CS 2630
Computer Organization
Meeting 7: data structures in assembly and more
Brandon Myers
University of Iowa
Where we are going

Instruction set architecture (e.g., MIPS)

Programs to assembly language
And linking your code to Library code

How the software talks To the hardware

Memory system

Processor

I/O system

Datapath & Control

Digital logic

translating source code (C or Java)

How a processor runs MIPS Programs!

How switches (1 or 0) can be used to build Interesting functions:
from integer arithmetic to programmable computers
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
Building data structures

```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);
        }
    }
}
```

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`

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

```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);
        }
    }
}
```

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

- Project 1 is due Sep 26, 11:59pm
Project 1

int x = arr[1];
arr[2] = x + 10;

```
lw $t0, 4($r0)
addi $t0, $t0, 10
sw $t0, 8($r0)
```

you will build this

you will build this
Project 1: Three phases to MiniMa

0. We assume the program has been parsed from a text file into an array of Instruction objects

1. Translate MAL (combo of real and pseudo instructions) into TAL (real instructions)
2. Convert logical labels into immediates (i.e., branch label turned into a relative address)
3. Translate each instruction into a 32-bit word

The main skeleton is provided, as well as test code and one test case

**your job:** complete the three phases one at a time, testing as you go
class Instruction {
    int instruction_id;
    int rd;
    int rs;
    int rt;
    int immediate;
    int jump_address
    int shift_amount;
    int label_id;
    int branch_label;
}

<table>
<thead>
<tr>
<th>Address</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x9F4</td>
<td>shift_amount</td>
</tr>
<tr>
<td>0x9F8</td>
<td>label_id</td>
</tr>
<tr>
<td>0x9FC</td>
<td>branch_label</td>
</tr>
<tr>
<td>0xA00</td>
<td>instruction_id</td>
</tr>
<tr>
<td>0xA04</td>
<td>rd</td>
</tr>
<tr>
<td>0xA08</td>
<td>rs</td>
</tr>
<tr>
<td>0xA0C</td>
<td>rt</td>
</tr>
<tr>
<td>0xA10</td>
<td>immediate</td>
</tr>
<tr>
<td>0xA14</td>
<td>jump_address</td>
</tr>
<tr>
<td>0xA18</td>
<td>shift_amount</td>
</tr>
<tr>
<td>0xA1C</td>
<td>label_id</td>
</tr>
<tr>
<td>0xA20</td>
<td>branch_label</td>
</tr>
<tr>
<td>0xA24</td>
<td>instruction_id</td>
</tr>
<tr>
<td>0xA28</td>
<td>rd</td>
</tr>
<tr>
<td>0xA2C</td>
<td>rs</td>
</tr>
<tr>
<td>0xA30</td>
<td>rt</td>
</tr>
<tr>
<td>0xA34</td>
<td>immediate</td>
</tr>
<tr>
<td>0xA38</td>
<td>jump_address</td>
</tr>
<tr>
<td>0xA3C</td>
<td>shift_amount</td>
</tr>
</tbody>
</table>
Peer instruction

Given an Instruction object named \( x \), what is the MIPS code for

\[
x.\text{rd} = x.\text{rs};
\]

Assume \( $s0 \) already holds the address of \( x \);

(short response)
Building data structures

1. how do we represent objects with fields?

2. how do we represent null?

3. how do we allocate new objects?
Memory organization of programs

Local variables, return addresses:
- Stack: RW

Dynamically allocated memory like Java objects:
- Heap: RW

Global data
- Static data (.data): RW
- (initialized when process starts)
- Assembled code
- (initialized when process starts):
  - Instructions (.text): RX
  - (other stuff)

Legend:
- R = readable
- W = writeable
- X = executable
Dynamic memory allocation

```java
class ListNode {
    int data;
    ListNode next;
}
	next = new ListNode(data);
```

```
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

```
addiu $a0,$zero,8  # call sbrk(8)
jal sbrk
addu $s1,$zero,$v0  # s1 = first address of allocated bytes
```

• 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 procedure called a system call, that we’ll learn about later
Putting it all together to define the constructor

class ListNode {
    int data;
    ListNode next;

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

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

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

class Instruction {
    int instruction_id;
    int rd;
    int rs;
    int rt;
    int immediate;
    int jump_address
    int shift_amount;
    int label_id;
    int branch_label;
}

(numeric 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);
        }
    }
}
```

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

...
Representing human language in the computer

declare string in MIPS:
```
.data
my_greeting: .asciiz "I love CS2630!"
```

array of characters:
```
'I' 'l' 'o' 'v' 'e' 'C' 'S' '2' '6' '3' '0' '!' '\0'
```

null terminator marks the end of the string

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 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
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

goto help in MARS