Performance

- Performance = 1 / Execution time
- Speedup = $T_{old} / T_{new}$
- Amdahl’s Law: Make the common case fast!
- CPI (clock cycles per instruction)

$$CPI = \frac{t_{ex} \times f_{ck}}{I}$$

where $I$ is the instruction count in the sample program, $f_{ck}$ is the clock frequency of the machine, $t_{ex}$ is the total execution time of the sample program

$$CPI = CPI_{A}w_{A} + CPI_{B}w_{B} + \cdots + CPI_{N}w_{N}$$

- MIPS (million instructions per second)

$$MIPS = \frac{I}{t_{ex} \times 10^6}$$

$$MIPS = \frac{f_{ck}}{CPI \times 10^6}$$

- Native MIPS uses mean (average) CPI
- Relative MIPS is relative to another machine
- Peak MIPS minimizes the CPI (uses best-case CPI)
- MOPS (million operations per second) is like MIPS, but one operation does not necessarily map to one instruction (an instruction can take multiple operations)
- FLOPS (floating point operations per second) is like MOPS, but takes into account only the floating point instructions (use CPI of floating-point class instructions only)

Benchmarks

- A benchmark is a standard way to quantify the performance of a machine.
- Run real applications or “synthetic” programs to get nice numbers.
- Arithmetic Mean (average): Sum up all $n$ values, divide by $n$.
- Weighted Arithmetic Mean: Multiply each value $n$ by weight $w$. Add.
- Geometric Mean: Multiply all $n$ values, take the $n$th root.

Architectures

- Stack (all operands are kept on a stack)
• Accumulator (one operand is in accumulator)
• General Purpose Register (load/store)
• Little-endian (word address is at LSByte)
• Big-endian (word address is at MSByte)

MIPS Instructions

• MIPS is a RISC machine (CPI ≈ 1) with a load/store architecture
• 32 registers, 32-bit (4-byte) words
• Instruction Types
  R-type (e.g., add, xor, jr)
  [ opcode(6) ][ rs(5) ][ rt(5) ][ rd(5) ][ shamt(5) ][ func(6) ]
  I-type (e.g., addi, lui, lw, sw, beq)
  [ opcode(6) ][ rs(5) ][ rt(5) ][ immediate(16) ]
  J-type (e.g., j, jal)
  [ opcode(6) ][ target(26) ]

Addressing Modes

Register: Specify register number
add $t0, $t1, $t2
Immediate: Operand embedded in instruction (16 bits only!)
addi $t0, $t1, 77
Base: (Displacement) Add register and immediate
lw $s0, 12($s1)
PC-Relative: Add immediate (and the “understood zeros”) to PC
beq $0, $t3, -44
Indirect: Register has address
jr, jalr
Pseudodirect: Add 26-bit immediate to top 4 bits of PC
j, jal
Direct (NOT IN MIPS): Address is the immediate
Indexed (NOT IN MIPS): Address is sum of two registers
Update (NOT IN MIPS): Base addressing, plus automatically increment register
Memory Indirect (NOT IN MIPS): Register points to memory location, which points to operand
Inherent (NOT IN MIPS): No operands provided; accumulator assumed

Number Conversion

• Given a number in base X, can you convert it to base Y?
• With a radix point?
• Binary (see 12c lab manual)
  – Unsigned: cannot represent negative numbers
  – Sign-Magnitude: MSb is sign bit
- One’s Complement: if negative, flip all bits
- Two’s Complement: if negative, add one to 1’s comp
- Bias (or Excess): to get in, add bias to number; to get out, subtract bias from number

- Octal: group binary into threes
- Hex: group binary into fours

Logic Design

- Truth tables, logical operations
- Simple logic: and, or, not, xor, mux
- Memory elements: flip-flops, registers
- Propagation delay (how long it takes for the device to give output)
- Simple CMOS circuits (inverter, nand, nor)
- Stuck-at faults
- Timing diagrams

Making Hardware Add

- Half-adder (works only with inputs a,b — not carry-in)
- Full adder (1 bit)
- Ripple-carry (slow): Adds serially; you don’t know the result until all bits are finished
- Carry lookahead (faster): Adds in parallel; uses generate and propagate equations to compute carry-ins.
  \[ g_i = a_i \times b_i \]
  \[ p_i = a_i + b_i \]
  \[ C_{in0} = c_0 \]
  \[ C_{in_n} = g_{n-1} + p_{n-1}C_{in_{n-1}} \]

- Carry select: Adds both with Cin=0 and =1, then figures out what you really wanted
- Two-level logic (fastest, but impractical)
- Manchester carry chain

The ALU

1. Logical operations
2. Full adder (one that works with carries)
3. Add comparison
4. Ripple-carry ALU

Multiplication Schemes

- Multiplication in Hardware:
  Let \( A = \text{multiplicand}; B = \text{multiplier}; P = \text{product} \)
  \( P = (A \times \text{LSb } B); \) shift \( B \) right, shift \( A \) left
- Booth’s Algorithm
Radix-2:

<table>
<thead>
<tr>
<th>Bit pattern</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>—</td>
</tr>
<tr>
<td>01</td>
<td>+A</td>
</tr>
<tr>
<td>10</td>
<td>−A</td>
</tr>
<tr>
<td>11</td>
<td>—</td>
</tr>
</tbody>
</table>

Radix-4:

<table>
<thead>
<tr>
<th>Bit pattern</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>—</td>
</tr>
<tr>
<td>001</td>
<td>+A</td>
</tr>
<tr>
<td>001</td>
<td>+A</td>
</tr>
<tr>
<td>011</td>
<td>+2A</td>
</tr>
<tr>
<td>100</td>
<td>−2A</td>
</tr>
<tr>
<td>101</td>
<td>−A</td>
</tr>
<tr>
<td>110</td>
<td>−A</td>
</tr>
<tr>
<td>111</td>
<td>—</td>
</tr>
</tbody>
</table>

- Remember to sign-extend partial products to $2n$ bits!!

Division Schemes

- Division in Hardware: Restoring, Non-restoring
- Remember to restore to positive remainder, regardless of algorithm
- Both restoring and non-restoring division work only for positive numbers!

Floating Point

- IEEE-754 Floating Point Standard:
  Single-precision (32 bits): 1-bit sign; 8-bit bias-127 exponent; 23-bit unsigned mantissa
  Double-precision (64 bits): 1-bit sign; 11-bit bias-1023 exponent; 52-bit unsigned mantissa
- FP numbers are in the form (S) 1.FFF.. x $2^E$
- The “1.” is the hidden bit – it is always 1 for a normalized number
- ..Which means you MUST have your number normalized to convert to FP

<table>
<thead>
<tr>
<th>Form</th>
<th>Sign</th>
<th>Exponent</th>
<th>Mantissa</th>
</tr>
</thead>
<tbody>
<tr>
<td>±0</td>
<td>0</td>
<td>All 0</td>
<td>All 0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>All 0</td>
<td>All 0</td>
</tr>
<tr>
<td>±Inf</td>
<td>0</td>
<td>All 1</td>
<td>All 0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>All 1</td>
<td>All 0</td>
</tr>
<tr>
<td>NaN</td>
<td>0</td>
<td>All 1</td>
<td>nonzero</td>
</tr>
<tr>
<td>Denorm</td>
<td>0</td>
<td>All 0</td>
<td>nonzero</td>
</tr>
</tbody>
</table>

- Special forms:
- Arithmetic with special forms: Division by zero gives ±Inf; Division by Inf gives ±0. NaN propagates through any computation.

Single-cycle CPU

- Each instruction takes one cycle
- Clock period is propagation delay of the longest instruction (usually load word)
- Control signals for various instructions

Other topics

- See your class notes