CS61C Guerrilla Section 2: MIPS and CALL

Led by Peijie, Connor and Sandy

June 30, 2016
Overview
1. MIPS Review
2. CALL

WELCOME TO THE WORLD OF MIPS!

Topics for Today
Overview
1. MIPS Review
2. CALL

WELCOME TO THE WORLD OF MIPS!

Topics for Today

» 1. MIPS Review
» 2. CALL
MIPS Review - Registers

- 32 registers, 32 bits wide, holds a word
MIPS Review - Registers

- 32 registers, 32 bits wide, holds a word
- Operands used by instructions
MIPS Review - Registers

- 32 registers, 32 bits wide, holds a word
- Operands used by instructions
- No data types, just raw bits.
MIPS Review - Registers

- 32 registers, 32 bits wide, holds a word
- Operands used by instructions
- No data types, just raw bits.
- EXTREMELY FAST
MIPS Review - Data Transfer

memop reg, off(bAddr)
MIPS Review - Data Transfer

\[ \text{memop reg, off(bAddr)} \]

- \text{memop} = operation name ("operator")
- \text{reg} = register for operation source or destination
- \text{bAddr} = register with pointer to memory ("base address")
- \text{off} = address offset (immediate) in bytes ("offset")
Overview
1. MIPS Review
2. CALL

WELCOME TO THE WORLD OF MIPS!

MIPS Review - Data Transfer

\[ \text{memop reg, off(bAddr)} \]

- \text{memop} = operation name ("operator")
- \text{reg} = register for operation source or destination
- \text{bAddr} = register with pointer to memory ("base address")
- \text{off} = address offset (immediate) in bytes ("offset")
- Accesses memory address \( b\text{Address} + \text{offset} \)
MIPS Review - Data Transfer

\[ \text{memop reg, off(bAddr)} \]

- \text{memop} = operation name ("operator")
- \text{reg} = register for operation source or destination
- \text{bAddr} = register with pointer to memory ("base address")
- \text{off} = address offset (immediate) in bytes ("offset")
- Accesses memory address \text{bAddress} + \text{offset}

- \text{lw} \ $t0, 12($s3)
- \text{sw} \ $t1, 40($a4)
MIPS Review - Instructions

- Arithmetic: add, sub, addi, muly, addu, subu, addiu
- Data Transfer: lw, sw, lb, sb
- Branching: beq, bne, slt, slti
- Bitwise: and, andi, or, ori, nor, xor, xori
- Shifting: sll, srl, sra
- Pseudo-Instruction: move, la, li
MIPS Review - Function Calls
MIPS Review - Function Calls

1. Put arguments in *arguments registers* $a^*$
2. Transfer control to the function: *jal*
3. The function acquires any storage resources it needs.
4. The function performs its desired task
5. The function returns and "cleans up": $v^*$
6. Control is returned to you: *jr $ra*
Who needs to store their registers in the stack? And where to save them?

<table>
<thead>
<tr>
<th>REGISTER NAME, NUMBER, USE, CALL CONVENTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
</tr>
<tr>
<td>$zero</td>
</tr>
<tr>
<td>$at</td>
</tr>
<tr>
<td>$v0-$v1</td>
</tr>
<tr>
<td>$a0-$a3</td>
</tr>
<tr>
<td>$t0-$t7</td>
</tr>
<tr>
<td>$s0-$s7</td>
</tr>
<tr>
<td>$t8-$t9</td>
</tr>
<tr>
<td>$k0-$k1</td>
</tr>
<tr>
<td>$gp</td>
</tr>
<tr>
<td>$sp</td>
</tr>
<tr>
<td>$fp</td>
</tr>
<tr>
<td>$ra</td>
</tr>
</tbody>
</table>
MIPS Review - Calling Conventions

Who needs to store their registers in the stack? And where to save them?

- $t^*, \$v^*, \$a^*$ — Caller (the calling function)
- $s^*, \$sp, \$ra$ — Callee (the function being called)

<table>
<thead>
<tr>
<th>REGISTERS</th>
<th>NUMBER</th>
<th>USE</th>
<th>PRESERVED ACROSS A CALL?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$zero$</td>
<td>0</td>
<td>The Constant Value 0</td>
<td>N.A.</td>
</tr>
<tr>
<td>$at$</td>
<td>1</td>
<td>Assembler Temporary</td>
<td>No</td>
</tr>
<tr>
<td>$v0-v1$</td>
<td>2-3</td>
<td>Values for Function Results and Expression Evaluation</td>
<td>No</td>
</tr>
<tr>
<td>$a0-a3$</td>
<td>4-7</td>
<td>Arguments</td>
<td>No</td>
</tr>
<tr>
<td>$t0-t7$</td>
<td>8-15</td>
<td>Temporaries</td>
<td>No</td>
</tr>
<tr>
<td>$s0-s7$</td>
<td>16-23</td>
<td>Saved Temporaries</td>
<td>Yes</td>
</tr>
<tr>
<td>$t8-t9$</td>
<td>24-25</td>
<td>Temporaries</td>
<td>No</td>
</tr>
<tr>
<td>$k0-k1$</td>
<td>26-27</td>
<td>Reserved for OS Kernel</td>
<td>No</td>
</tr>
<tr>
<td>$gp$</td>
<td>28</td>
<td>Global Pointer</td>
<td>Yes</td>
</tr>
<tr>
<td>$sp$</td>
<td>29</td>
<td>Stack Pointer</td>
<td>Yes</td>
</tr>
<tr>
<td>$fp$</td>
<td>30</td>
<td>Frame Pointer</td>
<td>Yes</td>
</tr>
<tr>
<td>$ra$</td>
<td>31</td>
<td>Return Address</td>
<td>No</td>
</tr>
</tbody>
</table>
Who needs to store their registers in the stack? And where to save them?

- $t^*, \$v^*, \$a^*$ — Caller (the calling function)
- $s^*, \$sp, \$ra$ — Callee (the function being called)
- $\$sp$: pointer to last used space of stack.

<table>
<thead>
<tr>
<th>REGISTER NAME, NUMBER, USE, CALL CONVENTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>$$zero</td>
</tr>
<tr>
<td>Sat</td>
</tr>
<tr>
<td>$v0$-$v1</td>
</tr>
<tr>
<td>$a0$-$a3</td>
</tr>
<tr>
<td>$t0$-$t7</td>
</tr>
<tr>
<td>$s0$-$s7</td>
</tr>
<tr>
<td>$t8$-$t9</td>
</tr>
<tr>
<td>$k0$-$k1</td>
</tr>
<tr>
<td>$gp$</td>
</tr>
<tr>
<td>$sp$</td>
</tr>
<tr>
<td>$fp$</td>
</tr>
<tr>
<td>$ra$</td>
</tr>
</tbody>
</table>
Overview
1. MIPS Review
2. CALL

WELCOME TO THE WORLD OF MIPS!

MIPS Review - Collecting the pieces

Basic Structure of a Function

Prologue

```assembly
func_label:
addi $sp, $sp, -framesize
sw $ra, <framesize-4>($sp)
save other regs if need be
```

Body

(call other functions...)

Epilogue

```assembly
restore other regs if need be
lw $ra, <framesize-4>($sp)
addi $sp, $sp, framesize
jr $ra
```
Overview
1. MIPS Review
2. CALL

WELCOME TO THE WORLD OF MIPS!

MIPS Review - Instruction Formats

**BASIC INSTRUCTION FORMATS**

<table>
<thead>
<tr>
<th>R</th>
<th>opcode</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>31</td>
<td>26</td>
<td>25</td>
<td>21</td>
<td>20</td>
<td>16</td>
</tr>
<tr>
<td>I</td>
<td>opcode</td>
<td>rs</td>
<td>rt</td>
<td></td>
<td>immediate</td>
<td></td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>26</td>
<td>25</td>
<td>21</td>
<td>20</td>
<td>16</td>
</tr>
<tr>
<td>J</td>
<td>opcode</td>
<td></td>
<td></td>
<td></td>
<td>address</td>
<td></td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>26</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Overview
1. MIPS Review
2. CALL
WELCOME TO THE WORLD OF MIPS!

MIPS Review - Instruction Formats

<table>
<thead>
<tr>
<th>BASIC INSTRUCTION FORMATS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>R</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>I</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>J</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

- J-Format: j and jal
- R-Format: all other instructions (including jr)
Overview
1. MIPS Review
2. CALL

WELCOME TO THE WORLD OF MIPS!

MIPS Review - Instruction Formats

<table>
<thead>
<tr>
<th>BASIC INSTRUCTION FORMATS</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>I</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>J</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

- J-Format: j and jal
- R-Format: all other instructions (including jr)
- e.g. add rd, rs, rt
MIPS Review - Jumps and Branches

Overview
1. MIPS Review
2. CALL

WELCOME TO THE WORLD OF MIPS!

MIPS Review - Jumps and Branches

- **J**: Pseudo-direct Addressing:
  \[ PC = \{ PC + 4 \} (31:28) + \text{target address} \ll 2 \]

- **Jr**: Register Addressing
  \[ PC = \text{full 32 bit address stored in src register} \]

- **beq/bne**: PC-Relative Addressing
  \[ PC = PC + 4 + \text{imm} \ll 2 \]

- **lw/lb/sw/sb**: Base Addressing
  \[ \text{Mem}[\text{register}] + \text{immediate} \]
MIPS Review - Jumps and Branches

- J: Pseudo-direct Addressing:
  - → PC = \{PC+4\}(31:28) + target address \ll 2
MIPS Review - Jumps and Branches

- J: Pseudo-direct Addressing:
  - $PC = \{PC+4\}(31:28) + \text{target address} \ll 2$

- Jr: Register Addressing
  - $PC = \text{full 32 bit address stored in src register}$
Overview
1. MIPS Review
2. CALL

WELCOME TO THE WORLD OF MIPS!

MIPS Review - Jumps and Branches

▷ J: Pseudo-direct Addressing:
▷ ——> PC = \{(PC+4)(31:28) + target address \ll 2
▷ Jr: Register Addressing
▷ ——> PC = full 32 bit address stored in src register
▷ beq/bne: PC-Relative Addressing
▷ ——> PC = PC +4+ imm \ll 2
MIPS Review - Jumps and Branches

- **J**: Pseudo-direct Addressing:
  - \( PC = (PC + 4)(31:28) + \text{target address} \ll 2 \)

- **Jr**: Register Addressing
  - \( PC = \text{full 32 bit address stored in src register} \)

- **beq/bne**: PC-Relative Addressing
  - \( PC = PC + 4 + \text{imm} \ll 2 \)

- **lw/lb/sw/sb**: Base Addressing
  - \( \text{Mem}[\text{register}] + \text{immediate} \)
Overview
1. MIPS Review
2. CALL
WELCOME TO THE WORLD OF MIPS!

MIPS Review - Pseudo-direct Addressing

Opcode | Offset
-------|-------

XXX | 00

Program Counter
MIPS Review - Jump/Branch: Range?
MIPS Review - Jump/Branch: Range?

- J: Pseudo-direct Addressing:
  - $2^{26}$ instructions;
  - $2^{26} \times 2 = 2^{28}$ bytes away
MIPS Review - Jump/Branch: Range?

- J: Pseudo-direct Addressing:
  - $2^{26}$ instructions;
  - $2^{26} \ll 2 = 2^{28}$ bytes away

- Jr: Register Addressing
  - $2^{32}$ instructions;
MIPS Review - Jump/Branch: Range?

- **J**: Pseudo-direct Addressing:
  - $2^{26}$ instructions;
  - Maximum $2^{26} \ll 2 = 2^{28}$ bytes away
- **Jr**: Register Addressing
  - $2^{32}$ instructions;
- **beq/bne**: PC-Relative Addressing
  - $2^{16}$ instructions;
  - Maximum $2^{17}$ bytes away in either direction
CALL - Translation

- C: produces MAL code

Steps to Starting a Program (translation):

1. C program: foo.c
2. Compiler
3. Assembly program: foo.s
4. Assembler
5. Object (mach lang module): foo.o
6. Linker
7. Executable (mach lang pgm): a.out
8. Loader
9. Memory
CALL - Translation

- C: produces MAL code
- A: 2 Passes, removes pseudo-instructions, produce TAL, Symbol Table and Relocation Table

Steps to Starting a Program (translation)

1. **C program:** `foo.c`
2. **Assembly program:** `foo.s`
3. **Object (mach lang module):** `foo.o`
4. **Executable (mach lang pgm):** `a.out`
5. **Libs:** `lib.o`

---

Led by Peijie, Connor and Sandy

CS61C Guerrilla Section 2: MIPS and CALL
CALL - Translation

- C: produces MAL code
- A: 2 Passes, removes pseudo-instructions, produce TAL, Symbol Table and Relocation Table
- L: combines object files and resolve absolute addresses

Steps to Starting a Program (translation):

1. C program: foo.c
2. Assembly program: foo.s
3. Object(mach lang module): foo.o
4. Executable(mach lang pgm): a.out
5. Lib: lib.o
6. Loader
7. Memory
CALL - Translation

- C: produces MAL code
- A: 2 Passes, removes pseudo-instructions, produce TAL, Symbol Table and Relocation Table
- L: combines object files and resolve absolute addresses
- L: loads executable and run program

Steps to Starting a Program (translation):
- C program: foo.c
- Assembly program: foo.s
- Object (mach lang module): foo.o
- Executable (mach lang pgm): a.out

Led by Peijie, Connor and Sandy
CS61C Guerrilla Section 2: MIPS and CALL
WELCOME TO THE WORLD OF MIPS!