Calling Conventions

Hakim Weatherspoon
CS 3410, Spring 2013
Computer Science
Cornell University

See P&H 2.8 and 2.12
Prelim 1 results

- Mean 78.5 (median 79), standard deviation 10
- Prelims available in 305 Upson
- Regrade requires written request
  - *Whole test is regraded*
Some Survey Results

• Lecture speed
  – 59% think lecture pace is a little to way too fast
  – 28% think just about right
  – 13% variable
  – < 1% think lecture is too slow (2 people)

• Read the book
  – Over 50% never read the book at all!
  – 10% read before class

• Need to meet halfway!
  – Slow lecture down a bit, but need to read!
Some Survey Results

• Taking notes
  – 40% paper/notebook
  – 22% computer/tables
  – 7% printed copy of lecture slides
  – 29% don’t take notes at all

• Comments
  – Some people’s computer use is distracting (playing games, facebook, email, youtube, etc)
  – Peer pressure: kindly ask neighbor not to distract with computer
Goals for Today

Review: Calling Conventions
- call a routine (i.e. transfer control to procedure)
- pass arguments
  - fixed length, variable length, recursively
- return to the caller
  - Putting results in a place where caller can find them
- Manage register

Today
- More on Calling Conventions
- globals vs local accessible data
- callee vs caller saved registers
- Calling Convention examples and debugging
Goals for Today

Review: Calling Conventions

• call a routine (i.e. transfer control to procedure)
• pass arguments
  • fixed length, variable length, recursively
• return to the caller
  • Putting results in a place where caller can find them
• Manage register

Today

• More on Calling Conventions
• globals vs local accessible data
• callee vs caller saved registers
• Calling Convention examples and debugging

Warning: There is no one true MIPS calling convention. lecture != book != gcc != spim != web
Recap: Conventions so far

- **first four** arg words passed in $a0, $a1, $a2, $a3
- remaining arg words passed in **parent’s stack frame**
- return value (if any) in $v0, $v1
- stack frame at $sp
  - contains $ra (clobbered on JAL to sub-functions)
  - contains $fp
  - contains local vars (possibly clobbered by sub-functions)
  - contains extra arguments to sub-functions (i.e. argument “spilling”)
  - contains space for first 4 arguments to sub-functions
- **callee save regs** are preserved
- **caller save regs** are not
- Global data accessed via $gp

Bottom of current stack frame

<table>
<thead>
<tr>
<th>saved ra</th>
<th>saved fp</th>
<th>saved regs ($s0 ... $s7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>locals</td>
<td></td>
<td></td>
</tr>
<tr>
<td>outgoing args</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Top of the stack

$sp → $fp → $sp
## MIPS Register Conventions

<table>
<thead>
<tr>
<th>Register</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>$zero</td>
</tr>
<tr>
<td>r1</td>
<td>$at</td>
</tr>
<tr>
<td>r2</td>
<td>$v0</td>
</tr>
<tr>
<td>r3</td>
<td>$v1</td>
</tr>
<tr>
<td>r4</td>
<td>$a0</td>
</tr>
<tr>
<td>r5</td>
<td>$a1</td>
</tr>
<tr>
<td>r6</td>
<td>$a2</td>
</tr>
<tr>
<td>r7</td>
<td>$a3</td>
</tr>
<tr>
<td>r8</td>
<td>$t0</td>
</tr>
<tr>
<td>r9</td>
<td>$t1</td>
</tr>
<tr>
<td>r10</td>
<td>$t2</td>
</tr>
<tr>
<td>r11</td>
<td>$t3</td>
</tr>
<tr>
<td>r12</td>
<td>$t4</td>
</tr>
<tr>
<td>r13</td>
<td>$t5</td>
</tr>
<tr>
<td>r14</td>
<td>$t6</td>
</tr>
<tr>
<td>r15</td>
<td>$t7</td>
</tr>
<tr>
<td>r16</td>
<td>$s0</td>
</tr>
<tr>
<td>r17</td>
<td>$s1</td>
</tr>
<tr>
<td>r18</td>
<td>$s2</td>
</tr>
<tr>
<td>r19</td>
<td>$s3</td>
</tr>
<tr>
<td>r20</td>
<td>$s4</td>
</tr>
<tr>
<td>r21</td>
<td>$s5</td>
</tr>
<tr>
<td>r22</td>
<td>$s6</td>
</tr>
<tr>
<td>r23</td>
<td>$s7</td>
</tr>
<tr>
<td>r24</td>
<td>$t8</td>
</tr>
<tr>
<td>r25</td>
<td>$t9</td>
</tr>
<tr>
<td>r26</td>
<td>$k0</td>
</tr>
<tr>
<td>r27</td>
<td>$k1</td>
</tr>
<tr>
<td>r28</td>
<td>$gp</td>
</tr>
<tr>
<td>r29</td>
<td>$sp</td>
</tr>
<tr>
<td>r30</td>
<td>$fp</td>
</tr>
<tr>
<td>r31</td>
<td>$ra</td>
</tr>
</tbody>
</table>

**zero**
- assembler temp
  - function
  - return values
  - function arguments
  - temps (caller save)
- saved (callee save)
  - more temps (caller save)
- reserved for kernel
  - global data pointer
  - stack pointer
  - frame pointer
  - return address
Globals and Locals

Global variables in data segment
• Exist for all time, accessible to all routines

Dynamic variables in heap segment
• Exist between `malloc()` and `free()`

Local variables in stack frame
• Exist solely for the duration of the stack frame

Dangling pointers into freed heap mem are bad
Dangling pointers into old stack frames are bad
• C lets you create these, Java does not
• `int *foo() { int a; return &a; }`
Caller-saved vs. Callee-saved

**Caller-save:** If necessary... ($t0 .. $t9)
- save before calling anything; restore after it returns

**Callee-save:** Always... ($s0 .. $s7)
- save before modifying; restore before returning

**Caller-save registers are responsibility of the caller**
- Caller-save register values saved only if used after call/return
- The callee function can use caller-saved registers

**Callee-save register are the responsibility of the callee**
- Values must be saved by callee before they can be used
- Caller can assume that these registers will be restored

Save if want to use *after* a call
Save *before* use
Caller-saved vs. Callee-saved

Caller-save: If necessary... ($t0 .. $t9)
• save before calling anything; restore after it returns

Callee-save: Always... ($s0 .. $s7)
• save before modifying; restore before returning

MIPS ($t0-$t9), x86 (eax, ecx, and edx) are caller-save...
• ... a function can freely modify these registers
• ... but must assume that their contents have been destroyed if it in turns calls a function.

MIPS ($s0 - $s7), x86 (ebx, esi, edi, ebp, esp) are callee-save
• A function may call another function and know that the callee-save registers have not been modified
• However, if it modifies these registers itself, it must restore them to their original values before returning.
Caller-saved vs. Callee-saved

Caller-save: If necessary... ($t0 .. $t9)
  • save before calling anything; restore after it returns

Callee-save: Always... ($s0 .. $s7)
  • save before modifying; restore before returning

A caller-save register must be saved and restored around any call to a subroutine.
In contrast, for a callee-save register, a caller need do no extra work at a call site (the callee saves and restores the register if it is used).
Caller-saved vs. Callee-saved

Caller-save: If necessary... ($t0 .. $t9)
- save before calling anything; restore after it returns

Callee-save: Always... ($s0 .. $s7)
- save before modifying; restore before returning

CALLER SAVED: MIPS calls these temporary registers, $t0-t9
- the calling routine saves the registers that it does not want a called procedure to overwrite
- register values are NOT preserved across procedure calls

CALLEE SAVED: MIPS calls these saved registers, $s0-s8
- register values are preserved across procedure calls
- the called procedure saves register values in its Activation Record (AR), uses the registers for local variables, restores register values before it returns.
Caller-saved vs. Callee-saved

Caller-save: If necessary... ($t0 .. $t9)
  • save before calling anything; restore after it returns

Callee-save: Always... ($s0 .. $s7)
  • save before modifying; restore before returning

Registers $t0-$t9 are caller-saved registers
  • ... that are used to hold temporary quantities
  • ... that need not be preserved across calls

Registers $s0-s8 are callee-saved registers
  • ... that hold long-lived values
  • ... that should be preserved across calls
Activity #1: Calling Convention Example

```c
int test(int a, int b) {
    int tmp = (a&b)+(a|b);
    int s = sum(tmp,1,2,3,4,5);
    int u = sum(s,tmp,b,a,b,a);
    return u + a + b;
}
```
Activity #1: Calling Convention Example

```c
int test(int a, int b) {
    int tmp = (a&b)+(a|b);
    int s = sum(tmp,1,2,3,4,5);
    int u = sum(s,tmp,b,a,b,a);
    return u + a + b;
}
```

```
test:

Prolog

MOVE $s0, $a0
MOVE $s1, $a1
AND $t0, $a0, $a1
OR $t1, $a0, $a1
ADD $t0, $t0, $t1
MOVE $a0, $t0
LI $a1, 1
LI $a2, 2
LI $a3, 3
LI $t1, 4
SW $t1 16($sp)
LI $t1, 5
SW $t1 20($sp)
SW $t0, 24($sp)
JAL sum
NOP

# add u (v0) and a (s0)
ADD $v0, $v0, $s0
ADD $v0, $v0, $s1
# $v0 = u + a + b

Epilog
```
Activity #1: Calling Convention Example

```c
int test(int a, int b) {
    int tmp = (a&b)+(a|b);
    int s = sum(tmp,1,2,3,4,5);
    int u = sum(s,tmp,b,a,b,a);
    return u + a + b;
}
```

How many bytes do we need to allocate for the stack frame?

a) 24
b) 32
c) 40
d) **44**
e) 48
### Activity #1: Calling Convention Example

```c
int test(int a, int b) {
    int tmp = (a&b)+(a|b);
    int s = sum(tmp,1,2,3,4,5);
    int u = sum(s,tmp,b,a,b,a);
    return u + a + b;
}
```

#### Test:

**Prolog**
- MOVE $s0, $a0
- MOVE $s1, $a1
- AND $t0, $a0, $a1
- OR $t1, $a0, $a1
- ADD $t0, $t0, $t1
- MOVE $a0, $t0
- LI $a1, 1
- LI $a2, 2
- LI $a3, 3
- SW $t1, 16($sp)
- SW $t1, 20($sp)
- JAL sum

**Epilog**
- LW $t0, 24($sp)
- MOVE $a0, $v0 # s
- MOVE $a1, $t0 # tmp
- MOVE $a2, $s1 # b
- MOV $a3, $s0 # a
- SW $s1, 16($sp)
- SW $s0, 20($sp)
- JAL sum
- NOP

# add u (v0) and a (s0)
- ADD $v0, $v0, $s0
- ADD $v0, $v0, $s1
- # $v0 = u + a + b

**Saved**: $fp →
- saved ra
- saved fp
- saved regs ($s0 and $s1)
- locals ($t0)
- outgoing args space for a0 - a3 and 5th and 6th arg

**FP**: $sp
int test(int a, int b) {
    int tmp = (a&b)+(a|b);
    int s = sum(tmp,1,2,3,4,5);
    int u = sum(s,tmp,b,a,b,a);
    return u + a + b;
}

int main() {
    test(a,b);
    return 0;
}

**Activity #1: Calling Convention Example**

### Prolog

- **Saved Register $s0**: Used for return address
- **Saved Register $s1**: Used for b
- **Saved Register $s0**: Used for a
- **Local register $t0**: Used for temporary results

### Pseudocode

```plaintext
int test(int a, int b) {
    int tmp = (a&b)+(a|b);
    int s = sum(tmp,1,2,3,4,5);
    int u = sum(s,tmp,b,a,b,a);
    return u + a + b;
}
```

### Assembly Code

```assembly
int test(int a, int b) {
    int tmp = (a&b)+(a|b);
    int s = sum(tmp,1,2,3,4,5);
    int u = sum(s,tmp,b,a,b,a);
    return u + a + b;
}
```

### Stack Layout

- **$fp**: Frame Pointer
- **$sp**: Stack Pointer
- **$a0**: Function Argument
- **$a1**: Function Argument
- **$a2**: Function Argument
- **$a3**: Function Argument
- **$t0**: Temporary Register
- **$t1**: Temporary Register
- **$t2**: Temporary Register
- **$t3**: Temporary Register
- **$t4**: Temporary Register
- **$t5**: Temporary Register
- **$s0**: Local Register
- **$s1**: Local Register

### Code

```assembly
int test(int a, int b) {
    int tmp = (a&b)+(a|b);
    int s = sum(tmp,1,2,3,4,5);
    int u = sum(s,tmp,b,a,b,a);
    return u + a + b;
}
```
Activity #2: Calling Convention Example:
Prolog, Epilog

test:

# allocate frame
# save $ra
# save old $fp
# callee save ...
# callee save ...
# set new frame ptr...
# restore ...
# restore ...
# restore old $fp
# restore $ra
# dealloc frame

$fp →

| 40 | saved ra |
| 36 | saved fp |
| 32 | saved reg $s1 |
| 28 | saved reg $s0 |
| 24 | local $t0 |
| 20 | outgoing 6th arg |
| 16 | outgoing 5th arg |
| 12 | space for $a3 |
|  8 | space for $a2 |
|  4 | space for $a1 |
|  0 | space for $a0 |

$sp →
Activity #2: Calling Convention Example:

Prolog, Epilog

Space for $s0 and six args to pass to subroutine

<table>
<thead>
<tr>
<th>$sp</th>
<th>$fp</th>
<th>saved ra</th>
<th>saved fp</th>
<th>saved reg $s1</th>
<th>saved reg $s0</th>
<th>local $t0</th>
<th>outgoing 6th arg</th>
<th>outgoing 5th arg</th>
<th>space for $a3</th>
<th>space for $a2</th>
<th>space for $a1</th>
<th>space for $a0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>40</td>
<td></td>
<td></td>
<td>saved ra</td>
<td>saved fp</td>
<td>local $t0</td>
<td>outgoing 6th arg</td>
<td>outgoing 5th arg</td>
<td>space for $a3</td>
<td>space for $a2</td>
<td>space for $a1</td>
<td>space for $a0</td>
</tr>
</tbody>
</table>

Body (previous slide, Activity #1)

ADDIU $sp, $sp, -44
SW $ra, 40($sp)
SW $fp, 36($sp)
SW $s1, 32($sp)
SW $s0, 28($sp)
ADDIU $fp, $sp, 40
LW $s0, 28($sp)
LW $s1, 32($sp)
LW $fp, 36($sp)
LW $ra, 40($sp)
ADDIU $sp, $sp, 44
JR $ra
NOP

# allocate frame
# save $ra
# save old $fp
# callee save ...
# callee save ...
# set new frame ptr

... ...

# restore ...
# restore ...
# restore old $fp
# restore $ra
# dealloc frame
Next Goal

Can we optimize the assembly code at all?
Activity #3: Calling Convention Example

```c
int test(int a, int b) {
    int tmp = (a&b)+(a|b);
    int s = sum(tmp,1,2,3,4,5);
    int u = sum(s,tmp,b,a,b,a);
    return u + a + b;
}
```

How can we optimize the assembly code?

```assembly
# Prolog
LW $t0, 24($sp)
MOVE $a0, $v0 # s
MOVE $a1, $t0 # tmp
MOVE $a2, $s1 # b
MOVE $a3, $s0 # a
SW $s1, 16($sp)
SW $s0, 20($sp)
JAL sum
NOP

# add u (v0) and a (s0)
ADD $v0, $v0, $s0
ADD $v0, $v0, $s1
# $v0 = u + a + b
SW $t0, 24($sp)
JAL sum
NOP

Epilog
```
Activity #3: Calling Convention Example:
Prolog, Epilog

```assembly
test:
    ADDIU $sp, $sp, -44          # allocate frame
    SW $ra, 40($sp)              # save $ra
    SW $fp, 36($sp)              # save old $fp
    SW $s1, 32($sp)              # callee save ...
    SW $s0, 28($sp)              # callee save ...
    ADDIU $fp, $sp, 40           # set new frame ptr
                              
Body

    LW $s0, 28($sp)              # restore ...
    LW $s1, 32($sp)              # restore ...
    LW $fp, 36($sp)              # restore ...
    LW $ra, 40($sp)              # restore old $fp
    ADDIU $sp, $sp, 44           # restore $ra
    JR $ra
    NOP
```

# dealloc frame
Minimum stack size for a standard function?
Minimum stack size for a standard function?

24 bytes = 6x 4 bytes ($ra + $fp + 4 args)

$fp →
- saved ra
- saved fp
- saved regs ($s0 ... $s7)
- locals
- outgoing args

$sp →
Leaf Functions

*Leaf function* does not invoke any other functions

```c
int f(int x, int y) { return (x+y); }
```

**Optimizations?**

- No saved regs (or locals)
- No outgoing args
- Don’t push $ra
- No frame at all?
  - Maybe.
Upcoming agenda

- Schedule PA2 Design Doc Mtg for next Monday, Mar 11\textsuperscript{th}
- HW3 due next Wednesday, March 13\textsuperscript{th}
- PA2 Work-in-Progress circuit due before spring break

- Spring break: Saturday, March 16\textsuperscript{th} to Sunday, March 24\textsuperscript{th}

- Prelim2 Thursday, March 28\textsuperscript{th}, right after spring break
- PA2 due Thursday, April 4\textsuperscript{th}
Recap

• How to write and Debug a MIPS program using calling convention
  • first four arg words passed in $a0, $a1, $a2, $a3
  • remaining arg words passed in parent’s stack frame
  • return value (if any) in $v0, $v1
• stack frame at $sp
  – contains $ra (clobbered on JAL to sub-functions)
  – contains $fp
  – contains local vars (possibly clobbered by sub-functions)
  – contains extra arguments to sub-functions (i.e. argument “spilling”)
  – contains space for first 4 arguments to sub-functions
• callee save regs are preserved
• caller save regs are not
• Global data accessed via $gp

\[ \text{fp} \rightarrow \]

<table>
<thead>
<tr>
<th></th>
<th>saved ra</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>saved fp</td>
</tr>
<tr>
<td>saved regs</td>
<td>($s0 \ldots \ $s7)</td>
</tr>
<tr>
<td></td>
<td>locals</td>
</tr>
<tr>
<td></td>
<td>outgoing args</td>
</tr>
</tbody>
</table>

\[ \text{sp} \rightarrow \]