MIPS Introduction

Philipp Koehn

17 October 2016
History

- Developed by MIPS Technologies in 1984, first product in 1986

- Used in
  - Silicon Graphics (SGI) Unix workstations
  - Digital Equipment Corporation (DEC) Unix workstation
  - Nintendo 64
  - Sony PlayStation

- Inspiration for ARM (esp. v8)
Overview

• 32 bit architecture (registers, memory addresses)

• 32 registers

• Similar types of instructions to 6502

• Multiply and divide instructions

• Floating point numbers
Example: Addition

- Mathematical view of addition
  
  \[ a = b + c \]

- MIPS instruction
  
  `add a, b, c`

  `a, b, c` are registers
32 Registers

• Some are special
  0 $zero always has the value 0
  31 $ra contains return address

• Some have usage conventions
  1 $at reserved for pseudo-instructions
  2-3 $v0-$v1 return values of a function call
  4-7 $a0-$a3 arguments for a function call
  8-15, 24, 25 $t0-$t9 temporaries, can be overwritten by function
  16-23 $s0-$s7 saved, have to be preserved by function
  26-27 $k0-$k1 reserved for kernel
  28 $gp global area pointer
  29 $sp stack pointer
  30 $fp frame pointer
## Endianness

- How are 16 bit numbers like 1234\text{hex} stored in memory?

<table>
<thead>
<tr>
<th>Address</th>
<th>Little Endian</th>
<th>Big Endian</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>34</td>
<td>12</td>
</tr>
<tr>
<td>0001</td>
<td>12</td>
<td>34</td>
</tr>
</tbody>
</table>

- From Jonathan Swift’s "Gulliver’s Travels" (1726):
  War over how to crack an egg:

  ![Big Endian
  People’s tradition](image)
  ![Little Endian
  King’s order](image)

- Little Endian: 6502, x86
- Big Endian: MIPS, Internet transfer protocols
instruction formats
Instruction Format (R Type)

- All instructions are encoded in 4 bytes --- 32 bits

- Instruction format (register type)
  - 6 bits: op: operation code
  - 5 bits: rs: first source operand register
  - 5 bits: rt: second source operand register
  - 5 bits: rd: return operand register
  - 5 bits: shamt: shift amount (for shift instructions)
  - 6 bits: funct: function code, indicates variant of operation

- Examples
  - add: operation code 0, function code 32
  - sub: operation code 0, function code 34
Instruction Format (I Type)

- Some operations may directly use 16 bit values

- Example: addi $s1, $s2, 100
  (adds value of register $s2 and 100, stores result in register $s1)

- Instruction format (immediate type)
  - 6 bits: op: operation code
  - 5 bits: rd: return operand register
  - 5 bits: rs: source operand register
  - 16 bits: constant or address
32 Bit Values

• All instructions are encoded in 32 bits
• Registers can hold 32 bit values
• How can we load 32 bit values into a register?

⇒ Solution: 2 instructions

• First load upper order 16 bits (load upper immediate)
  \[ \text{lui } s0, 0061h \]

• Then combine with lower order 16 bits (or immediate)
  \[ \text{ori } s0, s0, 2304h \]

• Stored value: 00612304h
Addressing in Jumps

- Jump instruction uses J Type format
  - 6 bits: operation code
  - 26 bits: address (relative)

- 26 bits, 4 byte increments $\rightarrow$ 256 MB address space

- There is also a "jump register" instruction
instructions
Instruction Types

• Arithmetic: add, sub, mult, div
• Memory access: lb, sb
• Logic: and, or, not, xor
• Comparison: slt
• Branch: beq, bne
• Jumps: j, jal
Data Types

- Instructions operate on varying data types

- 8 bits = 1 byte

- 16 bits = 2 bytes = 1 half word

- 32 bits = 4 bytes = 1 word

- 64 bits = 8 bytes = 2 words = 1 double word
**Arithmetic**

- Load immediately one number \((s0 = 2)\)
  ```
  li $s0, 2
  ```

- Add 4 \((s1 = s0 + 4)\)
  ```
  addi $s1, $s0, 4
  ```

- Subtract 3 \((s2 = s1 - 3)\)
  ```
  addi $s2, $s1, -3
  ```
Memory Access

- So far, assign absolute value to register

```
li $s0, 2
```

- Load value from memory address stored in register

```
lw $s0, 0($s1)
```

- `lw` = load word (4 bytes)
- `$s1` contains memory address
- `0(...) = offset 0`

- Bigger offset example:

```
lw $s0, 8($s1)
```

- Word takes 4 bytes
- Offset 8
→ 32 memory positions added
Direct Memory Access?

- Cannot specify address directly
  - address takes 32 bits
  - instruction size is 32 bits

- Workaround: store address in register first

- 2 instructions needed:
  
lui $s1, 3264h
  ori $s1, $s1, 8278h
  
  - address: 32648278h
  - first load upper memory address halfword (lui)
  - combine with lower memory address halfword (ori)

- Now retrieve value from that memory address
  
lw $s0, 0($s1)
Boolean Logic

• We already encountered Boolean OR:

```
ori $s1, $s1, 8278h
```

• Register only version ($s1 = s2$ OR $s3$)

```
or $s1, $s2, $s3
```

• Note: bitwise operation

```
01010101 OR 11110000 \rightarrow 11110101
```
Other Boolean Operators

- **AND**
  
  ```
  and $s1, $s2, $s3
  ```

- **NOT**
  
  ```
  not $s1, $s2
  ```

- **NOR**
  
  ```
  nor $s1, $s2, $s3
  ```

- **XOR**
  
  ```
  xor $s1, $s2, $s3
  ```
Shift

- Shift left logical

\[
\text{sll } \$s1, \$s2, 4
\]

- shifts all bits left by 4 positions
- \(0000\ 1001 \rightarrow 1001\ 0000\)
- equivalent to multiplication with \(2^4\)

- Corresponding command: shift right logical (srl)
Branches

- No flags!

- Branch includes test

- Example
  
  \[
  \text{beq } \$s1, \$s2, \text{ address}
  \]

  - `beq = branch if equal`
  - branches if registers \$s1 and \$s2 have same value

- Corresponding command: branch if not equal (bne)
Testing Inequality

• Another useful test: $s0 < $s1 ?

• Instruction: set on less than

\[
\text{slt } s2, s0, s1
\]

• Result: $s0 < s1 \rightarrow s2 = 1 \text{ (otherwise 0)}$

• Can be used in branching

\[
\text{slt } s2, s0, s1 \\
\text{bne } s2, \text{zero}, \text{address}
\]
**Addressing in Branches**

- Comparison of register values

  \[ \text{beq register1, register2, address} \]

- **Format:** I Type → address has 16 bits

- Address relative to current program counter

- Branches are typically local: 16 bits typically enough
  (also in 6502: 1 byte relative addressing)
spim
Simulator

• Available at http://spimsimulator.sourceforge.net/
  – versions for Windows, Linux, Mac, etc.

• Installed on CS machines

• We will use this for homeworks
Basic Usage

• Write assembly program as text file

• Start the spim simulator

  % spim
  SPIM Version 7.3. of August 28, 2006
  Copyright 1990-2004 by James R. Larus (larus@cs.wisc.edu).
  (spim)

• Load program and step through the program

• Useful instructions:
  – load "countdown.s"
  – step
  – print $s0
  – reinitialize
Example Program

- Text file "countdown.s"

    .text

    main:
        li $s0, 10  # store 10 in register $s0

    loop:
        addi $s0, $s0, -1  # decrement counter
        bne $s0, $zero, loop  # != 0 ? then loop

    exit:
        jr $ra  # return to callee

- Loops through numbers 10 ... 0 in register $s0
Step through Example

- Ignore initial header code:

  (spim) step
  [0x00400000] 0x8fa40000 lw $4, 0($29) ; 175: lw $a0 0($sp) # argc
  (spim)
  [0x00400004] 0x27a50004 addiu $5, $29, 4 ; 176: addiu $a1 $sp 4 # argv
  (spim)
  [0x00400008] 0x24a60004 addiu $6, $5, 4 ; 177: addiu $a2 $a1 4 # envp
  (spim)
  [0x0040000c] 0x00041080 sll $2, $4, 2 ; 178: sll $v0 $a0 2
  (spim)
  [0x00400010] 0x00041080 addu $6, $6, $2 ; 179: addu $a2 $a2 $v0
  (spim)
  [0x00400014] 0x0c10009 jal 0x00400024 [main] ; 180: jal main

- This handles parameters from the command line
Step through Example

- **First instruction**
  
  (spim) step
  
  [0x00400024] 0x3410000a ori $16, $0, 10 ; 4: li $s0, 10  # store 10 in register $s0

- **Inspect value of register $s0**
  
  (spim) print $s0
  
  Reg 16 = 0x0000000a (10)

- **Decrease loop index variable**
  
  (spim) step
  
  [0x00400028] 0x2210ffff addi $16, $16, -1 ; 7: addi $s0, $s0, -1  # decrement counter
  
  (spim) print $s0
  
  Reg 16 = 0x00000009 (9)
Step through Example

- Check loop termination condition

  (spim) step
  [0x0040002c] 0x1600ffff bne $16, $0, -4 [loop-0x0040002c]; 8: bne $s0, $zero,loop # != 0 ? then loop

- Next iteration

  (spim) step
  [0x00400028] 0x2210ffff addi $16, $16, -1 ; 7: addi $s0, $s0, -1 # decrement counter
  (spim) print $s0
  Reg 16 = 0x00000008 (8)

  [...]

  (spim)
  [0x00400028] 0x2210ffff addi $16, $16, -1 ; 7: addi $s0, $s0, -1 # decrement counter
  (spim)
  [0x0040002c] 0x1600ffff bne $16, $0, -4 [loop-0x0040002c]; 8: bne $s0, $zero,loop # != 0 ? then loop
  (spim)
  [0x00400030] 0x03e00008 jr $31 ; 11: jr $ra # return to callee

- Termination
• Print value of register $s0

• Place in loop:

  move $a0, $s0  # value to print in $a0
  li $v0, 1     # print int
  syscall

• Run in spim

  (spim) reinitialize
  (spim) load "countdown-and-print.s"
  (spim) run
  9876543210