Assembly

Register Set

Name	Number	Use
$\$0$	0	the constant value 0
$\$at$	1	assembler temporary
$\$v0 - \$v1$	2 - 3	procedure return values
$\$a0 - \$a_3$	4 - 7	procedure arguments
$\$t0 - \$t7$	8 - 15	temporary variables
$\$s0 - \$s7$	16 - 23	saved variables
$\$t8 - \$t9$	24 - 25	temporary variables
$\$k0 - \$k1$	26-27	OS temporaries
$\$gp$	28	global pointer
$\$sp$	29	stack point
$\$fp$	30	frame pointer
$\$ra$	31	procedure return addres

High Level Code	MIPS Assembly Code
$a = b + c$	`add a, b, c`
$a = b - c$	`sub a, b, c`
$a = b + c - d$	`sub t, c, d` `add a, b, t`

Memory

- Commonly used variables are kept in registers
- By using a combo of memory and registers a program can access a large amount of data fairly quickly
- Drawn with low memory addresses toward bottom and high memory addresses toward top
- Byte-addressable memory (each byte has a unique address)
- MIPS uses 32 bit memory addresses and 32 bit data words

Instruction Types

R Type

- Register type instruction
- 3 registers for operands
- 2 registers for sources
- 1 register for destination
- Each field is 5 or 6 bits
- op (6), rs (5), rt (5), rd (5), shamt (5), funct (6)

I Type

- Immediate type instruction
- 2 registers for operands
- 1 immediate operand
- op (6), rs (5), rt (5), imm (16)
- Do the Twos Complement for imm if it overshoots

J Type

- Jump type instruction
- Used only with jump instructions
- op (6), addr (26)

The Power of the Stored Program

- A machine language program is a series of 32-bit numbers representing the instructions
- These instructions can be stored in memory
  - This is the *stored program* concept
- Instructions in stored programs are *fetched* from memory and executed by the processors
- In MIPS programs, the instructions are stored starting at address $0x004000000$
  - Byte addressable
  - A 32-bit (4 byte) instruction addresses advance by 4 bytes
  - Imperative Paradigm

Arithmetic and Logical Instructions

- The AND instruction is useful for masking bits (forcing unwanted bits to 0)

Shift Instructions

- Shifts the value in a register let or right by up to 31 bits
- Shift operations multiply or divide by powers of two

Multiplication and Division Instructions

- Multiplying two 32-bit numbers produces a 64-bit number
- Dividing two 32-bit numbers produce a 32-bit quotient and 32-bit remainder

Branching

Conditional Branching

- Two conditional branch instructions
- Branch if equal (beq)
- Branch if not equal (bne)
- Written as $beq, \$rs, \$rt, imm$
  - This order is reversed from most I-type instructions

If/Else Statements

- bne, add

Switch Statements

- case#
- addi, bne, addi

While Loops

For Loops

Jump

- Jump (j) directly to the instruction at the specified label
- Jump and link (jal) is similar to j but is used by procedures to save a return address
- Jump register (jr) jumps to the address held in a register

Arrays

Addressing Modes

Register-only

- Uses registers for all source and destination operands
- All R-Type instructions

Immediate

- Uses the 16-bit immediate along with registers as operands

Base

- Instructions like lw and sw

PC Relative

- Conditional branching instructions

Pseudo-direct

Compiling, assembling, and loading

Memory Map

- With 32-bit addresses, the address space spans 2^{32} bytes (4 gigabytes (GB))
- Word addresses are divisible by 4 and range from 0 to $0xFFFFFFFC$

Translating and starting a program

- High level code compiled into assembly
- Assembly assembled into machine code in an *object file*
- The linker combines machine code with object code from libraries and other files to produce an entire executable program
- Most compilers do compiling, assembling, and linking
- Finally, the loader load program into memory and stars execution