Reminder: Homeworks for Next Two Weeks

- **Homework 0**
  - Due next Wednesday (Jan 23), right before lecture

- **Homework 1**
  - Due Monday Jan 28, right before lecture, on Blackboard
  - MIPS warmup, ISA concepts, basic performance evaluation
Reminder: Lab Assignment 1

- A functional C-level simulator for a subset of the MIPS ISA
- Due Friday Feb 1, at the end of Friday lab

- Start early, you will have a lot to learn
- Homework 1 and Lab 1 are synergistic
  - Homework questions are meant to help you in the Lab
Readings for Next Time

- P&H, Chapter 4, Sections 4.1-4.4
- P&P, revised Appendix C – LC3b datapath and microprogrammed operation

Optional:
  - P&P Chapter 5: LC-3 ISA
Review of Last Lecture

- What are two key characteristics of the von Neumann model?

- What difficulty does the dataflow model of computing at the ISA level pose to programmers?

- How is this difficulty eliminated while still exploiting benefits of dataflow?
Review: The Von Neumann Model

Also called **stored program computer** (instructions in memory). Two key properties:

- **Stored program**
  - Instructions stored in a linear memory array
  - Memory is unified between instructions and data
    - The interpretation of a stored value depends on the control signals

- **Sequential instruction processing**
  - One instruction processed (fetched, executed, and completed) at a time
  - Program counter (instruction pointer) identifies the current instr.
  - Program counter is advanced sequentially except for control transfer instructions

When is a value interpreted as an instruction?
A similar tradeoff (control vs. data-driven execution) can be made at the microarchitecture level.

**ISA:** Specifies how the programmer sees instructions to be executed
- Programmer sees a sequential, control-flow execution order vs.
- Programmer sees a data-flow execution order

**Microarchitecture:** How the underlying implementation actually executes instructions
- Microarchitecture can execute instructions in any order as long as it obeys the semantics specified by the ISA when making the instruction results visible to software
  - Programmer should see the order specified by the ISA
Review: Property of ISA vs. Uarch?

- ADD instruction’s opcode
- Number of general purpose registers
- Number of ports to the register file
- Number of cycles to execute the MUL instruction
- Whether or not the machine employs pipelined instruction execution

Remember
- Microarchitecture: Implementation of the ISA under specific design constraints and goals
Review: Design Point

- A set of design considerations and their importance
  - leads to tradeoffs in both ISA and uarch
- Considerations
  - Cost
  - Performance
  - Maximum power consumption
  - Energy consumption (battery life)
  - Availability
  - Reliability and Correctness (or is it?)
  - Time to Market

- Design point determined by the “Problem” space (application space)
Tradeoffs: Soul of Computer Architecture

- ISA-level tradeoffs
- Microarchitecture-level tradeoffs
- System and Task-level tradeoffs
  - How to divide the labor between hardware and software

- Computer architecture is the science and art of making the appropriate trade-offs to meet a design point
  - Why art?
Why Is It (Somewhat) Art?

- We do not (fully) know the future (applications, users, market)
And, the future is not constant (it changes)!
ISA Principles and Tradeoffs
Many Different ISAs Over Decades

- x86
- PDP-x: Programmed Data Processor (PDP-11)
- VAX
- IBM 360
- CDC 6600
- SIMD ISAs: CRAY-1, Connection Machine
- VLIW ISAs: Multiflow, Cydrome, IA-64 (EPIC)
- PowerPC, POWER
- RISC ISAs: Alpha, MIPS, SPARC, ARM

What are the fundamental differences?
- E.g., how instructions are specified and what they do
- E.g., how complex are the instructions
Instruction

- Basic element of the HW/SW interface
- Consists of
  - opcode: what the instruction does
  - operands: who it is to do it to

- Example from Alpha ISA:
Set of Instructions, Encoding, and Spec

- Example from LC-3b ISA
  - [http://www.ece.utexas.edu/~patt/11s.460N/handouts/new_byte.pdf](http://www.ece.utexas.edu/~patt/11s.460N/handouts/new_byte.pdf)
- x86 Manual
- Aside: concept of “bit steering”
  - A bit in the instruction determines the interpretation of other bits
- Why unused instructions?
Bit Steering in Alpha

Figure 3–4: Operate Instruction Format

If bit <12> of the instruction is 0, the Rb field specifies a source register operand.

If bit <12> of the instruction is 1, an 8-bit zero-extended literal constant is formed by bits <20:13> of the instruction. The literal is interpreted as a positive integer between 0 and 255 and is zero-extended to 64 bits. Symbolically, the integer Rbv operand is formed as follows:
What Are the Elements of An ISA?

- Instruction sequencing model
  - Control flow vs. data flow
  - Tradeoffs?

- Instruction processing style
  - Specifies the number of “operands” an instruction “operates” on and how it does so
  - 0, 1, 2, 3 address machines
    - 0-address: stack machine (push A, pop A, op)
    - 1-address: accumulator machine (ld A, st A, op A)
    - 2-address: 2-operand machine (one is both source and dest)
    - 3-address: 3-operand machine (source and dest are separate)
  - Tradeoffs? See your homework question
    - Larger operate instructions vs. more executed operations
    - Code size vs. execution time vs. on-chip memory space
An Example: Stack Machine

+ Small instruction size (no operands needed for operate instructions)
  - Simpler logic
  - Compact code

+ Efficient procedure calls: all parameters on stack
  - No additional cycles for parameter passing

-- Computations that are not easily expressible with “postfix notation” are difficult to map to stack machines
  - Cannot perform operations on many values at the same time (only top N values on the stack at the same time)
  - Not flexible

Figure 3.1 -- The canonical stack machine.
An Example: Stack Machine Operation

http://www.ece.cmu.edu/~koo pman/stack_computers/sec3 _2.html

Figure 3.2 -- An example stack machine.
Other Examples

- PDP-11: A 2-address machine
  - PDP-11 ADD: 4-bit opcode, 2 6-bit operand specifiers
  - Why? Limited bits to specify an instruction
  - Disadvantage: One source operand is always clobbered with the result of the instruction
    - How do you ensure you preserve the old value of the source?

- X86: A 2-address (memory/memory) machine
- Alpha: A 3-address (load/store) machine
- MIPS?
What Are the Elements of An ISA?

- **Instructions**
  - Opcode
  - Operand specifiers (addressing modes)
    - How to obtain the operand?

- **Data types**
  - Definition: Representation of information for which there are instructions that operate on the representation
  - Integer, floating point, character, binary, decimal, BCD
  - Doubly linked list, queue, string, bit vector, stack
    - VAX: INSQUEUE and REMQUEUE instructions on a doubly linked list or queue; FINDFIRST
    - X86: SCAN opcode operates on character strings; PUSH/POP
Data Type Tradeoffs

- What is the benefit of having more or high-level data types in the ISA?
- What is the disadvantage?

- Think compiler/programmer vs. microarchitect

- Concept of semantic gap
  - Data types coupled tightly to the semantic level, or complexity of instructions

- Example: Early RISC architectures vs. Intel 432
  - Early RISC: Only integer data type
  - Intel 432: Object data type, capability based machine
What Are the Elements of An ISA?

- **Memory organization**
  - **Address space**: How many uniquely identifiable locations in memory
  - **Addressability**: How much data does each uniquely identifiable location store
    - Byte addressable: most ISAs, characters are 8 bits
    - Bit addressable: Burroughs 1700. Why?
    - 64-bit addressable: Some supercomputers. Why?
    - 32-bit addressable: First Alpha
  - **Food for thought**
    - How do you add 2 32-bit numbers with only byte addressability?
    - How do you add 2 8-bit numbers with only 32-bit addressability?
    - **Big endian vs. little endian?** MSB at low or high byte.

- **Support for virtual memory**
Some Historical Readings

- If you want to dig deeper


What Are the Elements of An ISA?

- **Registers**
  - How many
  - Size of each register

- **Why is having registers a good idea?**
  - Because programs exhibit a characteristic called *data locality*
  - A recently produced/accessed value is likely to be used more than once (temporal locality)
    - Storing that value in a register eliminates the need to go to memory each time that value is needed
Programmer Visible (Architectural) State

Instructions (and programs) specify how to transform the values of programmer visible state.

| M[0] |  |
| M[1] |  |
| M[2] |  |
| M[3] |  |
| M[4] |  |
| M[N-1] |  |

Memory
array of storage locations
indexed by an address

Program Counter
memory address
of the current instruction

Registers
- given special names in the ISA
  (as opposed to addresses)
- general vs. special purpose

[M]
Aside: Programmer Invisible State

- Microarchitectural state
- Programmer cannot access this directly

- E.g. cache state
- E.g. pipeline registers
Evolution of Register Architecture

- **Accumulator**
  - a legacy from the “adding” machine days

- **Accumulator + address registers**
  - need register indirection
  - initially address registers were special-purpose, i.e., can only be loaded with an address for indirection
  - eventually arithmetic on addresses became supported

- **General purpose registers (GPR)**
  - all registers good for all purposes
  - grew from a few registers to 32 (common for RISC) to 128 in Intel IA-64
Instruction Classes

- Operate instructions
  - Process data: arithmetic and logical operations
  - Fetch operands, compute result, store result
  - Implicit sequential control flow

- Data movement instructions
  - Move data between memory, registers, I/O devices
  - Implicit sequential control flow

- Control flow instructions
  - Change the sequence of instructions that are executed
What Are the Elements of An ISA?

- **Load/store vs. memory/memory architectures**
  
  - Load/store architecture: operate instructions operate only on registers
    - E.g., MIPS, ARM and many RISC ISAs
  
  - Memory/memory architecture: operate instructions can operate on memory locations
    - E.g., x86, VAX and many CISC ISAs
What Are the Elements of An ISA?

- **Addressing modes** specify how to obtain the operands
  - **Absolute**  
    LW rt, 10000
    use immediate value as address
  - **Register Indirect:**  
    LW rt, \( (r_{base}) \)
    use GPR\[r_{base}\] as address
  - **Displaced or based:**  
    LW rt, offset\( (r_{base}) \)
    use offset+GPR\[r_{base}\] as address
  - **Indexed:**  
    LW rt, \( (r_{base}, r_{index}) \)
    use GPR\[r_{base}\]+GPR\[r_{index}\] as address
  - **Memory Indirect**  
    LW rt \((r_{base})\)
    use value at M[ GPR[ r_{base} ] ] as address
  - **Auto inc/decrement**  
    LW Rt, \( (r_{base}) \)
    use GPR\[r_{base}\] as address, but inc. or dec. GPR\[r_{base}\] each time
What Are the Benefits of Different Addressing Modes?

- Another example of programmer vs. microarchitect tradeoff

- Advantage of more addressing modes:
  - Enables better mapping of high-level constructs to the machine: some accesses are better expressed with a different mode → reduced number of instructions and code size
    - Think array accesses (autoincrement mode)
    - Think indirection (pointer chasing)
    - Sparse matrix accesses

- Disadvantage:
  - More work for the compiler
  - More work for the microarchitect
Orthogonal ISA:
- All addressing modes can be used with all instruction types
- Example: VAX
  - (~13 addressing modes) x (>300 opcodes) x (integer and FP formats)

Who is this good for?
Who is this bad for?
What Are the Elements of An ISA?

- **How to interface with I/O devices**
  - Memory mapped I/O
    - A region of memory is mapped to I/O devices
    - I/O operations are loads and stores to those locations
  - Special I/O instructions
    - IN and OUT instructions in x86 deal with ports of the chip

- **Tradeoffs?**
  - Which one is more general purpose?
What Are the Elements of An ISA?

- **Privilege modes**
  - User vs supervisor
  - Who can execute what instructions?

- **Exception and interrupt handling**
  - What procedure is followed when something goes wrong with an instruction?
  - What procedure is followed when an external device requests the processor?
  - Vectored vs. non-vectored interrupts (early MIPS)

- **Virtual memory**
  - Each program has the illusion of the entire memory space, which is greater than physical memory

- **Access protection**

- We will talk about these later