# CS/ECE 5381/7381 Computer Architecture Spring 2023

Dr. Manikas

Computer Science

Lecture 4: Jan. 26, 2023

# LOCKDOWN BROWSER

- Notes from this week's announcement
- Our exams will use Lockdown Browser
- Exam 1 will be in three weeks (Feb. 16)
- Please install and test Lockdown Browser on your computer
- Use Practice Exam to verify set up and to get familiar with Lockdown Browser

# Assignments

- Quiz 2 – due Sat., Jan. 28 (11:59 pm)
  - Covers concepts from Module 2 (this week)

# Quiz 2 Details

- The quiz is open book and open notes.
- You are allowed 90 minutes to take this quiz.
- You are allowed 2 attempts to take this quiz - your highest score will be kept.
  - Note that some questions (e.g., fill in the blank) will need to be graded manually
- Quiz answers will be made available 24 hours after the quiz due date.

# Instruction Set Principles

(Appendix A, Hennessy and Patterson)

Note: some course slides adopted from publisher-provided material

# Outline

- A.1 Introduction
- A.2 Classifying Instruction Set Architectures
- A.3 Memory Addressing
- A.4 Type and Size of Operands
- A.5 Operations in the Instruction Set
- A.6 Instructions for Control Flow
- A.7 Encoding an Instruction Set
- A.9 MIPS Architecture

# Memory Addressing

- For any ISA class, must define how memory is addressed
  - Address interpretation
  - Address specification

# Interpreting Memory Addresses

- Memory size
  - $2^{10}$ = 1 K (kilo)
  - $2^{20}$ = 1 M (mega)
  - $2^{30}$ = 1 G (giga)
  - $2^{40}$ = 1 T (tera)
  - $2^{50}$ = 1 P (peta)
- 1 B (byte) = 8 b (bits)

# Word Size

- CPU word size = w
  - Typical: w = 16 or 32 bits
- Memory word size = s
  - Typical: s = 8 bits (memory is *byte-addressable*)
- Need to identify order of memory words in CPU word

# Word-Ordering Schemes

- Big Endian
  - Stores the **MSB** ("the big end") at the **lower** byte address and the **LSB** at the **higher** byte address
- Little Endian
  - Stores the **LSB** ("the little end") at the **lower** byte address and the **MSB** at the **higher** byte address
- Example:
  - store ABCD1234H at word address location 0
  - AB is MSB, 34 is LSB

# Memory Ordering Example

Big Endian

| Byte Address | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Data | AB | CD | 12 | 34 |

Little Endian

| Byte Address | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Data | 34 | 12 | CD | AB |

# Addressing Modes

- What object is being addressed?
  - Memory location?
    - Specified by *effective address*
  - Register?
    - Specified by a *register number*
  - Constant?
    - Constant value contained in instruction

# Register Mode

- All values in registers
- ADD R0, R1
  - [Rx] = contents of register x
  - [R0] $\leftarrow$ [R0] + R[1]

# Immediate Mode

- For constants
- ADD R0, #3
  - [R0] $\leftarrow$ [R0] + 3

# Register Indirect Mode

- Use a register value as a *pointer* to memory
- ADD R0, (R1)
  - [R0] ← [R0] + Mem[[R1]]
  - Where Mem[x] is memory location x
  - Register R1 contains the value for x

# Indexed Mode

- Add a value (index) to the memory address
- ADD R0, (R1+R2)
  - [R0] ← [R0] + Mem[[R1] + [R2]]
- Used in array addressing
  - R1 contains base array value
  - R2 contains index value

# Outline

- A.1 Introduction
- A.2 Classifying Instruction Set Architectures
- A.3 Memory Addressing
- A.4 Type and Size of Operands
- A.5 Operations in the Instruction Set
- A.6 Instructions for Control Flow
- A.7 Encoding an Instruction Set
- A.9 MIPS Architecture

# Type and Size of Operands

- Character (8 bits)
- Half word (16 bits)
- Word (32 bits)
- Single-precision floating point (32 bits)
- Double-precision floating point (64 bits)

# Outline

- A.1 Introduction
- A.2 Classifying Instruction Set Architectures
- A.3 Memory Addressing
- A.4 Type and Size of Operands
- A.5 Operations in the Instruction Set
- A.6 Instructions for Control Flow
- A.7 Encoding an Instruction Set
- A.9 MIPS Architecture

# Operations in the Instruction Set

- Data Transfer (load/store)
- Arithmetic/Logic
  - Add, subtract, multiply, divide, negation (2's comp)
  - And, or, not (1's comp)
- Control flow (branch, jump)
- Other
  - System, graphics, etc.

# Outline

- A.1 Introduction
- A.2 Classifying Instruction Set Architectures
- A.3 Memory Addressing
- A.4 Type and Size of Operands
- A.5 Operations in the Instruction Set
- A.6 Instructions for Control Flow
- A.7 Encoding an Instruction Set
- A.9 MIPS Architecture

# Instructions for Control Flow

- Default instruction execution is *sequential*
- Control flow instructions allow instruction execution to go out of sequence
  - Conditional branches
  - Jumps (unconditional branches)
  - Procedure calls and returns

# Control Flow Addressing

- Destination address of next instruction?
  - PC-relative: displacement added to the program counter value
  - Dynamic address: register (or other location) contains destination address

# Conditional Branch Options

- Branch based on condition code (CC)
  - Special test bits (zero/nonzero, positive/negative)
- Branch based on value in condition register
- Compare and branch
  - CJNE – compare and jump if not equal

```
        LOAD R0, 1          ; initialize loop counter
LOOP:   (loop code)
        ADD R0,1            ;increment loop counter
        CJNE R0, 10, LOOP   ;repeat if loop counter < 10
        (next code)
```

# Procedure Invocation Options

- Procedures: functions, subroutines
- Procedure Call
  - Save desired values from calling program
  - Save current PC value (so procedure knows where to return)
- Procedure Return
  - Use saved PC value to return to calling program
  - Restore saved values

# Outline

- A.1 Introduction
- A.2 Classifying Instruction Set Architectures
- A.3 Memory Addressing
- A.4 Type and Size of Operands
- A.5 Operations in the Instruction Set
- A.6 Instructions for Control Flow
- A.7 Encoding an Instruction Set
- A.9 MIPS Architecture

# Encoding an Instruction Set

- How many bits for the instruction (16, 32)?
- How to allocate bits for instruction fields?
  - Op code – operation (load, add, branch)
  - Operand(s) – register/memory addresses
  - Constants (for immediate addressing)
  - Displacement (for indexing, branching)
- Fields may differ for different instructions

# Outline

- A.1 Introduction
- A.2 Classifying Instruction Set Architectures
- A.3 Memory Addressing
- A.4 Type and Size of Operands
- A.5 Operations in the Instruction Set
- A.6 Instructions for Control Flow
- A.7 Encoding an Instruction Set
- A.9 MIPS Architecture

# Outline

- A.9 MIPS Architecture
  - MIPS Instruction Set
  - MIPS Processor Design

# The MIPS Instruction Set

- NOTE: Authors describe “RISC-V” architecture
  - We will use a specific version of this architecture: MIPS
- Why use MIPS?
  - Widely used version
  - Used in prerequisite course CS 4381/ECE 3382
  - Many resources available
    - E.g. tools that we will use in this course
- MIPS Reference Data sheet posted on Canvas

# Arithmetic Operations

- Add and subtract, three operands
  - Two sources and one destination

```
add a, b, c  # a gets b + c
```

- All arithmetic operations have this form

# Arithmetic Example

- C code:

```
f = (g + h) - (i + j);
```

- Compiled MIPS code:

```
add t0, g, h   # temp t0 = g + h
add t1, i, j   # temp t1 = i + j
sub f, t0, t1  # f = t0 - t1
```

# Register Operands

- Arithmetic instructions use register operands
- MIPS has a 32 × 32-bit register file
  - Use for frequently accessed data
  - Numbered 0 to 31
  - 32-bit data called a “word”
- Assembler names
  - $t0, $t1, …, $t9 for temporary values
  - $s0, $s1, …, $s7 for saved variables

# Register Operand Example

- C code:

```
f = (g + h) - (i + j);
```

  - f, …, j in $s0, …, $s4

- Compiled MIPS code:

```
add $t0, $s1, $s2
add $t1, $s3, $s4
sub $s0, $t0, $t1
```

# Memory Operands

- Main memory used for composite data
  - Arrays, structures, dynamic data
- To apply arithmetic operations
  - Load values from memory into registers
  - Store result from register to memory
- Memory is byte addressed
  - Each address identifies an 8-bit byte
- Words are aligned in memory
  - Address must be a multiple of 4
- MIPS is Big Endian
  - Most-significant byte at least address of a word
  - *c.f.* Little Endian: least-significant byte at least address

# Memory Operand Example 1

- C code:

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

  - g in $s1, h in $s2, base address of A in $s3

- Compiled MIPS code:
  - Index 8 requires offset of 32
    - 4 bytes per word

```
lw  $t0, 32($s3)     # load word
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
```

# Registers vs. Memory

- Registers are faster to access than memory
- Operating on memory data requires loads and stores
  - More instructions to be executed
- Compiler must use registers for variables as much as possible
  - Only spill to memory for less frequently used variables
  - Register optimization is important!

# Immediate Operands

- Constant data specified in an instruction

```
addi $s3, $s3, 4
```

- No subtract immediate instruction
  - Just use a negative constant

```
addi $s2, $s1, -1
```

# The Constant Zero

- MIPS register 0 ($zero) is the constant 0
  - Cannot be overwritten
- Useful for common operations
  - E.g., move between registers

```
add $t2, $s1, $zero
```

# Unsigned Binary Integers

- Given an n-bit number

$$x = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2^1 + x_0 2^0$$

- Range: 0 to $+2^n - 1$
- Example
  - $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1011_2$
    $= 0 + \ldots + 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$
    $= 0 + \ldots + 8 + 0 + 2 + 1 = 11_{10}$
- Using 32 bits
  - 0 to +4,294,967,295

# 2s-Complement Signed Integers

- Given an n-bit number

$$x = -x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2^1 + x_0 2^0$$

- Range: $-2^{n-1}$ to $+2^{n-1} - 1$
- Example
  - $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_2$
    $= -1\times2^{31} + 1\times2^{30} + \ldots + 1\times2^2 + 0\times2^1 + 0\times2^0$
    $= -2{,}147{,}483{,}648 + 2{,}147{,}483{,}644 = -4_{10}$
- Using 32 bits
  - –2,147,483,648 to +2,147,483,647

# 2s-Complement Signed Integers

- Bit 31 is sign bit
  - 1 for negative numbers
  - 0 for non-negative numbers
- $-(-2^{n-1})$ can't be represented
- Non-negative numbers have the same unsigned and 2s-complement representation
- Some specific numbers
  - 0: 0000 0000 … 0000
  - –1: 1111 1111 … 1111
  - Most-negative: 1000 0000 … 0000
  - Most-positive: 0111 1111 … 1111

# Signed Negation

- Complement and add 1
  - Complement means 1 → 0, 0 → 1

$$x + \bar{x} = 1111...111_2 = -1$$

$$\bar{x} + 1 = -x$$

- Example: negate +2
  - +2 = 0000 0000 … $0010_2$
  - –2 = 1111 1111 … $1101_2$ + 1
    = 1111 1111 … $1110_2$

# Sign Extension

- Representing a number using more bits
  - Preserve the numeric value
- In MIPS instruction set
  - `addi`: extend immediate value
  - `lb`, `lh`: extend loaded byte/halfword
  - `beq`, `bne`: extend the displacement
- Replicate the sign bit to the left
  - c.f. unsigned values: extend with 0s
- Examples: 8-bit to 16-bit
  - +2: 0000 0010 => 0000 0000 0000 0010
  - –2: 1111 1110 => 1111 1111 1111 1110