CS 2630
Computer Organization

Meeting 10/11: data structures in MIPS
Brandon Myers
University of Iowa
Where we are going

Instruction set architecture (e.g., MIPS)

Compiler
- translating source code (C or Java)
- Programs to assembly language
- And linking your code to Library code

Instruction set architecture (e.g., MIPS)

How the software talks To the hardware

Processor

Memory system

I/O system

Datapath & Control

How a processor runs MIPS Programs!

Digital logic

How switches (1 or 0) can be used to build Interesting functions: from integer arithmetic to programmable computers
Memory organization of *programs*

- **Stack (0x7FFFFFFF)**: local variables, return addresses
  - RW
- **Heap (0x10040000)**: dynamically allocated memory like Java objects
  - RW
- **Static data (.data) (0x10010000)**: global data
  - initialized when process starts
  - RW
- **Instructions (.text) (0x00400000)**: assembled code
  - initialized when process starts
  - RX

Legend:
- R=readable
- W=writeable
- X=executable
Building data structures

class ListNode {
    int data;
    ListNode next;

    ListNode(int data) {
        this.data = data;
        next = null;
    }

    void append(int data) {
        if (next==null) {
            next = new ListNode(data);
        } else {
            next.append(data);
        }
    }
}

1. how do we represent objects with fields?
2. how do we represent null?
3. how do we allocate new objects?
1. Structs/objects

class ListNode {
    int data;
    ListNode next;
}

<table>
<thead>
<tr>
<th>address</th>
<th>contents</th>
<th>notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x40</td>
<td>10</td>
<td>data</td>
</tr>
<tr>
<td>0x44</td>
<td>0x58</td>
<td>next</td>
</tr>
<tr>
<td>0x48</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x4C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x50</td>
<td>3</td>
<td>data</td>
</tr>
<tr>
<td>0x54</td>
<td>0x40</td>
<td>next</td>
</tr>
<tr>
<td>0x58</td>
<td>2</td>
<td>data</td>
</tr>
<tr>
<td>0x5C</td>
<td>0x60</td>
<td>next</td>
</tr>
<tr>
<td>0x60</td>
<td>1</td>
<td>data</td>
</tr>
<tr>
<td>0x64</td>
<td>&lt;null?&gt;</td>
<td>next</td>
</tr>
</tbody>
</table>
2. null == 0

class ListNode {
    int data;
    ListNode next;
}
Building data structures

1. how do we represent objects with fields?

```java
class ListNode {
    int data;
    ListNode next;

    ListNode(int data) {
        this.data = data;
        next = null;
    }

    void append(int data) {
        if (next == null) {
            next = new ListNode(data);
        } else {
            next.append(data);
        }
    }
}
```

2. how do we represent null?

3. how do we allocate new objects?

how do we allocate new objects?
Peer instruction

given an ListNode object named L, what is the MIPS code for

L.data = L.next.data;

assume $s0 already holds the address of L;

(short response)
Building data structures

class ListNode {
    int data;
    ListNode next;

    ListNode(int data) {
        this.data = data;
        next = null;
    }

    void append(int data) {
        if (next == null) {
            next = new ListNode(data);
        } else {
            next.append(data);
        }
    }
}
Memory organization of programs

- Stack: local variables, return addresses
  RW
- Heap: dynamically allocated memory like Java objects
  RW
- Static data (.data): global data (initialized when process starts)
  RW
- Instructions (.text): assembled code (initialized when process starts)
  RX

Legend:
- R=readable
- W=writeable
- X=executable
Multiple students asked: “What happens if the Stack and Heap overlap?”

This picture doesn’t tell the whole story.

1) Our maximum stack size is actually permanently set by the OS when the program starts. We control $sp$ so the only way the OS can stop us is by checking our lw/sw addresses to make sure they are in bounds.

2) The heap is different because the OS can give us more of it dynamically (when we call sbrk). If sbrk ever returns the address 0, then we know it failed, possibly because there is no more space left. As with the stack, we can try to lw/sw to an address that doesn’t belong to us, but the OS will throw an error.
quick test of the stack behavior

jal foo

```
foo:
addiu $sp, $sp, -4
sw $ra, 0($sp)
jal foo
lw $ra, 0($sp)
addiu $sp, $sp, 4
```

recursively call foo without a base case

In MARS

```
line 6: Runtime exception at 0x00400008: address out of range 0x7fbbff
```
quick test of the heap behavior

call sbrk in an infinite loop

```
loop:
    addiu $a0, $zero, 128
    li $v0, 9  # syscall code for sbrk
    syscall
    li $t0, 0xC0FFEE
    sw $t0, 0($v0)
    j loop
```

In MARS

*line 4: Runtime exception at 0x00400008:* request (128) exceeds available heap storage (syscall 9)*
CS 2630
Computer Organization

Meeting 12: dynamic memory, strings
Brandon Myers
University of Iowa
Dynamic memory allocation

```cpp
class ListNode {
    int data;
    ListNode next;
}

next = new ListNode(data);
```

```cpp
next = sbrk(sizeof(ListNode))
    = sizeof(int) + sizeof(reference)
    = 4 + 4
    = 8
```

`sbrk (“s-break”) takes the number of bytes, and
• reserves a contiguous block of bytes in the heap
• returns the address of the first byte of that block`
In MIPS

\[
\begin{align*}
\text{addiu } & \text{a0, zero, 8} \quad \# \text{ call sbrk(8)} \\
\text{jal } & \text{sbrk} \\
\text{addu } & \text{s1, zero, v0} \quad \# \text{s1 = first address of allocated bytes}
\end{align*}
\]

• details:
  • Most C programs use a library function called malloc, which is implemented using sbrk but is fancier
  • the label sbrk is not actually defined for our MIPS programs to use; sbrk is a special operating system procedure (known as a system call), that we’ll learn about later
Putting it all together to define the constructor

```java
class ListNode {
    int data;
    ListNode next;
}

ListNode(int data) {
    this.data = data;
    next = null;
}

new ListNode:
    # ...not shown: save registers
    move $s0, $a0       # s0=data
    addiu $a0,$zero,8   # call sbrk(sizeof(ListNode))
    jal sbrk           # i.e., sbrk(8)
    addu $s1,$zero,$v0  # s1 = this
    sw $s0, 0($s1)      # this.data = data
    addu $t0,$zero,$zero
    sw $t0, 4($s1)      # this.next = null
    addu $v0,$zero,$s1  # return this
    # ...not shown: restore registers
    jr $ra

next = new ListNode(data);
...
Peer instruction

• What is the argument to sbrk if you are allocating an array of 3 DoublyListNode objects?

class DoublyListNode {
    int data;
    DoublyListNode prev;
    DoublyListNode next;
}

(numeric response)
Detail: we can’t call sbrk using jal

sbrk is a special procedure called a system call (or syscall) that is defined by the OS. Calling a syscall in MIPS is actually a bit different.

```
addiu $a0, $zero, 128   # set argument to sbrk
li $v0, 9   # set syscall code for sbrk
syscall
# when syscall returns, $v0 contains return value
```

In MARS, see help > MIPS > Syscalls for more information
Building data structures

```
class ListNode {
    int data;
    ListNode next;

    ListNode(int data) {
        this.data = data;
        next = null;
    }

    void append(int data) {
        if (next == null) {
            next = new ListNode(data);
        } else {
            next.append(data);
        }
    }
}
```

1. how do we represent objects with fields?
2. how do we represent null?
3. how do we allocate new objects?
Arguments to functions

// multiply each array element by c
void multiply_all(int[] arr,
    int num_elements,
    int c);

MIPS has four argument registers $a0, $a1, $a2, $a3.

...so how do we pass an array to a procedure?
Pass by reference

// multiply each array element by c
void multiply_all(int[] arr, int num_elements, int c);

.data
my_array: .word 3 2 4 1 2  # array of five 4-byte integers

.text
la $a0, my_array  # first argument: address of array
li $a1, 5  # second argument: num of elements
li $a2, 100  # third argument: c
jal multiply_all

...

multiply_all:  # (int[] arr, int num_elements, int c)
    addu $t0,$zero,$zero  # t0 is counter i
loop:
    blt $t0,$a1,done  # while i < num_elements
    lw $t1, 0($a0)  # t1 = arr[i]
    mul $t1,$t1,$a2  # t1 = t1 * c
    sw $t1, 0($a0)  # arr[i] = t1
    addiu $t0,$t0,1  # increment counter
    addiu $a0,$a0,4  # increment array pointer
loop:
    j loop

done:
    jr $ra
Representing human language in the computer

declare string in MIPS:

```
.data
my_greeting: .asciiz "I love CS2630!"
```

array of characters:

<table>
<thead>
<tr>
<th>'I'</th>
<th>' '</th>
<th>'l'</th>
<th>'o'</th>
<th>'v'</th>
<th>'e'</th>
<th>' '</th>
<th>'C'</th>
<th>'S'</th>
<th>'2'</th>
<th>'6'</th>
<th>'3'</th>
<th>'0'</th>
<th>'!'</th>
<th>'\0'</th>
</tr>
</thead>
</table>

ASCII-encoded 1-byte characters (shown in decimal):

| 73  | 32  | 108 | 111 | 118 | 101 | 32  | 67  | 83  | 50  | 54  | 51  | 48  | 10  | 0   |

(shown in binary):

```
01001001 00100000 01101100 01110110 01100101 00100000 01000011 01010011 00110010 00110110 00110011 00110000 01000011 01010011 00110010 00110110 00110011 00110000 00001010 00000000
```
Peer instruction

What does the function mystery do?
(initially called with $a0 as the address of an ascii-encoded, null-terminated string)

- a) reverse the substring between index 48 and 57
- b) nothing sensible because $a0 gets overwritten
- c) turn all letters to upper case
- d) turn all letters to lower case
- e) add one to only integer characters
- f) infinite recursion

you need an ascii table in front of you for this problem
System.out.println?
How do you perform I/O in MIPS?

```
.data
my_greeting: .asciiz "I love CS2630!"

.text
la $a0, my_greeting
li $v0, 4
syscall
```