System Programming

The SPIM S20 MIPS Simulator

Hugo Marcondes
hugom@lisha.ufsc.br
http://www.lisha.ufsc.br/~hugom

Sep 2006

SPIM S20

- MIPS R2000/R3000 simulator
  - Ignores cache and memory latency
  - Ignores delay of operations (multiplies/divides)
  - Expands pseudo instructions to real machine instructions
  - Self contained system
    - Support of few Operating System services
    - Basic I/O (console)
  - Compilation not necessary
    - Directly interprets assembly

Installing SPIM S20

- The source code is available on the class homepage
- Already installed on Lab computers
- Dependencies
  - Bison: parser generator
  - Flex: lexical analyzer generator
  - Xaw3d: graphical library (X Window version)

Running the Simulator

- Just run command spim

- Some useful flags
  - `-bare`: simulate without pseudo-instructions and additional addressing modes
  - `-asm`: simulate the virtual MIPS machine (default)
  - `-file`: Load and execute the assembly code in file
  - `-ef`: Specify the exception handler assembler file

Running an Application

- Application
  - Calculates the mean value of an integer array in memory
- Simulation

  `spim -file lab01.asm`
- Or

1) Run Spim: `spim`
2) Load application: `(spim) read "lab01.asm"`
3) Start the execution of application: `run`

Inspecting Programs Data

- Registers
  - All: `(spim) print_all_regs [hex]`
  - Integer register N: `(spim) print $N`
  - Floating point register N: `(spim) print $fN`
- Application Symbols

  `(spim) print_symbols`
- Memory

  `(spim) print <ADDR>`

Step by Step and Breakpoints

- Executing a program step by step

  `(spim) step [N]` (N is the number of instructions to run)

  - Leaving step-by-step mode

    `(spim) continue`

- Breakpoints

  - Define points on execution flow that stops the simulation and enter in “step” mode
    - Set breakpoint: `(spim) breakpoint <addr>`
    - Delete breakpoint: `(spim) delete <addr>`
    - List Breakpoints: `(spim) list`

User Interfaces

- Xspim
  - X Window Graphical Interface
  - Provides a operation to change the values of registers/memory locations
- PCSpim
  - Windows version of SPIM
  - Tutorial available at http://www.inf.ufsc.br/~cancian/personal/arquitetura.html

System Programming

The SPIM Assembler

Hugo Marcondes
hugom@lisha.ufsc.br
http://www.lisha.ufsc.br/~hugom

Sep 2006

MIPS Register Usage

- Defined in Application Binary Interface (ABI)
  - Low-level interface between Application and Operating System

Register Name	Number	Usage
$zero	0	Constant ZERO
$at	1	Assembly reserved
$v0-1	2-3	Expression Evaluation and result of function
$a0-3	4-7	Arguments
$t0-7	8-15	Temporary (not preserved across call)
$s0-7	16-23	Saved temporary (preserved across call)
$t8-9	24-25	Temporary (not preserved across call)
$k0-1	26-27	Reserved for OS kernel
$gp	28	Pointer to global area
$sp	29	Stack Pointer
$fp	30	Frame Pointer
$ra	31	Return Address

SPIM Assembler Directives

Directive	Function
.data <addr>	Start a data segment, optionally specify an <addr>
.text <addr>	Start a text segment, optionally specify an <addr>
.globl sym	Declare global symbol sym
.align n	Align next allocation pointer in 2^n bytes
.space n	Skip the allocation pointer in n bytes
.byte b_1, ..., b_n	Initialize n byte size data
.half h_1, ..., h_n	Initialize n half-word size data
.word w_1, ..., w_n	Initialize n word size data
.float f_1, ..., f_n	Initialize n single precision floating-point data
.double d_1, ..., d_n	Initialize n double precision floating-point data
.ascii str	Initialize a ASCII str
.asciiz str	Initialize a ASCII str and null-terminate it

MIPS Architecture

- Load/Store Architecture
- Bare machine provides only imm(register) for memory addressing
- Pseudo-instructions supports more 5 memory-addressing mode

(register)	opcode ($gp)
imm	opcode 100
label	opcode array
label +/- imm	opcode array + 8
label +/- imm(register)	opcode array + 8 ($gp)

Basic Data Operations

- Move data between registers

- Use

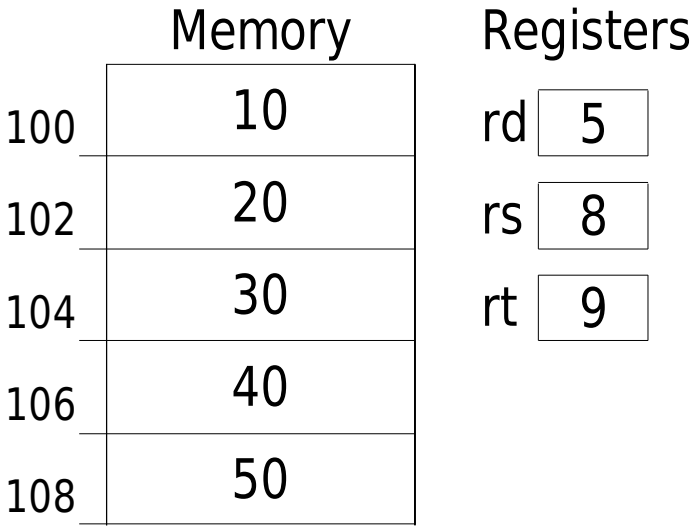
  Move temporary register to a specific purpose register

  Eg. Function arguments

Data Movement

move rd, rs

rd <- rs



Basic Data Operations

- Move data between registers

Data Movement

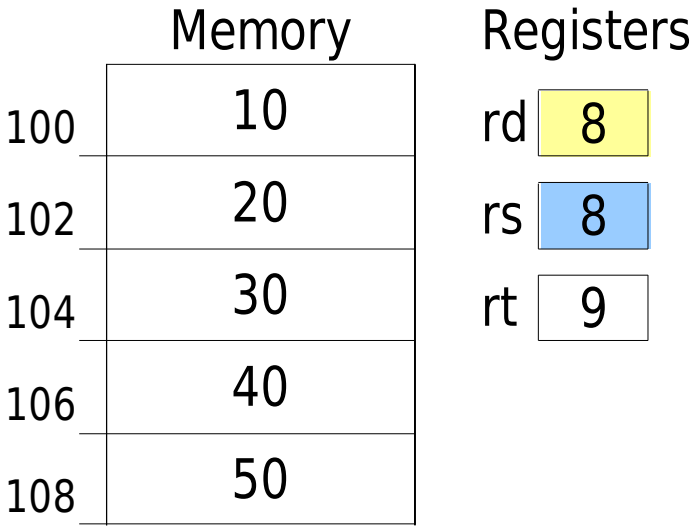
move rd, rs

rd <- rs

- Use

  Move temporary register to a specific purpose register

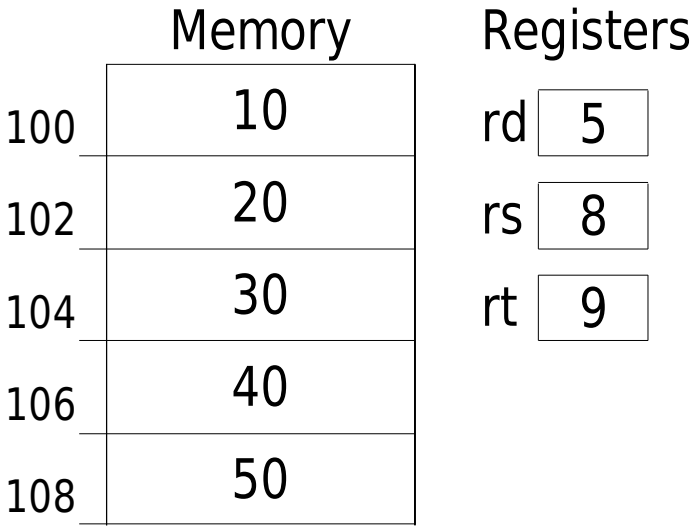
  Eg. Function arguments



Basic Data Operations

- Sum data in registers
- Use
  Basic calculations
- Related Opcodes
  addu – ADD Unsigned
  Do not generate a
  overflow exception

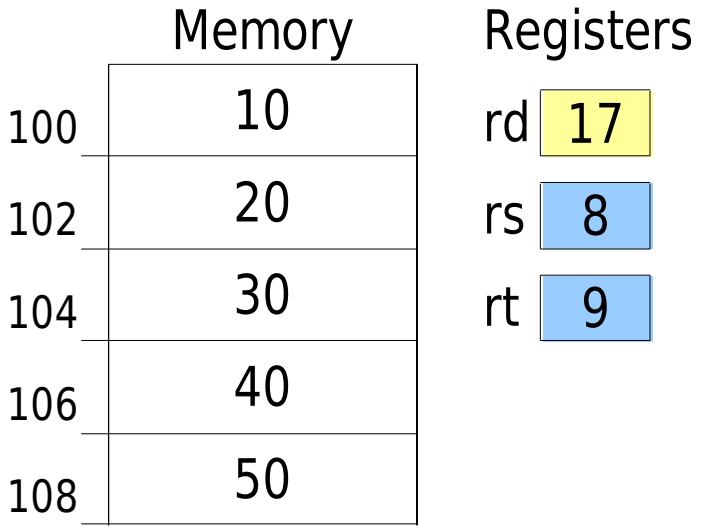
Sum Data
add rd, rs, rt
rd <- rs + rt



Basic Data Operations

- Sum data in registers
- Use
  Basic calculations
- Related Opcodes
  addu – ADD Unsigned
  Do not generate a
  overflow exception

Sum Data
add rd, rs, rt
rd <- rs + rt



Basic Data Operations

- Sum data in register and a 16 bits value

Sum Data

addi rt, rs, imm

rt <- rs + imm

- Use

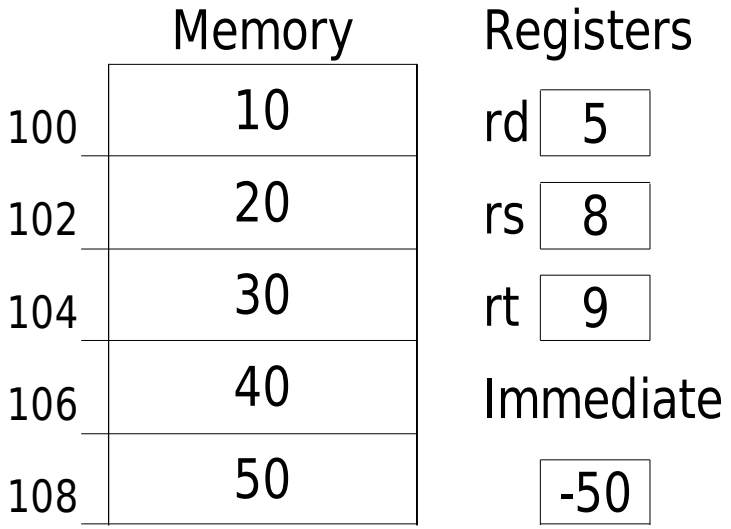
  Basic calculations

  Initialization of values

- Related Opcodes

  addiu – ADD Immediate Unsigned

  Do not generate a overflow exception



Basic Data Operations

- Sum data in register and a 16 bits value

Sum Data

addi rt, rs, imm

rt <- rs + imm

- Use

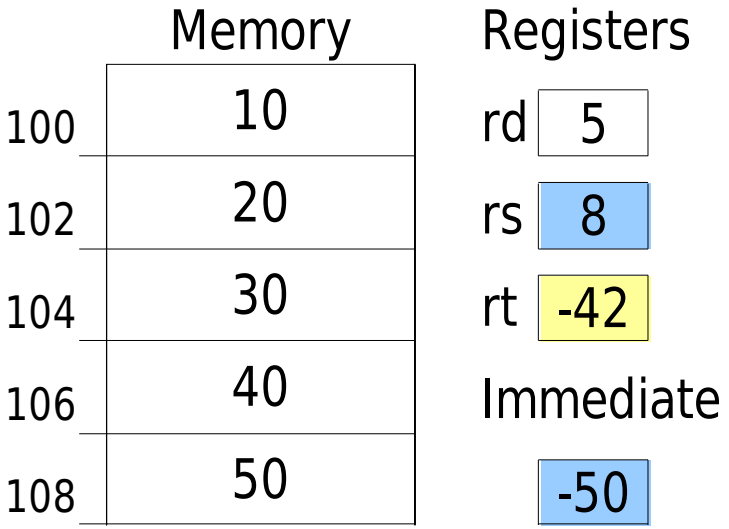
  Basic calculations

  Initialization of values

- Related Opcodes

  addiu – ADD Immediate Unsigned

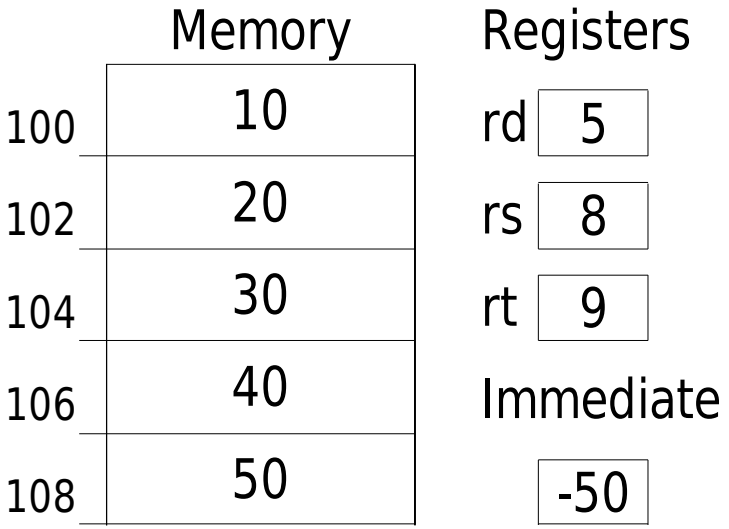
  Do not generate a overflow exception



Memory Access Instructions

- Load a 16 bits value in a register
- Use
  Initialization of values
- Related Opcodes
  lui – Load Upper Immediate
  rt <- imm << 16

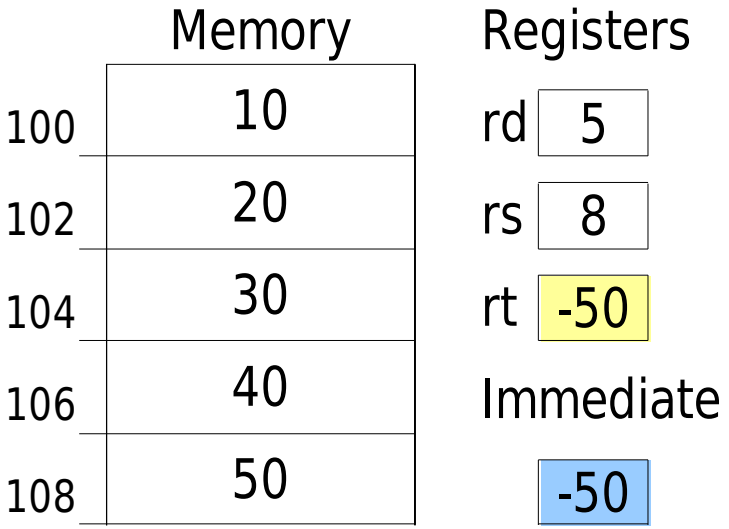
Load Immediate
li rt, imm
rt <- imm



Memory Access Instructions

- Load a 16 bits value in a register
- Use
  Initialization of values
- Related Opcodes
  lui – Load Upper Immediate
  rt <- imm << 16

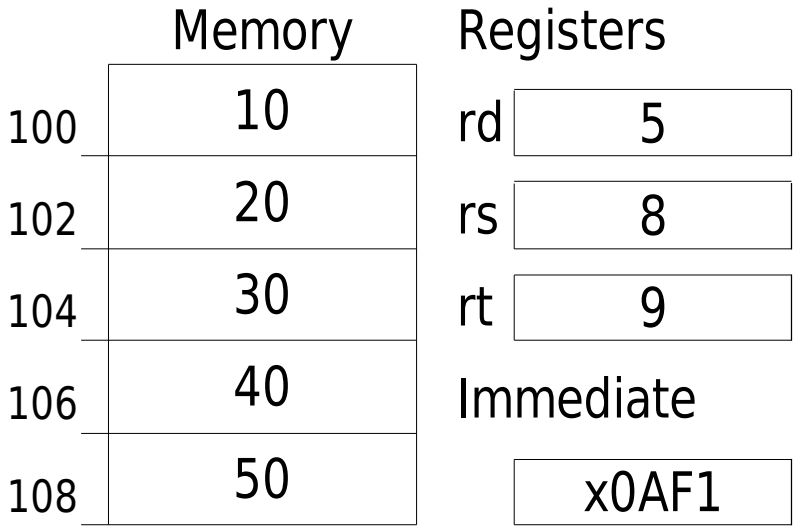
Load Immediate
li rt, imm
rt <- imm



Memory Access Instructions

- Load a 16 bits value in a register
- Use
  Initialization of values
- Related Opcodes
  lui – Load Upper Immediate
  rt <- imm << 16

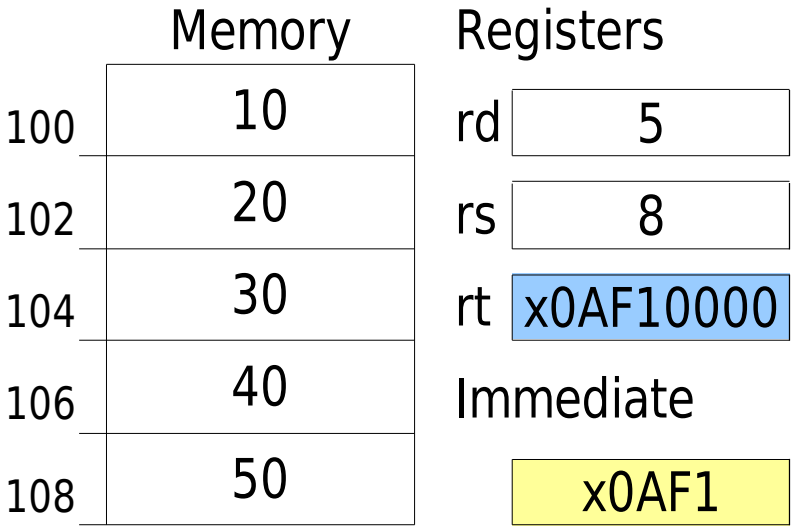
Load Upper Immediate
lui rt, imm
rt <- imm << 16



Memory Access Instructions

- Load a 16 bits value in a register
- Use
  Initialization of values
- Related Opcodes
  lui – Load Upper Immediate
  rt <- imm << 16

Load Upper Immediate
lui rt, imm
rt <- imm << 16



Memory Access Instructions

- Load a 32 bits value in a register

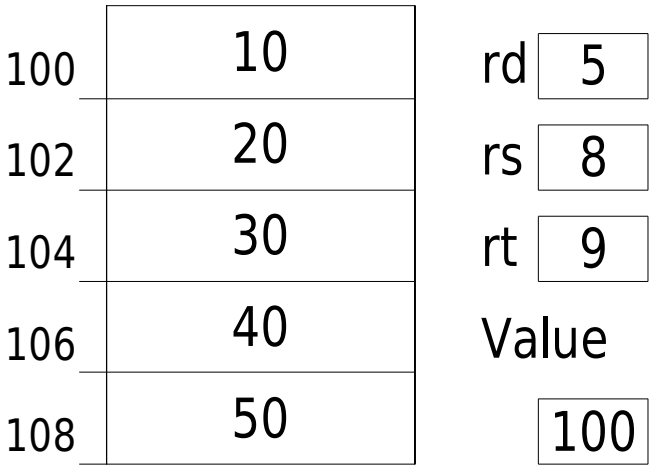
Load Address

la rt, value

rt <- value

- Use

Initialization of values



Memory Access Instructions

- Load a 32 bits value in a register

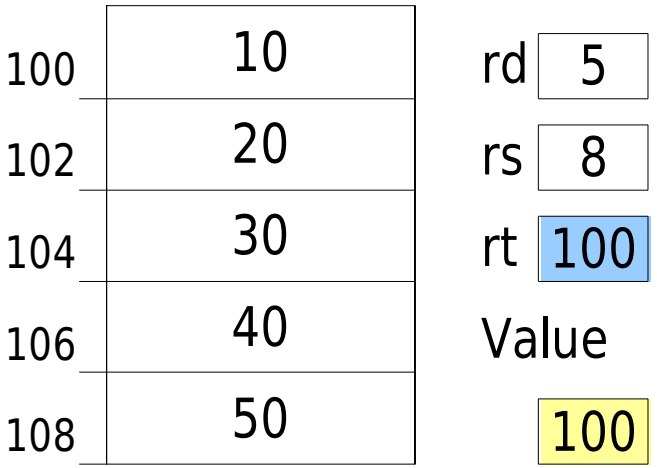
Load Address

la rt, value

rt <- value

- Use

Initialization of values



Memory Access Instructions

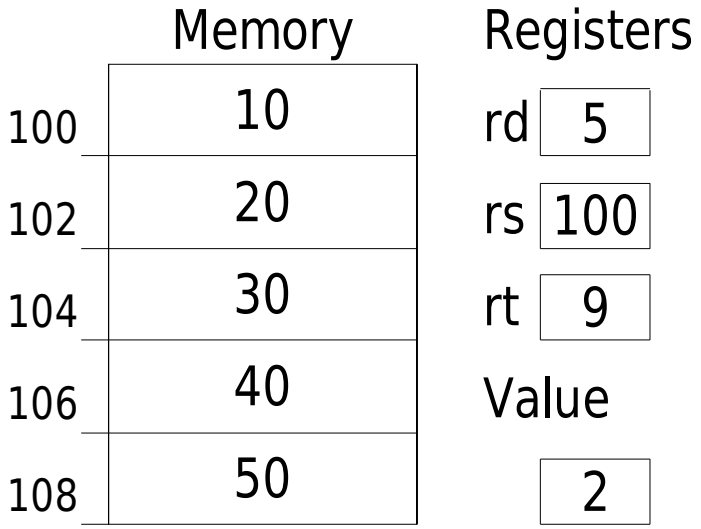
- Load a word on memory to a register

- Use
  Load data from memory

- Related Opcodes

lb – Load byte
lbu – Load byte unsigned
lh – Load Half-Word
lhu – Load Half-Word unsigned
ld – Load Double-Word

Load Word
lw rt, address
rt <- M[address]



Memory Access Instructions

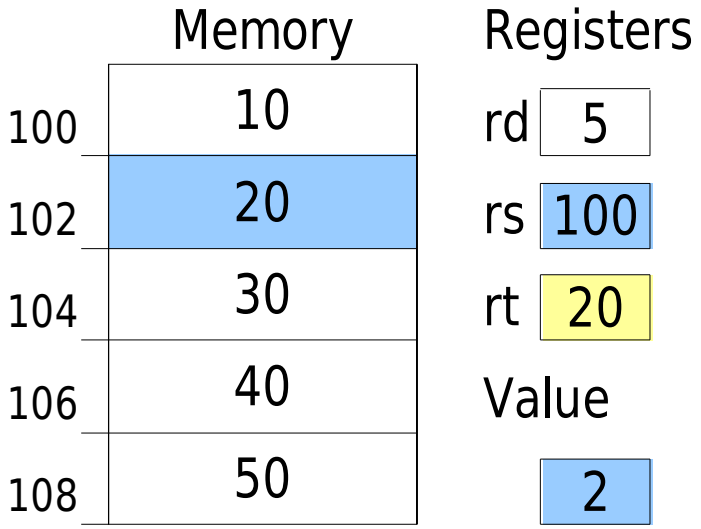
- Load a word on memory to a register

- Use
  Load data from memory

- Related Opcodes

lb – Load byte
lbu – Load byte unsigned
lh – Load Half-Word
lhu – Load Half-Word unsigned
ld – Load Double-Word

Load Word
lw rt, address
rt <- M[address]



Memory Access Instructions

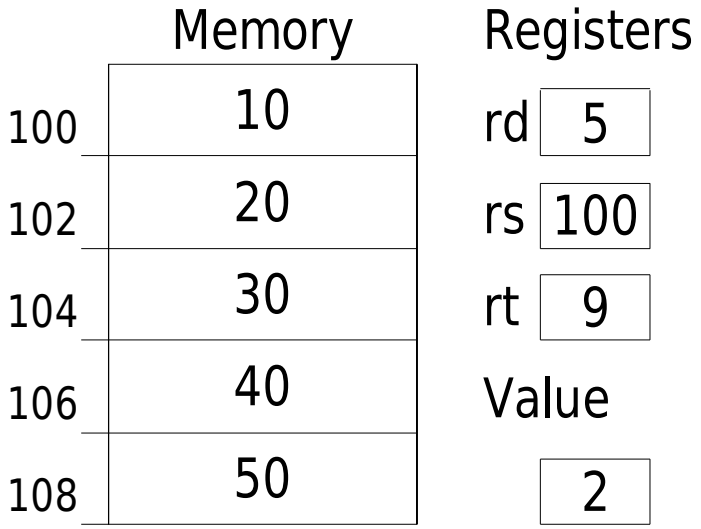
- Store a word in a register in memory

- Use
  Store data in memory

- Related Opcodes

sb – Load byte
sh – Load Half-Word
sd – Load Double-Word

Store Word
sw rt, address
M[address] <- rt



Memory Access Instructions

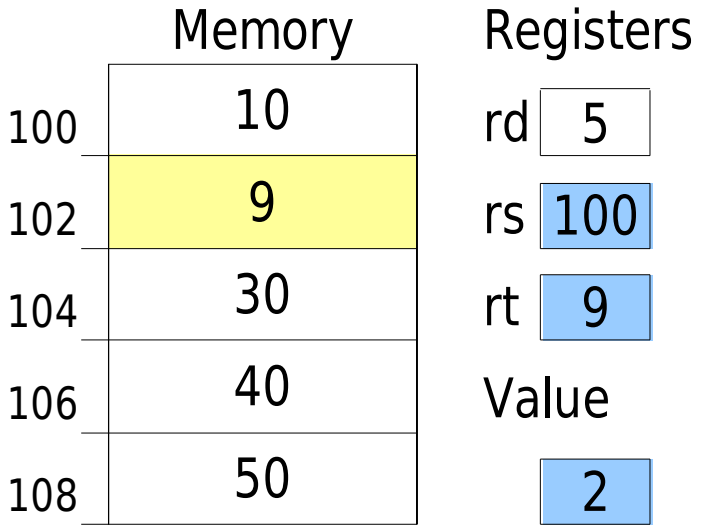
- Store a word in a register in memory

- Use
  Store data in memory

- Related Opcodes

sb – Load byte
sh – Load Half-Word
sd – Load Double-Word

Store Word
sw rt, address
M[address] <- rt



SPIM Assembler
Data Addressing Modes

Mode	MIPS Example Instruction	Meaning
Register	move rt, rs	rt <- rs
Immediate	li rt, imm	rt <- imm
Absolute	lw rt, imm	rt <- M[imm]
Reg. Ind.	lw rt, (rs)	rt <- M[rs]
Reg. Ind. Disp.	lw rt, imm(rs)	rt <- M[rs + imm]
Reg. Ind. Ind.	unavailable	-
Reg. Ind. Post-	unavailable	-
Reg. Ind. -Pre	unavailable	-
Reg. Ind. Scal.	unavailable	-
Reg. Ind. Disp. Scal.	unavailable	-
Memory Ind.	unavailable	-