oldhome/jinyang/classes/cso/a.out
            0x600000           0x601000     0x1000        0x0 /oldhome/jinyang/classes/cso/a.out
            0x601000           0x602000     0x1000     0x1000 /oldhome/jinyang/classes/cso/a.out
            0x602000           0x623000    0x21000        0x0 [heap]
      0x7ffff7a0d000     0x7ffff7bcd000   0x1c0000        0x0 /lib/x86_64-linux-gnu/libc-2.23.so
      0x7ffff7bcd000     0x7ffff7dcd000   0x200000   0x1c0000 /lib/x86_64-linux-gnu/libc-2.23.so
      0x7ffff7dcd000     0x7ffff7dd1000     0x4000   0x1c0000 /lib/x86_64-linux-gnu/libc-2.23.so
      0x7ffff7dd1000     0x7ffff7dd3000     0x2000   0x1c4000 /lib/x86_64-linux-gnu/libc-2.23.so
      0x7ffff7dd3000     0x7ffff7dd7000     0x4000        0x0
      0x7ffff7dd7000     0x7ffff7dfd000    0x26000        0x0 /lib/x86_64-linux-gnu/ld-2.23.so
      0x7ffff7fce000     0x7ffff7fd1000     0x3000        0x0
      0x7ffff7ff6000     0x7ffff7ff8000     0x2000        0x0
      0x7ffff7ff8000     0x7ffff7ffa000     0x2000        0x0 [vvar]
      0x7ffff7ffa000     0x7ffff7ffc000     0x2000        0x0 [vdso]
      0x7ffff7ffc000     0x7ffff7ffd000     0x1000    0x25000 /lib/x86_64-linux-gnu/ld-2.23.so
      0x7ffff7ffd000     0x7ffff7ffe000     0x1000    0x26000 /lib/x86_64-linux-gnu/ld-2.23.so
      0x7ffff7ffe000     0x7ffff7fff000     0x1000        0x0
      0x7ffffffde000     0x7ffffffff000    0x21000        0x0 [stack]
  0xffffffffff600000 0xffffffffff601000     0x1000        0x0 [vsyscall]
```

(gdb) p &arr3[0]
(int *) 0x602010

(gdb) p &arr3
(int **) 0x7fffffffe118

Array Accessing Example

Suppose

%rdi contains arr

%rsi contains i

%eax is to contain arr[i]

```
int arr[5];

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

No bound checking!!

```
movl (%rdi, %rsi, 4), %eax
```

l: move 4 bytes

q: move 8 bytes

b: move 1 byte

Array Accessing Example

Suppose

%rdi contains arr

%rsi contains i

%rax is to contain arr[i]

```
char *arr[5];

char*
getpointer(char **arr, long i)
{
  return arr[i];
}
```

```
movq (%rdi, %rsi, 8), %rax
```

Binary Puzzle

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

```
  movl $0, %eax
  jmp  .L3
.L4:
  movslq %eax, %rdx
  addl $1, (%rdi,%rdx,4)
  addl $1, %eax
.L3:
  cmpl %esi, %eax
  jl  .L4
  ret
```

%rdi has the value of arr
%esi has the value of n

Binary Puzzle

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

```
  movl $0, %eax
  jmp  .L3
.L4:
  movslq %eax, %rdx
  addl $1, (%rdi,%rdx,4)
  addl $1, %eax
.L3:
  cmpl %esi, %eax
  jl  .L4
  ret
```

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

%rdi has the value of arr
%esi has the value of n

Binary Puzzle

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

```
  movl $0, %eax
  jmp  .L3
.L4:
  movslq %eax, %rdx
  addl $1, (%rdi,%rdx,4)
  addl $1, %eax
.L3:
  cmpl %esi, %eax
  jl  .L4
  ret
```

```
  a = 0;
  goto .L3



.L3:
  if a < n
     goto .L4
  return
```

%rdi has the value of arr
%esi has the value of n

Binary Puzzle

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

```
  movl $0, %eax
  jmp  .L3
.L4:
  movslq %eax, %rdx
  addl $1, (%rdi,%rdx,4)
  addl $1, %eax
.L3:
  cmpl %esi, %eax
  jl  .L4
  ret
```

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

%rdi has the value of arr
%esi has the value of n

Binary Puzzle

type of a?

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

```
  movl $0, %eax
  jmp  .L3
.L4:
  movslq %eax, %rdx
  addl $1, (%rdi,%rdx,4)
  addl $1, %eax
.L3:
  cmpl %esi, %eax
  jl  .L4
  ret
```

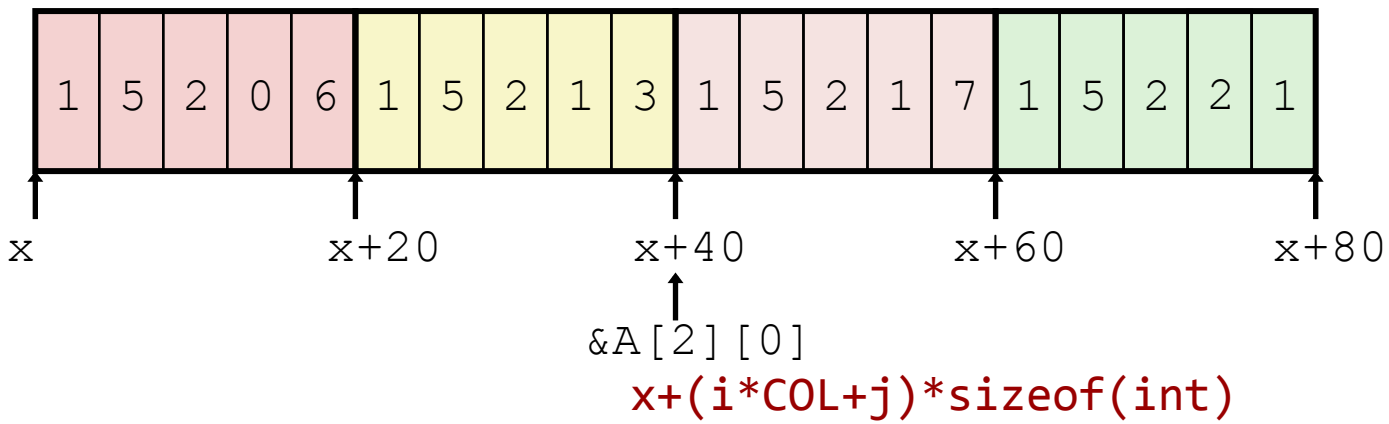
```
  a = 0;
  goto .L3
.L4
  arr[a] = arr[a] + 1
  a++
.L3:
  if a < n
     goto .L4
  return
```

%rdi has the value of arr
%esi has the value of n

2D arrays

```
int A[4][5] =
  {{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 getnum(int A[4][5], long i, long j) {
  return A[i][j];
}
```

%rdi contains A
%rsi contains i
%rdx contains j
%eax is to contain A[i]

```
leaq   (%rsi,%rsi,4), %rcx   # %rcx = 5*i
addq   %rdx, %rcx            # %rcx = 5*i+j
movl   (%rdi,%rcx,4), %eax   # %eax = *(int *)((char *)A+(5*i+j)*4)
```

```
leaq   (%rsi,%rsi,4), %rax   # %rax = 5*i
leaq   (%rdi,%rax,4), %rax   # %rax = (char *)A + 5*i*4
movl   (%rax,%rdx,4), %eax   # %eax = *(int *)(%rax+4*j)
```

Array of pointers

```
int getnum(int **A, long i, long j) {
  return A[i][j];
}
```

%rdi contains A
%rsi contains i
%rdx contains j
%eax is to contain A[i]

```
int main() {
   int a0[3] = {1, 2, 3};
   int a1[3] = {4, 5, 6};
   int *a[2] = {a0, a1};
   int n = getnum(a, 1, 2);
}
```

```
movq    (%rdi, %rsi, 8),  %rax      # %rax = *(int **)((char  *)A + i*8)
movl    (%rax, %rdx, 4),  %eax      # %eax = %rax + j*4
```

Identify the mystery function

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

      ???
}
```

%rdi contains s

```
      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) {

      ???
}
```

%rdi contains 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;
```

Identify the mystery function

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

      ???
}
```

%rdi contains 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) {

      ???
}
```

%rdi contains 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;
```

Identify the mystery function

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

      ???
}
```

%rdi contains 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) {

      ???
}
```

%rdi contains 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]) {
       goto L2;
    }
```

Identify the mystery function

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

      ???
}
```

%rdi contains 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;
}
```

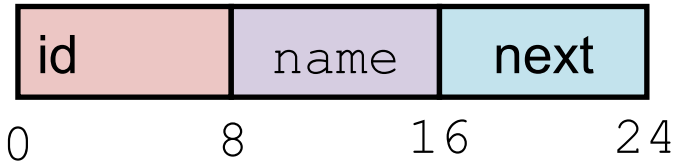
%rdi contains s

```
      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;
    }
    ret;
```

Structure

```
typedef struct node {
    long id;
    char *name;
    struct node *next;
}node;
```



Struct example

```
void init_node(node *n,
    long id, char *name)
{
   n->id = id;
   n->name = name;
   n->next = NULL;
}
```

```
int main() {
   node *n;
   n = malloc(sizeof(node));
   init_node(n, 333, “john”);
}
```

%rdi contains n
%rsi contains id
%rdx contains name

```
movq    %rsi, (%rdi)
movq    %rdx, 8(%rdi)
movq    $0, 16(%rdi)
```

Binary Puzzle

```
?? mystery(node *n, long id) {
   ???
}
```

```
  jmp     .L1
.L3:
  cmpq    %rsi, (%rdi)
  jne     .L2
  movq    8(%rdi), %rax
  ret
.L2:
  movq    16(%rdi), %rdi
.L1:
  testq   %rdi, %rdi
  jne     .L3
  movq    $0, %rax
  ret
```

%rdi has the value of n
%rsi has the value of id
%rax is to contain return value

Binary Puzzle

```
?? mystery(node *n, long id) {
   ???
}
```

```
  jmp     .L1
.L3:
  cmpq    %rsi, (%rdi)
  jne     .L2
  movq    8(%rdi), %rax
  ret
.L2:
  movq    16(%rdi), %rdi
.L1:
  testq   %rdi, %rdi
  jne     .L3
  movq    $0, %rax
  ret
```

```
goto .L1
```

%rdi has the value of n
%rsi has the value of id
%rax is to contain return value

Binary Puzzle

```
?? mystery(node *n, long id) {
   ???
}
```

```
  jmp     .L1
.L3:
  cmpq    %rsi, (%rdi)
  jne     .L2
  movq    8(%rdi), %rax
  ret
.L2:
  movq    16(%rdi), %rdi
.L1:
  testq   %rdi, %rdi
  jne     .L3
  movq    $0, %rax
  ret
```

```
  goto .L1

.L1:
  if (n != 0)
     goto .L3
```

%rdi has the value of n
%rsi has the value of id
%rax is to contain return value

Binary Puzzle

```
?? mystery(node *n, long id) {
   ???
}
```

```
  jmp     .L1
.L3:
  cmpq    %rsi, (%rdi)
  jne     .L2
  movq    8(%rdi), %rax
  ret
.L2:
  movq    16(%rdi), %rdi
.L1:
  testq   %rdi, %rdi
  jne     .L3
  movq    $0, %rax
  ret
```

```
  goto .L1

.L1:
  if (n != 0)
     goto .L3
  return 0;
```

%rdi has the value of n
%rsi has the value of id
%rax is to contain return value

Binary Puzzle

```
?? mystery(node *n, long id) {
   ???
}
```

```
  jmp     .L1
.L3:
  cmpq    %rsi, (%rdi)
  jne     .L2
  movq    8(%rdi), %rax
  ret
.L2:
  movq    16(%rdi), %rdi
.L1:
  testq   %rdi, %rdi
  jne     .L3
  movq    $0, %rax
  ret
```

```
  goto .L1
.L3:
  if (*((long *)n) != id)
     goto .L2
```

n->id != id

```
.L1:
  if (n != 0)
     goto .L3
  return 0;
```

%rdi has the value of n
%rsi has the value of id
%rax is to contain return value

Binary Puzzle

```
?? mystery(node *n, long id) {
   ???
}
```

```
  jmp     .L1
.L3:
  cmpq    %rsi, (%rdi)
  jne     .L2
  movq    8(%rdi), %rax
  ret
.L2:
  movq    16(%rdi), %rdi
.L1:
  testq   %rdi, %rdi
  jne     .L3
  movq    $0, %rax
  ret
```

```
  goto .L1;
.L3:
  if (n->id != id)
     goto .L2;

   return n->name;


.L1:
  if (n != 0)
     goto .L3;
  return 0;
```

%rdi has the value of n
%rsi has the value of id
%rax is to contain return value

Binary Puzzle

```
?? mystery(node *n, long id) {
   ???
}
```

```
  jmp      .L1
.L3:
  cmpq     %rsi, (%rdi)
  jne      .L2
  movq     8(%rdi), %rax
  ret
.L2:
  movq     16(%rdi), %rdi
.L1:
  testq    %rdi, %rdi
  jne      .L3
  movq     $0, %rax
  ret
```

```
  goto .L1;
.L3:
  if (n->id != id)
     goto .L2;

   return n->name;
.L2
   n = n->next;
.L1:
  if (n != 0)
     goto .L3;
  return 0;
```

%rdi has the value of n
%rsi has the value of id
%rax is to contain return value

Binary Puzzle

```
char *mystery(node *n, long id) {
   while (n) {
      if (n->id == id)
         return n->name;
      n= n->next;
   }
   return NULL;
}
```

```
  jmp      .L1
.L3:
  cmpq     %rsi, (%rdi)
  jne      .L2
  movq     8(%rdi), %rax
  ret
.L2:
  movq     16(%rdi), %rdi
.L1:
  testq    %rdi, %rdi
  jne      .L3
  movq     $0, %rax
  ret
```

```
  goto .L1;
.L3:
  if (n->id != id)
     goto .L2;

   return n->name;
.L2
   n = n->next;
.L1:
  if (n != 0)
     goto .L3;
  return 0;
```