COMPUTER ORGANIZATION

CODE TRANSLATION I

PROF H ROUMANI
Dept. of Electrical Engineering and Computer Science, York University

CASE #1: AN APP

- A single class
  Hence, no instantiation of objects and no linking
- No attributes
  Hence no heap and no data segment
- A single method, `main`
  Hence no stack needed for method invocation
- Few integer variables
  Hence no spilling —everything fits in registers

2

I/O

- Each O/S comes with a host of syscalls. Also known as interrupts
- SPIM uses $v0 for the service number.
- See the SPIM Syscall Sheet in the course resources.

3

The MIPS Registers

2	\$0, \$ra	Reserved by hardware
10	\$t0 - \$t9	Callee-saved (temporary)
8	\$s0 - \$s7	Caller-saved (global)
4	\$a0 - \$a3	Method parameters
2	\$v0 - \$v1	Method returns
2	\$sp, \$fp	stack/frame pointers
1	\$gp	Reserved for global data
1	\$at	Reserved by assembler
2	\$k0 - \$k1	Reserved by O/S

4

Example

Java main method:

- Read `x` and `y`, both int
- Compute and output: `z + t*r + 1`, where

```
z = x + y
t = max(x,y)
r = x – y -10
```

5

General Patterns

- **The `u` suffix:**
  Stands for un-trapped for add / sub
  Stands for unsigned for mult, div, and slt
- **Handling Immediates**
  5-bit, zero-extended for shifts
  16-bit, zero-extended for logical and lui
  26-bit, sign-extended for jump
  16-bit, sign-extended for all the rest
- **Large Immediates**
  The instruction `lui`

6

The Arithmetic Family

- **add/sub**
  Why three operands? Why addi but not subi?
- **slt**
  Why do we need it? Why sltu?
- **mult**
  Why two operands? When to ignore HI?
- **div**
  Why two operands?

7

The Logical Bitwise FDamily

- **and, or, and xor**
  Is there an immediate version?
- **nor**
  *What about **not**?*
- **sll and srl**
  Is there a variable version?
- **sra**
  Why do we need it?

8

The Jump Family

- **j and jr**
  What are their algorithms? Do we need both?
- **jal and jalr**
  What are their algorithms? Do we need both?

This family is used for unconditional branching to skip the else fragment of if statements and to implement method invocation / return.

9

The Branch Family

- `beq and bne`
  What are their algorithms?
- `bltz, bltez`
  Signed implied here.

Used for conditional branching to implement if statements and loops.

10

The Load / Store Family

- `lw and sw`
  What are their algorithms?
- `lb and sb`
  What does the u suffix do to lb?

Used to transfer data to/from DRAM to implement read/write to .data, to the heap, and to the stack.

11

Case #2: A Utility Class

- A single class plus its client
  Hence, two linked classes but no objects
- Static Attributes only
  Hence .data is needed but no heap
- Can have several static methods
  Hence stack is needed

12

STATIC ATTRIBUTES IN .data

- To allocate static int x = 5 in .data:
```
x:  .word* 5
```
- To transfer the value of x to register r:
```
lb/h/w  $r, x($0)
```
- To transfer the value of register r to x:
```
sb/h/w  $r, x($0)
```

Can use byte/half/word; ascii/asciiz; or space for declaration.

13

Example #1

```
        .data
x:      .byte    65
y:      .word    123
z:      .byte    -30
u:      .half    120
v:      .ascii   "York"
s:      .asciiz  "York"
t:      .float   2.45

        .text
        lb       $t0, x($0)
        lw       $t0, y($0)
        lb       $t0, z($0)
        lbu      $t0, z($0)
        lhu      $t0, u($0)
        lbu      $t0, v($0)
```

14
HR/14

Example #2

```
        .data
y:      .word    123, 150, 22
z:      .byte    -30, 12
p:      .word    y
        .text
        addi     $t1, $0, 8
        lw       $t0, y($t1)        # index-like

        addi     $t1, $0, 1
        lb       $t0, z($t1)        # index-like

        la       $t1, y
        lw       $t0, 0($t1)        # pointer-like
        lw       $t0, 4($t1)        # pointer-like

        lw       $t1, p($0)
        addi     $t1, $t1, 8
        lw       $t0, 0($t1)        # pointer-like
```

15
HR/15

Example #3

```
        .data
x:      .byte    65
y:      .word    123, 150, 22
z:      .byte    -30, 12
u:      .half    120
v:      .ascii   “York”
s:      .asciiz  “York”
p:      .word    y

        .text
        la       $t1, x
        addi     $t1, $t1, 8
        lw       $t0, 0($t1)          # alignment!

        lw       $t1, p($0)
        addi     $t1, $t1, 16         # endianness
        lw       $t0, 0($t1)
```

16 HR/16

Stack Usage

- To push the con tent of register r on the stack:
```
sw    $r, 0($sp)
addi  $sp, $sp, -4
```
- To pop the word at the top of the stack into r:
```
addi  $sp, $sp, 4
lw    $r, 0($sp)
```
- Every method must preserve $sp, $ra, and $s? plus any other register it needs after a call.

17

Example

	O/S
16	...
20	jal main
24	...
28	...
32	...
36	...

	main
100	main:
104	...
108	jal mathod
112	...
116	...
120	jr $ra

	method
600	method:
604	...
608	...
612	...
616	...
620	jr $ra

PC	20	100	108	600	608	620	112	120	112
$ra	24	24	112	112	112	112	112	112	112

Case #3: Any Class

We need to be able to accommodate

- Non-Static Attributes
  Storage allocated on the heap
- Multi-Class Applications
  Multiple classes loaded and linked

19

Heap Usage

To allocate four bytes on the heap:

```
        .text
main:   ...

        addi    $a0, $0, 4
        addi    $v0, $0, 9
        syscall
        sw      $s0, 0($v0)      # heap store
        ...
        lw      $s0, 0($v0)      # heap load
```

20

OPP and the Heap

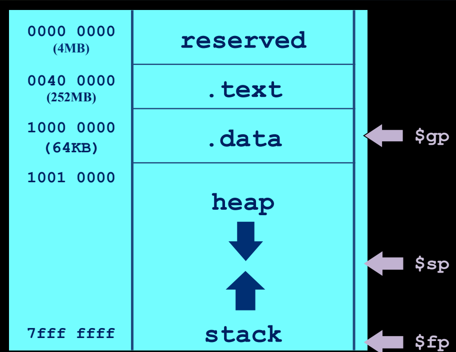
1. Class A instantiates class B via: B b = new B(…);
2. Class A invokes a method m in B via: b.m();

Class B constructor:

- Determines #of bytes needed to hold the state; i.e. the sum of the sizes of all attributes.
- Requests a block of that many bytes on the heap.
- Let x = beginning address of the returned block.
- Store all attribute values beginning at x.
- Return x

21

Memory Map



22

Data Storage Recap

- Registers
- Data Segment
- Stack Segment
- Heap Segment
- Disk / SSD

DRAM

23

Exercises

For each data storage option, determine:

- Its latency (time to retrieve one byte) in seconds
- Its typical size (in bytes)
- The lifetime of its content
- What is it used for in Java / C
- How to allocate storage in it
- How to transfer data to/from it

24