Accessing Data in Memory

- RISC architectures are referred to as Load/Store architectures
  - Access to memory is via load and store instructions
  - No memory access is allowed by arithmetic or logic instructions
- Numeric data required for processing must be loaded into registers before it can be referenced by instructions

What Must We Know to Access Memory

- Address of memory location we want to reference
- Whether we want to read a value from memory or write a value to memory
  - Implies direction (remember the data bus is bidirectional)
- If reading from memory,
  - What register will the value be placed
- If writing to memory,
  - What register contains the value to write

Data Transfer/Memory Access

- Reading data from memory: load instructions

Instruction Mnemonic	Operation
Lb	Load byte
lh	Load halfword
lw	Load word
ld	Load double word

- Writing data to memory: store instructions

Instruction Mnemonic	Operation
sb	Store byte
sh	Store halfword
sw	Store word
sd	Store double word

- Data type is part of the mnemonic

Operands for Data Transfer Instructions

- Two operands are required:
  - A register name
    - For loads, the register where data will be placed
    - For stores, the register where the data to be written is located
  - A memory address
    - For loads, the address where the data is read from
    - For stores, the address where the data is to be written

Addresses

- Load and store instructions use *base addressing*
- The memory address is specified in two parts
  - Base address – value in a register
  - Offset value – a constant that is added to the value of the base address
- This type of addressing makes it easy to access data contained in structures; i.e. arrays
- It also makes it easy to implement iterative programming logic; i.e. loops

Address Syntax

Offset (Base Address Register)

Base Address value is established in programming

Offset value is determined by location from the base memory address and the data type being accessed

Examples of Load & Store Instructions

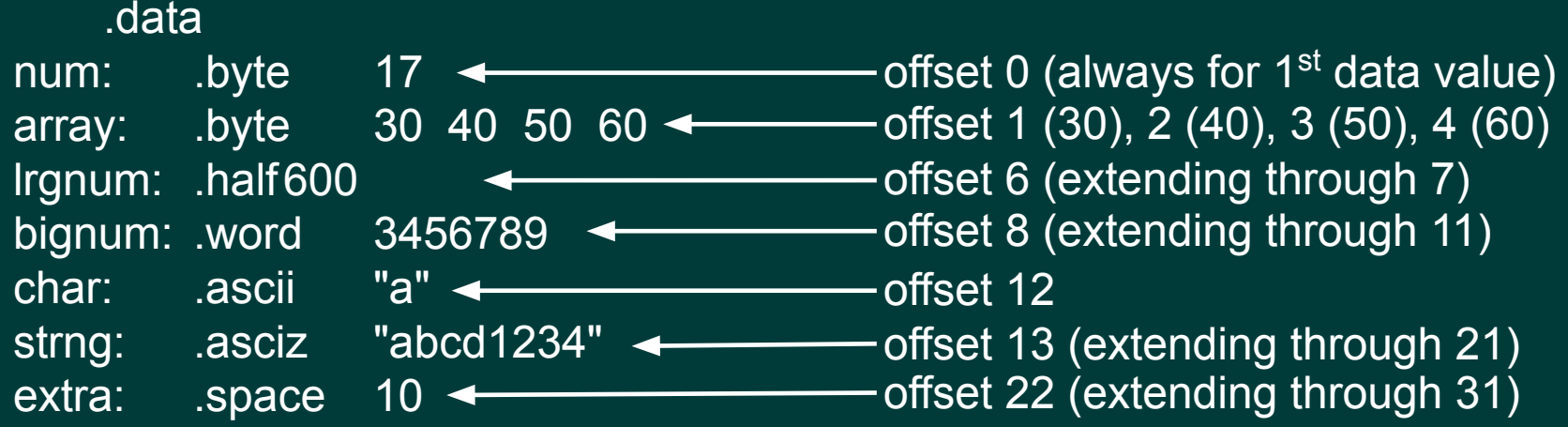
- lb t0, 0(s0)
  - Load into register t0 the byte at offset zero from the memory base address contained in register s0
- lw s4, 8(s0)
  - Load into register s4 the word at offset 8 from the memory base address contained in register s0
- sh a0, 2(s0)
  - Store the halfword value contained in register a0 to offset 2 from the memory base address contained in register s0
- sw s1, 4(s0)
  - Store the word value contained in register s1 to offset 4 from the memory base address contained in register s0

Memory Base Address

- This is a value that typically is the beginning memory address of the data segment
  - Hex value = 0x10010000 (in RARS)
- The base address value must be defined prior to any memory access and is usually done as the first operation in an assembly language program
- Load upper immediate instruction is used
  - Example: lui s0, 0x10010
    - This instruction places the 20-bit value 0x10010 in bits 31 - 12 of register s0 and places zero in the lowest 12 bits.
    - Only lower 32 bits of 64 bit register are affected

Memory Offset

- The memory offset is a value that is added to the base address forming the actual memory address of the memory location to be referenced
- The offset value is determined by the order and types of data defined in the data segment
- Example: our previous data definition



Using Offsets with Memory Base Address

- Using the example on a previous slide, and shown below, write the sequence of instructions to load only the numeric data items into registers t0 – t6.
- Establish the base address in register s0.

```
        .data
num:    .byte   17
array:  .byte   30  40  50  60
lrgnum: .half600
bignum: .word   3456789
char:   .ascii  "a"
strng:  .asciz  "abcd1234"
extra:  .space  10
```

```
        .text
main:   lui   s0, 0x10010
        lb    t0, 0(s0)
        lb    t1, 1(s0)
        lb    t2, 2(s0)
        lb    t3, 3(s0)
        lb    t4, 4(s0)
        lh    t5, 6(s0)
        lw    t6, 8(s0)
```