Security researchers have discovered a new Windows vulnerability in the shortcut system of the Windows Shell (Explorer). An exploit already exists such that simply viewing an infected USB drive causes a rootkit to be installed on the victim’s system. Here’s the fun part: The rootkit is signed by RealTek.

Datapath Walkthroughs (1/3)

  - Stage 1: fetch this instruction, inc. PC
  - Stage 2: decode to find it’s an add, then read registers $1 and $2
  - Stage 3: add the two values retrieved in Stage 2
  - Stage 4: idle (nothing to write to memory)
  - Stage 5: write result of Stage 3 into register $3

Datapath Walkthroughs (2/3)

- **slti $3, $1, 17**
  - Stage 1: fetch this instruction, inc. PC
  - Stage 2: decode to find it’s an slti, then read register $1
  - Stage 3: compare value retrieved in Stage 2 with the integer 17
  - Stage 4: idle
  - Stage 5: write the result of Stage 3 into register $3

Example: add Instruction

Example: slti Instruction
Datapath Walkthroughs (3/3)

- \texttt{sw} $3$, $17$($1$)
  - Stage 1: fetch this instruction, inc. PC
  - Stage 2: decode to find it’s a \texttt{sw}, then read registers $1$ and $3$
  - Stage 3: add $17$ to value in register $1$ (retrieved in Stage 2)
  - Stage 4: write value in register $3$ (retrieved in Stage 2) into memory address computed in Stage 3
  - Stage 5: idle (nothing to write into a register)

Example: \texttt{sw} Instruction

Why Five Stages? (1/2)

- Could we have a different number of stages?
  - Yes, and other architectures do
- So why does MIPS have five if instructions tend to idle for at least one stage?
  - The five stages are the union of all the operations needed by all the instructions.
  - There is one instruction that uses all five stages: the \texttt{load}

Why Five Stages? (2/2)

- \texttt{lw} $3$, $17$($1$)
  - Stage 1: fetch this instruction, inc. PC
  - Stage 2: decode to find it’s a \texttt{lw}, then read register $1$
  - Stage 3: add $17$ to value in register $1$ (retrieved in Stage 2)
  - Stage 4: read value from memory address compute in Stage 3
  - Stage 5: write value found in Stage 4 into register $3$

Datapath Summary

- The datapath based on data transfers required to perform instructions
- A controller causes the right transfers to happen
A. If the destination reg is the same as the source reg, we could compute the incorrect value!

B. We’re going to be able to read 2 registers and write a 3rd in 1 cycle

CPU clocking (1/2)

• Single Cycle CPU: All stages of an instruction are completed within one long clock cycle.
  • The clock cycle is made sufficient long to allow each instruction to complete all stages without interruption and within one cycle.

CPU clocking (2/2)

• Multiple-cycle CPU: Only one stage of instruction per clock cycle.
  • The clock is made as long as the slowest stage.
  • Several significant advantages over single cycle execution: Unused stages in a particular instruction can be skipped OR instructions can be pipelined (overlapped).

How to Design a Processor: step-by-step

1. Analyze instruction set architecture (ISA) ⇒ datapath requirements
   1. meaning of each instruction is given by the register transfers
   2. datapath must include storage element for ISA registers
   3. datapath must support each register transfer
2. Select set of datapath components and establish clocking methodology
3. Assemble datapath meeting requirements
4. Analyze implementation of each instruction to determine setting of control points that affects the register transfer.
5. Assemble the control logic

Review: The MIPS Instruction Formats

• All MIPS instructions are 32 bits long. 3 formats:
  - R-type
    ```
    op rs rt rd shamt funct
    6 bits 5 bits 5 bits 5 bits 5 bits 5 bits
    ```
  - I-type
    ```
    op rs rt address/immediate
    6 bits 5 bits 16 bits
    ```
  - J-type
    ```
    op address
    6 bits 26 bits
    ```

  The different fields are:
  - op: operation ("opcode") of the instruction
  - rs, rt, rd: the source and destination register specifiers
  - shamt: shift amount
  - funct: selects the variant of the operation in the "op" field
  - address / immediate: address offset or immediate value
  - target address: target address of jump instruction
Step 1a: The MIPS-lite Subset for today

- ADDU and SUBU
- OR Immediate:
- LOAD and STORE Word
- BRANCH:

Step 1: Requirements of the Instruction Set

- Memory (MEM)
- Registers ($R: 32 \times 32$)
- PC
- Extender (sign/zero extend)
- Add/Sub/OR unit for operation on register(s) or extended immediate
- Add 4 (+ maybe extended immediate) to PC
- Compare registers?

Step 2: Components of the Datapath

- Combinational Elements
- Storage Elements
- Clocking methodology

Register Transfer Language (RTL)

- RTL gives the meaning of the instructions
- All start by fetching the instruction
- inst Register Transfers
- ADDU $R(rd) \leftarrow R(rs) + R(rt)$; $PC \leftarrow PC + 4$
- SUBU $R(rd) \leftarrow R(rs) - R(rt)$; $PC \leftarrow PC + 4$
- ORI $R(rt) \leftarrow R(rs) | \text{sign_ext(imm16)}$; $PC \leftarrow PC + 4$
- LOAD $R(rt) \leftarrow \text{MEM}[\text{R(rs) + sign_ext(imm16))]; PC \leftarrow PC + 4$
- STORE $\text{MEM}[\text{R(rs) + sign_ext(imm16)}] \leftarrow \text{R(rt)}$; $PC \leftarrow PC + 4$
- BEQ if ($R(rs) == R(rt)$) then $PC \leftarrow PC + 4 + \text{sign_ext(imm16) (00)}$
- else $PC \leftarrow PC + 4$

ALU Needs for MIPS-lite + Rest of MIPS

- Addition, subtraction, logical OR, ==:
  - ADDU $R(rd) = R[rs] + R[rt]$; ...
  - SUBU $R(rd) = R[rs] - R[rt]$; ...
  - ORI $R(rt) = R[rs] | \text{zero_ext(imm16)}$...
  - BEQ if ($R[rs] == R[rt]$)...
- Test to see if output == 0 for any ALU operation gives == test. How?
- P&H also adds AND, Set Less Than (1 if $A < B$, 0 otherwise)
What Hardware Is Needed? (1/2)

- PC: a register which keeps track of memory addr of the next instruction
- General Purpose Registers
  - used in Stages 2 (Read) and 5 (Write)
  - MIPS has 31 of these (plus $0$)
- Memory
  - used in Stages 1 (Fetch) and 4 (R/W)
  - cache system makes these two stages as fast as the others, on average

What Hardware Is Needed? (2/2)

- ALU
  - used in Stage 3
  - something that performs all necessary functions: arithmetic, logicals, etc.
  - we'll design details later
- Miscellaneous Registers
  - In implementations with only one stage per clock cycle, registers are inserted between stages to hold intermediate data and control signals as they travel from stage to stage.
  - Note: Register is a general purpose term meaning something that stores bits. Not all registers are in the "register file".

Storage Element: Idealized Memory

- Memory (idealized)
  - One input bus: Data In
  - One output bus: Data Out
- Memory word is found by:
  - Address selects the word to put on Data Out
  - Write Enable = 1: address selects the memory word to be written via the Data In bus
- Clock input (CLK)
  - The CLK input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
    - Address valid ⇒ Data Out valid after “access time.”

Storage Element: Register (Building Block)

- Similar to D Flip Flop except
  - N-bit input and output
  - Write Enable input
- Write Enable:
  - negated (or deasserted) (0): Data Out will not change
  - asserted (1): Data Out will become Data In on positive edge of clock

Storage File: Register File

- Register File consists of 32 registers:
  - Two 32-bit output busses: busA and busB
  - One 32-bit input bus: busW
- Register is selected by:
  - RA (number) selects the register to put on busA (data)
  - RB (number) selects the register to put on busB (data)
  - RW (number) selects the register to be written via busW (data) when Write Enable is 1
- Clock input (clk)
  - The clk input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
    - RA or RB valid ⇒ busA or busB valid after “access time.”

Step 3: Assemble DataPath meeting requirements

- Register Transfer Requirements ⇒ Datapath Assembly
- Let's start by analyzing the following Register Transfer Requirements, and build our datapath piece by piece based on those requirements
  - Instruction Fetch
  - add / subtract
  - We'll do the rest next time.
3a: Overview of the Instruction Fetch Unit

- The common RTL operations
  - Fetch the Instruction: mem[PC]
  - Update the program counter:
    - Sequential Code: PC ← PC + 4
    - Branch and Jump: PC ← “something else”

3b: Add & Subtract

• \( R[rd] = R[rs] \text{ op } R[rt] \) (addu rs, rt, rd)
• \( R[rd] = R[rs] \text{ op } \) (addu rs, rt, rd)
• \( R[rd] = R[rs] \text{ op } \) (addu rs, rt, rd)
• \( R[rd] = R[rs] \text{ op } \) (addu rs, rt, rd)

<table>
<thead>
<tr>
<th>Instruction Word</th>
<th>Address</th>
<th>Instruction</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td>PC</td>
<td>Next Address Logic</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Instruction Word</td>
<td>32</td>
<td></td>
</tr>
</tbody>
</table>

Peer Instruction

1) We should use the main ALU to compute PC=PC+4
2) The ALU is inactive for memory reads or writes.
3) The ALU is a synchronous device

How to Design a Processor: step-by-step

1. Analyze instruction set architecture (ISA) ⇒ datapath requirements
   1. meaning of each instruction is given by the register transfers
   2. datapath must include storage element for ISA registers
   3. datapath must support each register transfer
2. Select set of datapath components and establish clocking methodology
3. Assemble datapath meeting requirements
4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
5. Assemble the control logic