What is Computer Architecture?

- **Structure**: static arrangement of the parts
- **Organization**: dynamic interaction of the parts and their control
- **Implementation**: design of specific building blocks
- **Performance**: behavioral study of the system or of some of its components
Alternate definition: Instruction Set Architecture (ISA)

• Architecture is an **interface** between layers
• ISA is the interface between hardware and software
• ISA is what is visible to the programmer (and ISA might be different for O.S. and applications)
• ISA consists of:
  – instructions (operations and how they are encoded)
  – information units (size, how are they addressed etc.)
  – registers (or more generally processor state)
  – input-output control
Computer structure: Von Neumann model

Memory hierarchy

<table>
<thead>
<tr>
<th>Memory bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
</tr>
<tr>
<td>control</td>
</tr>
</tbody>
</table>

Data path +

CPU

Control

ALU

PC

state

I/O

Memory bus

I/O bus
Computer Organization

- Organization and architecture often used as synonyms
- **Organization** (in this course) refers to:
  - what are the basic blocks of a computer system, more specifically
    - basic blocks of the CPU
    - basic blocks of the memory hierarchy
  - how are the basic blocks designed, controlled, connected?
- Organization used to be transparent to the ISA.
- Today more and more of the ISA is “exposed” to the user/compiler.
## Advances in technology

<table>
<thead>
<tr>
<th>Processor technology</th>
<th>Vacuum tubes</th>
<th>Transistors</th>
<th>Integrated circuits</th>
<th>VLSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory technology</td>
<td>Ferrite core</td>
<td>Semi-conductor</td>
<td>Semi-conductor</td>
<td></td>
</tr>
<tr>
<td>Processor structure</td>
<td>Single processor</td>
<td>Main frames</td>
<td>Micros and minis</td>
<td>PC’s 64-bit arch Superscalar Multithreaded</td>
</tr>
</tbody>
</table>
Some Computer families

• Computers that have the same (or very similar) ISA
  – Compatibility of software between various implementations

• IBM
  – 704, 709, 70xx etc.. From 1955 till 1965
  – 360, 370, 43xx, 33xx From 1965 to the present
  – Power PC

• DEC
  – PDP-11, VAX From 1970 till 1985
  – Alpha (now Compaq, now HP) in 1990’s
More computer families

• Intel
  – Early micros 40xx in early 70’s
  – x86 (086,…,486, Pentium, Pentium Pro, Pentium 3, Pentium 4) from 1980 on
  – IA-64 (Itanium) in 2001

• SUN
  – Sparc, Ultra Sparc 1985 on

• MIPS-SGI
  – Mips 2000, 3000, 4400, 10000 from 1985 on
MIPS is a RISC

- **RISC** = *Reduced Instruction Set Computer*
- R could also stand for “regular”
- All arithmetic-logical instructions are of the form
  \[ R_a \leftarrow R_b \text{ op } R_c \]

- MIPS (as all RISC’s) is a *Load-Store* architecture
  - ALU operates only on operands that are in registers
  - The only instructions accessing memory are load and store
- Sloop is also a RISC, load-store architecture
Registers

- Registers are the “bricks” of the CPU
- Registers are an essential part of the ISA
  - Visible to the hardware and to the programmer
- Registers are
  - Used for high speed storage for operands. For example, if $a, b, c$ are in registers 8, 9, 10 respectively
    \[
    \text{add } $8, $9, $10 \quad \# \ a = b + c
    \]
  - Easy to name (most computer have 32 registers visible to the programmer and their names are 0, 1, 2, …, 31)
  - Used for addressing memory
Registers (ct’d)

• Not all registers are “equal’
  – Some are special-purpose (e.g., register 0 in MIPS is wired to the value 0)
  – Some are used for integer and some for floating-point (e.g., 32 of each in MIPS)
  – Some have restricted use by convention (cf. App. A pp A-22-23)
  – Why no more than 32 or 64 registers
    • Well, sometimes there is (SPARC, Itanium, Cray, Tera)
    • Smaller is faster
    • Instruction encoding (names have to be short)
    • There can be more registers but they are invisible to the ISA
      – this is called register renaming (see CSE 471)
Memory system

- Memory is a *hierarchy* of devices with faster and more expensive ones closer to CPU
  - Registers
  - Caches (hierarchy: on-chip, off-chip)
  - Main memory (DRAM)
  - Secondary memory (disks)
Information units

- Basic unit is the *bit* (has value 0 or 1)
- Bits are grouped together in information units:
  - Byte = 8 bits
  - Word = 4 bytes
  - Double word = 2 words
  - etc.
Memory addressing

- Memory is an array of information units
  - Each unit has the same size
  - Each unit has its own address
  - Address of an unit and contents of the unit at that address are different
Addressing

• In most of today’s computers, the basic I-unit that can be
  addressed is a byte
  – MIPS is *byte addressable*

• The *address space* is the set of all I-units that a program
  can reference
  – The address space is tied to the length of the registers
  – MIPS has 32-bit registers. Hence its address space is 4G bytes
  – Older micros (minis) had 16-bit registers, hence 64 KB address
    space (too small)
  – Some current (Alpha, Itanium, Sparc) machines have 64-bit
    registers, hence an enormous address space
### Addressing words

- Although machines are byte-addressable, words are the most commonly used I-units.
- Every word *starts at an address divisible by 4*.

<table>
<thead>
<tr>
<th>Address</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Word at address 0</td>
</tr>
<tr>
<td>4</td>
<td>Word at address 4</td>
</tr>
<tr>
<td>8</td>
<td>Word at address 8</td>
</tr>
</tbody>
</table>
Big-endian vs. little endian

• Byte order within a word:

\[
\begin{array}{cccc}
3 & 2 & 1 & 0 \\
\end{array}
\]

Little-endian (we’ll use this)

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
\end{array}
\]

Big-endian
The CPU - Instruction Execution Cycle

• The CPU executes a program by repeatedly following this cycle
  1. Fetch the next instruction, say instruction $i$
  2. Execute instruction $i$
  3. Compute address of the next instruction, say $j$
  4. Go back to step 1

• Of course we’ll optimize this but it’s the basic concept
What’s in an instruction?

• An instruction tells the CPU
  – the operation to be performed via the **OPCODE**
  – where to find the operands (source and destination)

• For a given instruction, the ISA specifies
  – what the OPCODE means (semantics)
  – how many operands are required and their types, sizes etc.(syntax)

• Operand is either
  – register (integer, floating-point, PC)
  – a memory address
  – a constant