Load and Store

Lecture 5
Sections 2.5 - 2.6

Robb T. Koether

Hampden-Sydney College

Wed, Sep 4, 2019

1. Load and Store Instructions
   - Load Instructions
   - Store Instructions
2. Big Endian vs. Little Endian
3. Assignment

Outline

1. Load and Store Instructions
   - Load Instructions
   - Store Instructions
2. Big Endian vs. Little Endian
3. Assignment

Outline

1. Load and Store Instructions
   - Load Instructions
   - Store Instructions
2. Big Endian vs. Little Endian
3. Assignment

Load Instructions

The Load Instructions

```
lw      rd,imm(rs)      # rd <- M[rs + imm] 4 bytes
lh      rd,imm(rs)      # rd <- M[rs + imm] 2 bytes
lb      rd,imm(rs)      # rd <- M[rs + imm] 1 byte
```

- The source register *rs* holds a base address (memory).
- The immediate value *imm* is an offset from the base address.
- When loading a halfword or a byte, the value is sign-extended to 32 bits.

Loading an Array

Example (Loading an Array)

```
        .data
array:  .word   10, 20, 30, 40, 50
        .text
        la      $s0,array           # Load base addr
        lw      $t0,0($s0)          # Load 1st element
        lw      $t1,4($s0)          # Load 2nd element
        lw      $t2,8($s0)          # Load 3rd element
        lw      $t3,12($s0)         # Load 4th element
        lw      $t4,16($s0)         # Load 5th element
```

- The program will load the elements of the array into registers `$t0` through `$t4`.

Loading an Array in C++

Example (Loading an Array in C++)

```
int arr[5];
cin >> arr[0];
cin >> arr[1];
cin >> arr[2];
cin >> arr[3];
cin >> arr[4];
```

Example (Loading an Array in C++)

```
int arr[5];
for (int i = 0; i < 5; i++)
    cin >> arr[i];
```

- In C, the array name `arr` is the base address and the index is the offset (times 4).

Load Instructions

The Load Instructions

```
lui     rd,imm          # rd(31:16) <- imm
lhu     rd,imm(rs)      # rd <- M[rs + imm] 2 bytes
lbu     rd,imm(rs)      # rd <- M[rs + imm] 1 byte
```

- Load upper immediate will load a 16-bit value into the upper two bytes of the destination register.
- When loading *unsigned* halfword or byte, the value is zero-extended to 32 bits.

Load Address Pseudo-Instruction

Load Address Pseudo-Instruction

```
lui     $at,0x00001001      # at(31:16) <- 0x1001
                            # at(15:0) <- zero
ori     $a0,$at,0x00000060  # a0 <- at | 0x00000060
```

- The pseudo-instruction `la` (load address) is assembled as “load upper immediate” followed by “or immediate.”
- In this example, the base address is `0x10010060`.
- There are very many load pseudo-instructions.
- See MARS’s help page.

Outline

1. Load and Store Instructions
   - Load Instructions
   - Store Instructions
2. Big Endian vs. Little Endian
3. Assignment

Store Instructions

The Store Instructions

```
sw      rt,imm(rs)      # rt(31:0) -> M[rs + imm]
sh      rt,imm(rs)      # rt(15:0) -> M[rs + imm]
sb      rt,imm(rs)      # rt(7:0) -> M[rs + imm]
```

- There is no sign extension in any of these instructions.

Outline

1. Load and Store Instructions
   - Load Instructions
   - Store Instructions
2. Big Endian vs. Little Endian
3. Assignment

Big Endian vs. Little Endian

- There are two ways to store data in memory.
- Big Endian
  - High-order byte in the lower address.
- Little Endian
  - Low-order byte in the lower address.

Big Endian vs. Little Endian

Example (Big Endian vs. Little Endian)

```
        .data
num:    .word   0x12345678
        .text
        la      $s0, num        # Load address
        lbu     $t0, 0($s0)     # Load byte 0
        lbu     $t1, 1($s0)     # Load byte 1
        lbu     $t2, 2($s0)     # Load byte 2
        lbu     $t3, 3($s0)     # Load byte 3
```

- Run the above program to determine whether a MIPS processor is big endian or little endian.

Outline

1. Load and Store Instructions
   - Load Instructions
   - Store Instructions
2. Big Endian vs. Little Endian
3. Assignment

Assignment

Assignment

- Read Sections 2.5 - 2.6.