CSE 210: Computer Architecture
Lecture 5: MIPS, Number Systems

Stephen Checkoway

Oberlin College

Oct. 13, 2021

Slides from Cynthia Taylor

Announcements

- Problem Set 1 due Friday 11:59 pm
- Office hours Friday 13:30 – 14:30 pm

Memory Instructions

- `lw   $t0, 0($t1)`
  - `$t0 = Mem[$t1+0]`
  - Loads 4 bytes from `$t1, $t1+1, $t1+2, and $t1+3`
- `sw   $t0, 4($t1)`
  - `Mem[$t1+4] = $t0`
  - Stores 4 bytes at `$t1+4,  $t1+5,  $t1+6, and $t1+7`
- These instructions are the cornerstones of our being able to go to and from memory

Memory Organization

- Viewed as a large, single-dimension array, with an address.
- A memory address is an index into the array
- “Byte Addressing" means that the index points to a byte of memory.

0	**8 bits of data**
1	**8 bits of data**
2	**8 bits of data**
3	**8 bits of data**
4	**8 bits of data**
5	**8 bits of data**
6	**8 bits of data**
...	

Memory Organization

- Bytes are nice, but most data items use larger "words"
- For MIPS, a word is 32 bits or 4 bytes.

0	**32 bits of data**
4	**32 bits of data**
8	**32 bits of data**
12	**32 bits of data**

Registers hold 32 bits of data

- 2^{32} bytes with byte addresses from 0 to $2^{32} - 1$

- 2^{30} words with byte addresses 0, 4, 8, ... $2^{32} - 4$

Memory Operand Example 1

- C code:

```
g = h + A[8];
```

  - g in $s1, h in $s2, base address of A in $s3, A is an array of 4 byte ints
- Compiled MIPS code:
  - Index 8 requires offset of 32

```
lw  $t0, 32($s3)
add $s1, $s2, $t0
```

Translate to MIPS

- C code: g = h + A[5];
  - g in $s1, h in $s2, base address of A in $s3.
  - A is an array of 4-byte ints

A.
```
lw  $t0, 5($s3)
add $s1, $s2, $t0
```

B.
```
lw  $t0, 20($s3)
add $s1, $s2, $t0
```

C.
```
lw  $t0, $s5
add $s1, $s2, $t0
```

D.
```
lw  $t0, $s3
add $s1, $s2, $t0
```

Memory Operand Example 2

- C code:

```
A[12] = h + A[8];
```

  - h in $s2, base address of A in $s3

- Compiled MIPS code:
  - Index 8 requires offset of 32

```
lw  $t0, 32($s3)     # load word
add $t0, $s2, $t0
sw  $t0, 48($s3)     # store word
```

When a 2-byte word is stored in byte-addressed memory (occupying two consecutive bytes), is the most significant byte (MSB) stored in the lower address or the higher address?

A. Low

0	0000 1111	= 15
1	0000 0000	

B. High

0	0000 0000	= 15
1	0000 1111	

C. It Depends

Byte ordering

- Big Endian: Most significant byte in lowest address
- Little Endian: Most significant byte in highest address

Immediate Operands

- Constant data specified in an instruction
  - `addi $s3, $s3, 4`
  - `li $t0, -25`
  - `ori $v0, $t8, 1`

Subtract 2 from $s0 and store in register $s1

A. `addi $s0, $s1, -2`

B. `addi $s1, $s0, -2`

C. `subi $s0, $s1, 2`

D. `subi $s1, $s0, 2`

E. More than one of the above

MIPS Design Principles

- Simplicity favors regularity
  - fixed size instructions
  - small number of instruction formats
- Smaller is faster
  - limited instruction set
  - limited number of registers in register file
- Make the common case fast
  - arithmetic operands from the register file (load-store machine)
  - allow instructions to contain immediate operands

Pseudoinstructions

- `move dest, src          => add dest, $zero, src`
- `subi dest, src, imm     => addi dest, src, -imm`
- `li dest, imm            => addi dest, $zero, imm`

- More complicated expansions are possible

Loading a large number into a register

- Immediates are limited to 16 bits
  - -32768 to 32767 or 0 to 65535
- Numbers outside this range need to be loaded into registers before being used
- load upper immediate instruction sets the most-significant 16 bits of a register
  - ```
    lui   $t0, 0x1234
    ori   $t0, $t0, 0x5678
    ```
- When li is given a value that’s too large, the assembler expands it to lui/ori
    ```

MIPS Questions?

Why we need to learn binary (and other number systems)

- Fundamental to how your computer works
  - Will need a good grasp of binary to understand things like logical operations
  - Will need to translate to binary to work out examples

- Need to understand it to understand many things like network protocols (IP addresses), bit masking, etc.

Positional Notation

- The meaning of a digit depends on its position in a number.
- A number, written as the sequence of digits $d_n d_{n-1} \dots d_2 d_1 d_0$ in base b represents the value

$$d_n * b^n + d_{n-1} * b^{n-1} + \dots + d_2 * b^2 + d_1 * b^1 + d_0 * b^0$$

Consider 101

- In base 10, it represents the number 101 (one hundred one) =
- In base 2, 101_2 =
- In base 8, 101_8 =

$101_5 = ?$

A. 26

B. 51

C. 126

D. 130

$101_{-3}=?$

A. -10

B. 8

C. 10

D. -30

Binary: Base 2

- Used by computers

- A number, written as the sequence of digits $d_n d_{n-1} \ldots d_2 d_1 d_0$ where d is in {0, 1}, represents the value

$$d_n * 2^n + d_{n-1} * 2^{n-1} + \ldots + d_2 * 2^2 + d_1 * 2^1 + d_0 * 2^0$$

Computers Use Binary Because

A. Decimal takes too much space

B. It’s easier to do math with binary

C. It is easy to represent two states (on/off) with electricity

D. None of the above

Decimal: Base 10

- Used by humans

- A number, written as the sequence of digits $d_n d_{n-1} \ldots d_2 d_1 d_0$ where d is in {0,1,2,3,4,5,6,7,8,9}, represents the value

$$d_n * 10^n + d_{n-1} * 10^{n-1} + \ldots + d_2 * 10^2 + d_1 * 10^1 + d_0 * 10^0$$

Hexadecimal: Base 16

- Like binary, but shorter!
- Each digit is a "nibble", or half a byte (4 bits)
- Indicated by prefacing number with 0x (usually)

- A number, written as the sequence of digits $d_n d_{n-1} \ldots d_2 d_1 d_0$ where d is in {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}, represents the value

$$d_n * 16^n + d_{n-1} * 16^{n-1} + \ldots + d_2 * 16^2 + d_1 * 16^1 + d_0 * 16^0$$

Octal: Base 8

- Sometimes used to shorten binary
  - Used to specify UNIX permissions (remember 241?)

- A number, written as the sequence of digits $d_n d_{n-1} \ldots d_2 d_1 d_0$ where d is in {0,1,2,3,4,5,6,7}, represents the value

$$d_n * 8^n + d_{n-1} * 8^{n-1} + \ldots + d_2 * 8^2 + d_1 * 8^1 + d_0 * 8^0$$

$$31_8 = ?_{10}$$

A. 24

B. 25

C. 200

D. 208

E. None of the above

Reading

- Next lecture: Negatives in binary
  - Section 2.4

- Problem Set 1 – due Friday