Computer Organization EECC 550

Week 1  •  Introduction: Modern Computer Design Levels, Components, Technology Trends, Register Transfer Notation (RTN). [Chapters 1, 3]
  •  Instruction Set Architecture (ISA) Characteristics and Classifications: CISC Vs. RISC. [Chapter 3]

Week 2  •  MIPS: An Example RISC ISA. Syntax, Instruction Formats, Addressing Modes, Encoding & Examples. [Chapter 3]

Week 3  •  Central Processor Unit (CPU) & Computer System Performance Measures. [Chapter 2]
  •  CPU Organization: Datapath & Control Unit Design. [Chapter 5]
  –  MIPS Single Cycle Datapath & Control Unit Design.
  –  MIPS Multicycle Datapath and Finite State Machine Control Unit Design.

Week 4  •  Microprogrammed Control Unit Design. [Chapter 5]
  –  Microprogramming Project

Week 5  •  Midterm Review and Midterm Exam

Week 6  •  CPU Pipelining. [Chapter 6]

Week 7  •  The Memory Hierarchy: Cache Design & Performance. [Chapter 7]

Week 8  •  The Memory Hierarchy: Main & Virtual Memory. [Chapter 7]

Week 9  •  Input/Output Organization & System Performance Evaluation. [Chapter 8] If time permits

Week 10 •  Computer Arithmetic & ALU Design. [Chapter 4] If time permits.

Week 11 •  Final Exam.
Computer Hardware Generations

• The First Generation, 1946-59: Vacuum Tubes, Relays, Mercury Delay Lines:
  – First stored program computer: EDSAC (Electronic Delay Storage Automatic Calculator).

• The Second Generation, 1959-64: Discrete Transistors.

• The Third Generation, 1964-75: Small and Medium-Scale Integrated (MSI) Circuits.

The Von-Neumann Computer Model

- Partitioning of the computing engine into components:
  - Central Processing Unit (CPU): Control Unit (instruction decode, sequencing of operations), Datapath (registers, arithmetic and logic unit, buses).
  - Memory: Instruction and operand storage.
  - Input/Output (I/O).
  - The stored program concept: Instructions from an instruction set are fetched from a common memory and executed one at a time.
CPU Machine Instruction Execution Steps

1. **Instruction Fetch**
   - Obtain instruction from program storage

2. **Instruction Decode**
   - Determine required actions and instruction size

3. **Operand Fetch**
   - Locate and obtain operand data

4. **Execute**
   - Compute result value or status

5. **Result Store**
   - Deposit results in storage for later use

6. **Next Instruction**
   - Determine successor or next instruction
Hardware Components of Any Computer

Five classic components of all computers:

1. Control Unit; 2. Datapath; 3. Memory; 4. Input; 5. Output

Processor

Five classic components of all computers:

1. Control Unit; 2. Datapath; 3. Memory; 4. Input; 5. Output

Processor

Keyboard, Mouse, etc.

Disk

Display, Printer, etc.
CPU Organization

• Datapath Design:
  – Capabilities & performance characteristics of principal Functional Units (FUs):
  – (e.g., Registers, ALU, Shifters, Logic Units, ...)
  – Ways in which these components are interconnected (buses connections, multiplexors, etc.).
  – How information flows between components.

• Control Unit Design:
  – Logic and means by which such information flow is controlled.
  – Control and coordination of FUs operation to realize the targeted Instruction Set Architecture to be implemented (can either be implemented using a finite state machine or a microprogram).

• Hardware description with a suitable language, possibly using Register Transfer Notation (RTN).
A Typical Microprocessor Layout:
The Intel Pentium Classic
A Typical Microprocessor Layout:
The Intel Pentium Classic
A Typical Personal Computer (PC) System Board Layout (90% of all computing systems worldwide).
Computer System Components

- Proc
- Caches
- Memory
- System Bus
  - I/O Buses
    - Controllers
    - NICs
    - Networks
  - I/O Devices:
    - Disks
    - Displays
    - Keyboards

I/O Buses

EECC550 - Shaaban
Performance Increase of Workstation-Class Microprocessors 1987-1997

Integer SPEC92 Performance
Moore’s Law:
2X transistors/Chip
Every 1.5 years

Alpha 21264: 15 million
Pentium Pro: 5.5 million
PowerPC 620: 6.9 million
Alpha 21164: 9.3 million
Sparc Ultra: 5.2 million
Increase of Capacity of VLSI Dynamic RAM Chips

<table>
<thead>
<tr>
<th>Year</th>
<th>size (Megabit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1980</td>
<td>0.0625</td>
</tr>
<tr>
<td>1983</td>
<td>0.25</td>
</tr>
<tr>
<td>1986</td>
<td>1</td>
</tr>
<tr>
<td>1989</td>
<td>4</td>
</tr>
<tr>
<td>1992</td>
<td>16</td>
</tr>
<tr>
<td>1996</td>
<td>64</td>
</tr>
<tr>
<td>1999</td>
<td>256</td>
</tr>
<tr>
<td>2000</td>
<td>1024</td>
</tr>
</tbody>
</table>

1.55X/yr, or doubling every 1.6 years
Computer Technology Trends: Rapid Change

• Processor:
  – 2X in speed every 1.5 years; 100X performance in last decade.

• Memory:
  – DRAM capacity: > 2x every 1.5 years; 1000X size in last decade.
  – Cost per bit: Improves about 25% per year.

• Disk:
  – Capacity: > 2X in size every 1.5 years.
  – Cost per bit: Improves about 60% per year.
  – 200X size in last decade.

• Expected State-of-the-art PC by end of year 2003:
  – Processor clock speed: > 3500 MegaHertz (3 GigaHertz)
  – Memory capacity: > 4000 MegaByte (4 GigaBytes)
  – Disk capacity: > 200 GigaBytes (0.2 TeraBytes)
A Simplified View of The Software/Hardware Hierarchical Layers
Hierarchy of Computer Architecture

High-Level Language Programs

Application

Operating System

Compiler

Firmware

Instr. Set Proc.

I/O system

Datapath & Control

Digital Design

Circuit Design

Layout

Software

Machine Language Program

Software/Hardware Boundary

Hardware

Logic Diagrams

Circuit Diagrams

Assembly Language Programs

Instruction Set Architecture

Microprogram

Register Transfer Notation (RTN)
Levels of Program Representation

High Level Language Program

Compiler

Assembly Language Program

Assembler

Machine Language Program

Machine Interpretation

Control Signal Specification

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw $15, 0($2)
lw $16, 4($2)
sw $16, 0($2)
sw $15, 4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

ALUOP[0:3] <= InstReg[9:11] & MASK

Register Transfer Notation (RTN)
## A Hierarchy of Computer Design

<table>
<thead>
<tr>
<th>Level</th>
<th>Name</th>
<th>Modules</th>
<th>Primitives</th>
<th>Descriptive Media</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Electronics</td>
<td>Gates, FF’s</td>
<td>Transistors, Resistors, etc.</td>
<td>Circuit Diagrams</td>
</tr>
<tr>
<td>2</td>
<td>Logic</td>
<td>Registers, ALU’s ...</td>
<td>Gates, FF’s ....</td>
<td>Logic Diagrams</td>
</tr>
<tr>
<td>3</td>
<td>Organization</td>
<td>Processors, Memories</td>
<td>Registers, ALU’s …</td>
<td>Register Transfer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Notation (RTN)</td>
</tr>
<tr>
<td>4</td>
<td>Microprogramming</td>
<td>Assembly Language</td>
<td>Microinstructions</td>
<td>Microprogram</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Assembly language</td>
<td>OS Routines</td>
<td>Assembly language</td>
<td>Assembly Language Programs</td>
</tr>
<tr>
<td></td>
<td>programming</td>
<td></td>
<td>Instructions</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Procedural</td>
<td>Applications</td>
<td>OS Routines</td>
<td>High-level Language Programs</td>
</tr>
<tr>
<td></td>
<td>Programming</td>
<td>Drivers ..</td>
<td>High-level Languages</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Application</td>
<td>Systems</td>
<td>Procedural Constructs</td>
<td>Problem-Oriented Programs</td>
</tr>
</tbody>
</table>

**Low Level - Hardware**

**Firmware**

**High Level - Software**
Hardware Description

• Hardware visualization:
  – **Block diagrams** (spatial visualization):
    Two-dimensional representations of functional units and their interconnections.
  – **Timing charts** (temporal visualization):
    Waveforms where events are displayed vs. time.

• Register Transfer Notation (RTN):
  – A way to describe microoperations capable of being performed by the data flow (data registers, data buses, functional units) at the register transfer level of design (RT).
  – Also describes conditional information in the system which cause operations to come about.
  – A “shorthand” notation for microoperations.

• Hardware Description Languages:
Register Transfer Notation (RTN)

- **Dependent RTN:** When RTN is used after the data flow is assumed to be frozen. No data transfer can take place over a path that does not exist. No statement implies a function the data flow hardware is incapable of performing.

- **Independent RTN:** Describe actions on registers without regard to nonexistence of direct paths or intermediate registers. No predefined data flow.

- The general format of an RTN statement:
  
  **Conditional information:** Action1; Action2

- The conditional statement is often an AND of literals (status and control signals) in the system (a p-term). The p-term is said to imply the action.

- Possible actions include transfer of data to/from registers/memory data shifting, functional unit operations etc.
RTN Statement Examples

A ← B

- A copy of the data in entity B (typically a register) is placed in Register A
- If the destination register has fewer bits than the source, the destination accepts only the lowest-order bits.
- If the destination has more bits than the source, the value of the source is sign extended to the left.

CTL • T0: A = B

- The contents of B are presented to the input of combinational circuit A
- This action to the right of “:” takes place when control signal CTL is active and signal T0 is active.
RTN Statement Examples

MD ← M[MA]
- Memory locations are indicated by square brackets.
- Means the memory data register receives the contents of the main memory (M) as addressed from the Memory Address (MA) register.

AC(0), AC(1), AC(2), AC(3)
- Register fields are indicated by parenthesis.
- The concatenation operation is indicated by a comma.
- Bit AC(0) is bit 0 of the accumulator AC
- The above expression means AC bits 0, 1, 2, 3
- More commonly represented by AC(0-3)

E • T3: CLRWRITE
- The control signal CLRWRITE is activated when the condition E • T3 is active.
Computer Architecture Vs. Computer Organization

• The term **Computer architecture** is sometimes erroneously restricted to computer instruction set design, with other aspects of computer design called implementation.

• More accurate definitions:
  – **Instruction set architecture**: The actual programmer-visible instruction set and serves as the boundary between the software and hardware.
  – **Implementation of a machine has two components**:
    • **Organization**: includes the high-level aspects of a computer’s design such as: The memory system, the bus structure, the internal CPU unit which includes implementations of arithmetic, logic, branching, and data transfer operations.
    • **Hardware**: Refers to the specifics of the machine such as detailed logic design and packaging technology.

• In general, **Computer Architecture** refers to the above three aspects:
  1- Instruction set architecture  2- Organization.  3- Hardware.
Instruction Set Architecture (ISA)

“... the attributes of a [computing] system as seen by the programmer, i.e. the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls the logic design, and the physical implementation.”
– Amdahl, Blaaw, and Brooks, 1964.

The instruction set architecture is concerned with:

- Organization of programmable storage (memory & registers): Includes the amount of addressable memory and number of available registers.
- Data Types & Data Structures: Encodings & representations.
- Instruction Set: What operations are specified.
- Instruction formats and encoding.
- Modes of addressing and accessing data items and instructions
- Exceptional conditions.
Computer Instruction Sets

• Regardless of computer type, CPU structure, or hardware organization, every machine instruction must specify the following:

  – Where to find the operand or operands, if any: Operands may be contained in CPU registers, main memory, or I/O ports.
  – Where to put the result, if there is a result: May be explicitly mentioned or implicit in the opcode.
  – Where to find the next instruction: Without any explicit branches, the instruction to execute is the next instruction in the sequence or a specified address in case of jump or branch instructions.
Instruction Set Architecture (ISA)

Specification Requirements

- Instruction Format or Encoding:
  - How is it decoded?
- Location of operands and result (addressing modes):
  - Where other than memory?
  - How many explicit operands?
  - How are memory operands located?
  - Which can or cannot be in memory?
- Data type and Size.
- Operations
  - What are supported
- Successor instruction:
  - Jumps, conditions, branches.
- Fetch-decode-execute is implicit.
General Types of Instructions

- Data Movement Instructions, possible variations:
  - Memory-to-memory.
  - Memory-to-CPU register.
  - CPU-to-memory.
  - Constant-to-CPU register.
  - CPU-to-output.
  - etc.

- Arithmetic Logic Unit (ALU) Instructions.

- Branch Instructions:
  - Unconditional.
  - Conditional.
# Examples of Data Movement Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Meaning</th>
<th>Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A,B</td>
<td>Move 16-bit data from memory loc. A to loc. B</td>
<td>VAX11</td>
</tr>
<tr>
<td>lwz R3,A</td>
<td>Move 32-bit data from memory loc. A to register R3</td>
<td>PPC601</td>
</tr>
<tr>
<td>li $3,455</td>
<td>Load the 32-bit integer 455 into register $3</td>
<td>MIPS R3000</td>
</tr>
<tr>
<td>MOV AX,BX</td>
<td>Move 16-bit data from register BX into register AX</td>
<td>Intel X86</td>
</tr>
<tr>
<td>LEA.L (A0),A2</td>
<td>Load the address pointed to by A0 into A2</td>
<td>MC68000</td>
</tr>
</tbody>
</table>
## Examples of ALU Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Meaning</th>
<th>Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>MULF A,B,C</td>
<td>Multiply the 32-bit floating point values at mem. locations A and B, and store result in loc. C</td>
<td>VAX11</td>
</tr>
<tr>
<td>nabs r3,r1</td>
<td>Store the negative absolute value of register r1 in r2</td>
<td>PPC601</td>
</tr>
<tr>
<td>ori $2,$1,255</td>
<td>Store the logical OR of register $1 with 255 into $2</td>
<td>MIPS R3000</td>
</tr>
<tr>
<td>SHL AX,4</td>
<td>Shift the 16-bit value in register AX left by 4 bits</td>
<td>Intel X86</td>
</tr>
<tr>
<td>ADD.L D0,D1</td>
<td>Add the 32-bit values in registers D0, D1 and store the result in register D0</td>
<td>MC68000</td>
</tr>
</tbody>
</table>
Examples of Branch Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Meaning</th>
<th>Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLBS A, Tgt</td>
<td>Branch to address Tgt if the least significant bit at location A is set.</td>
<td>VAX11</td>
</tr>
<tr>
<td>bun r2</td>
<td>Branch to location in r2 if the previous comparison signaled that one or more values was not a number.</td>
<td>PPC601</td>
</tr>
<tr>
<td>Beq $2,$1,32</td>
<td>Branch to location PC+4+32 if contents of $1 and $2 are equal.</td>
<td>MIPS R3000</td>
</tr>
<tr>
<td>JCXZ Addr</td>
<td>Jump to Addr if contents of register CX = 0.</td>
<td>Intel X86</td>
</tr>
<tr>
<td>BVS next</td>
<td>Branch to next if overflow flag in CC is set.</td>
<td>MC68000</td>
</tr>
</tbody>
</table>
## Operation Types in The Instruction Set

<table>
<thead>
<tr>
<th>Operator Type</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic and logical</td>
<td>Integer arithmetic and logical operations: add, or</td>
</tr>
<tr>
<td>Data transfer</td>
<td>Loads-stores (move on machines with memory addressing)</td>
</tr>
<tr>
<td>Control</td>
<td>Branch, jump, procedure call, and return, traps.</td>
</tr>
<tr>
<td>System</td>
<td>Operating system call, virtual memory management instructions</td>
</tr>
<tr>
<td>Floating point</td>
<td>Floating point operations: add, multiply.</td>
</tr>
<tr>
<td>Decimal</td>
<td>Decimal add, decimal multiply, decimal to character conversion</td>
</tr>
<tr>
<td>String</td>
<td>String move, string compare, string search</td>
</tr>
<tr>
<td>Graphics</td>
<td>Pixel operations, compression/ decompression operations</td>
</tr>
</tbody>
</table>
# Instruction Usage Example: Top 10 Intel X86 Instructions

<table>
<thead>
<tr>
<th>Rank</th>
<th>Instruction</th>
<th>Integer Average</th>
<th>Percent Total Executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>load</td>
<td></td>
<td>22%</td>
</tr>
<tr>
<td>2</td>
<td>conditional branch</td>
<td></td>
<td>20%</td>
</tr>
<tr>
<td>3</td>
<td>compare</td>
<td></td>
<td>16%</td>
</tr>
<tr>
<td>4</td>
<td>store</td>
<td></td>
<td>12%</td>
</tr>
<tr>
<td>5</td>
<td>add</td>
<td></td>
<td>8%</td>
</tr>
<tr>
<td>6</td>
<td>and</td>
<td></td>
<td>6%</td>
</tr>
<tr>
<td>7</td>
<td>sub</td>
<td></td>
<td>5%</td>
</tr>
<tr>
<td>8</td>
<td>move register-register</td>
<td></td>
<td>4%</td>
</tr>
<tr>
<td>9</td>
<td>call</td>
<td></td>
<td>1%</td>
</tr>
<tr>
<td>10</td>
<td>return</td>
<td></td>
<td>1%</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td><strong>96%</strong></td>
<td></td>
</tr>
</tbody>
</table>

Observation: Simple instructions dominate instruction usage frequency.
Types of Instruction Set Architectures According To Operand Addressing Fields

Memory-To-Memory Machines:
- Operands obtained from memory and results stored back in memory by any instruction that requires operands.
- No local CPU registers are used in the CPU datapath.
- Include:
  - The 4 Address Machine.
  - The 3-address Machine.
  - The 2-address Machine.

The 1-address (Accumulator) Machine:
- A single local CPU special-purpose register (accumulator) is used as the source of one operand and as the result destination.

The 0-address or Stack Machine:
- A push-down stack is used in the CPU.

General Purpose Register (GPR) Machines:
- The CPU datapath contains several local general-purpose registers which can be used as operand sources and as result destinations.
- A large number of possible addressing modes.
- Load-Store or Register-To-Register Machines: GPR machines where only data movement instructions (loads, stores) can obtain operands from memory and store results to memory.
Types of Instruction Set Architectures

Memory-To-Memory Machines: The 4-Address Machine

- No program counter (PC) or other CPU registers are used.
- Instructions specify:
  - Location of first operand.
  - Place to store the result.
  - Location of second operand.
  - Location of next instruction.

Memory-To-Memory Machines:

<table>
<thead>
<tr>
<th>Instruction Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opcode</td>
</tr>
<tr>
<td>add</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bits:</th>
<th>8</th>
<th>24</th>
<th>24</th>
<th>24</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResAddr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Op1Addr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Op2Addr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NextiAddr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Meaning:

\((\text{Res} \leftarrow \text{Op1} + \text{Op2})\)
Types of Instruction Set Architectures
Memory-To-Memory Machines: The 3-Address Machine

- A program counter is included within the CPU which points to the next instruction.
- No CPU storage (general-purpose registers).

### Instruction: `add Res, Op1, Op2`

**Meaning:**

\[(\text{Res} \leftarrow \text{Op1} + \text{Op2})\]

### Instruction Format

<table>
<thead>
<tr>
<th>Bits</th>
<th>8</th>
<th>24</th>
<th>24</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opcode</td>
<td>ResAddr</td>
<td>Op1Addr</td>
<td>Op2Addr</td>
<td></td>
</tr>
</tbody>
</table>

- Opcode specifies which operation to perform.
- ResAddr indicates where to put the result.
- Op1Addr and Op2Addr specify where to find the operands.
Types of Instruction Set Architectures
Memory-To-Memory Machines: The 2-Address Machine

- The 2-address Machine: Result is stored in the memory address of one of the operands.

Instruction:  
add Op2, Op1

Meaning:  
(Op2 ← Op1 + Op2)

Instruction Format

<table>
<thead>
<tr>
<th>Bits</th>
<th>Opcode</th>
<th>Op2Addr</th>
<th>Op1Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>add</td>
<td>24</td>
<td>24</td>
</tr>
</tbody>
</table>

Opcode: Which operation  
Where to find operands  

Where to find result  
24
Types of Instruction Set Architectures
The 1-address (Accumulator) Machine

- A single accumulator in the CPU is used as the source of one operand and result destination.

Instruction:
add Op1

Meaning:
(Acc ← Acc + Op1)

Instruction Format

<table>
<thead>
<tr>
<th>Bits:</th>
<th>8</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Op1Addr</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Where to find operand2, and where to put result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Counter (PC)</td>
<td>Where to find next instruction</td>
</tr>
</tbody>
</table>
A push-down stack is used in the CPU.

Types of Instruction Set Architectures
The 0-address (Stack) Machine

- Instruction: `push Op1`
  - Meaning: \((\text{TOS} \leftarrow \text{Op1})\)
  - Instruction Format
    - Bits: 8
    - Opcode: `push`
    - Where to find operand: `Op1Addr`

- Instruction: `add`
  - Meaning: \((\text{TOS} \leftarrow \text{TOS} + \text{SOS})\)
  - Instruction Format
    - Bits: 8
    - Opcode: `add`

- Instruction: `pop Res`
  - Meaning: \((\text{Res} \leftarrow \text{TOS})\)
  - Instruction Format
    - Bits: 8
    - Opcode: `pop`
    - Memory Destination: `ResAddr`
Types of Instruction Set Architectures
General Purpose Register (GPR) Machines

- CPU contains several general-purpose registers which can be used as operand sources and result destination.

### General Purpose Register (GPR) Machines

**Instruction:**
```
load R8, Op1
```
**Meaning:**
```
(R8 ← Op1)
```

<table>
<thead>
<tr>
<th>Instruction Format</th>
<th>Bits:</th>
<th>Opcode</th>
<th>Where to find operand1</th>
</tr>
</thead>
<tbody>
<tr>
<td>load</td>
<td>8</td>
<td>R8</td>
<td>Op1Addr</td>
</tr>
</tbody>
</table>

**Instruction:**
```
add R2, R4, R6
```
**Meaning:**
```
(R2 ← R4 + R6)
```

<table>
<thead>
<tr>
<th>Instruction Format</th>
<th>Bits:</th>
<th>Opcode</th>
<th>Des Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>8</td>
<td>R2</td>
<td>R4 R6</td>
</tr>
</tbody>
</table>

**Instruction:**
```
store R2, Op2
```
**Meaning:**
```
(Op2 ← R2)
```

<table>
<thead>
<tr>
<th>Instruction Format</th>
<th>Bits:</th>
<th>Opcode</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>store</td>
<td>8</td>
<td>R2</td>
<td>ResAddr</td>
</tr>
</tbody>
</table>
### Expression Evaluation Example with 3-, 2-, 1-, 0-Address, And GPR Machines

For the expression $A = (B + C) \times D - E$ where A-E are in memory

<table>
<thead>
<tr>
<th>3-Address</th>
<th>2-Address</th>
<th>1-Address Accumulator</th>
<th>0-Address Stack</th>
<th>GPR Register-Memory</th>
<th>Load-Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>add A, B, C</td>
<td>load A, B</td>
<td>load B</td>
<td>push B</td>
<td>load R1, B</td>
<td>load R1, B</td>
</tr>
<tr>
<td>mul A, A, D</td>
<td>add A, C</td>
<td>add C</td>
<td>push C</td>
<td>add R1, C</td>
<td>load R2, C</td>
</tr>
<tr>
<td>sub A, A, E</td>
<td>mul A, D</td>
<td>add</td>
<td>add</td>
<td>add R3, R1, R2</td>
<td>load R3, R1, R2</td>
</tr>
<tr>
<td></td>
<td>sub E</td>
<td>push D</td>
<td>sub</td>
<td>load R1, D</td>
<td>load R1, D</td>
</tr>
<tr>
<td></td>
<td>store A</td>
<td>push E</td>
<td>mul</td>
<td>mul R1, D</td>
<td>mul R3, R3, R1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>push</td>
<td>push</td>
<td>sub R1, E</td>
<td>load R1, E</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E</td>
<td>sub</td>
<td>store A, R1</td>
<td>sub R3, R3, R1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pop A</td>
<td></td>
<td>store A, R3</td>
<td>store A, R3</td>
</tr>
</tbody>
</table>

- 3 instructions
- Code size: 30 bytes
- 9 memory accesses
- 4 instructions
- Code size: 28 bytes
- 12 memory accesses
- 5 instructions
- Code size: 20 bytes
- 5 memory accesses
- 8 instructions
- Code size: 23 bytes
- 5 memory accesses
- 5 instructions
- Code size: about 22 bytes
- 5 memory accesses
- 8 instructions
- Code size: about 29 bytes
- 5 memory accesses
## Typical ISA Addressing Modes

<table>
<thead>
<tr>
<th>Addressing Mode</th>
<th>Sample Instruction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>Add R4, R3</td>
<td>R4 ← R4 + R3</td>
</tr>
<tr>
<td>Immediate</td>
<td>Add R4, #3</td>
<td>R4 ← R4 + 3</td>
</tr>
<tr>
<td>Displacement</td>
<td>Add R4, 10 (R1)</td>
<td>R4 ← R4 + Mem[10 + R1]</td>
</tr>
<tr>
<td>Indirect</td>
<td>Add R4, (R1)</td>
<td>R4 ← R4 + Mem[R1]</td>
</tr>
<tr>
<td>Indexed</td>
<td>Add R3, (R1 + R2)</td>
<td>R3 ← R3 + Mem[R1 + R2]</td>
</tr>
<tr>
<td>Absolute</td>
<td>Add R1, (1001)</td>
<td>R1 ← R1 + Mem[1001]</td>
</tr>
<tr>
<td>Memory indirect</td>
<td>Add R1, @ (R3)</td>
<td>R1 ← R1 + Mem[Mem[R3]]</td>
</tr>
<tr>
<td>Autoincrement</td>
<td>Add R1, (R2) +</td>
<td>R1 ← R1 + Mem[R2]</td>
</tr>
<tr>
<td>Autodecrement</td>
<td>Add R1, - (R2)</td>
<td>R2 ← R2 - d</td>
</tr>
<tr>
<td>Scaled</td>
<td>Add R1, 100 (R2) [R3]</td>
<td>R1 ← R1 + Mem[100 + R2 + R3 * d]</td>
</tr>
</tbody>
</table>
Addressing Modes Usage Example

For 3 programs running on VAX ignoring direct register mode:

- Displacement: 42% avg, 32% to 55%
- Immediate: 33% avg, 17% to 43%
- Register deferred (indirect): 13% avg, 3% to 24%
- Scaled: 7% avg, 0% to 16%
- Memory indirect: 3% avg, 1% to 6%
- Misc: 2% avg, 0% to 3%

75% displacement & immediate
88% displacement, immediate & register indirect.

Observation: In addition Register direct, Displacement, Immediate, Register Indirect addressing modes are important.
Displacement Address Size Example

Avg. of 5 SPECint92 programs v. avg. 5 SPECfp92 programs

1% of addresses > 16-bits
12 - 16 bits of displacement needed
Instruction Set Encoding

Considerations affecting instruction set encoding:

– To have as many registers and addressing modes as possible.

– The Impact of the size of the register and addressing mode fields on the average instruction size and on the average program.

– To encode instructions into lengths that will be easy to handle in the implementation. On a minimum to be a multiple of bytes.
  
    • Fixed length encoding: Faster and easiest to implement in hardware.
    • Variable length encoding: Produces smaller instructions.
    • Hybrid encoding.
### Three Examples of Instruction Set Encoding

<table>
<thead>
<tr>
<th>Operations &amp; no of operands</th>
<th>Address specifier 1</th>
<th>Address field 1</th>
<th>Address specifier n</th>
<th>Address field n</th>
</tr>
</thead>
</table>

#### Variable Length Encoding: VAX (1-53 bytes)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Address field 1</th>
<th>Address field 2</th>
<th>Address field 3</th>
</tr>
</thead>
</table>

#### Fixed Length Encoding: DLX, MIPS, PowerPC, SPARC

<table>
<thead>
<tr>
<th>Operation</th>
<th>Address Specifier</th>
<th>Address field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation</td>
<td>Address Specifier 1</td>
<td>Address Specifier 2</td>
</tr>
<tr>
<td>Operation</td>
<td>Address Specifier</td>
<td>Address field 1</td>
</tr>
</tbody>
</table>

#### Hybrid Encoding: IBM 360/370, Intel 80x86
Instruction Set Architecture Trade-offs

- **3-address machine**: shortest code sequence; a large number of bits per instruction; large number of memory accesses.
- **0-address (stack) machine**: Longest code sequence; shortest individual instructions; more complex to program.
- **General purpose register machine (GPR)**:
  - Addressing modified by specifying among a small set of registers with using a short register address (all machines since 1975).
  - **Advantages of GPR**:
    - Low number of memory accesses. Faster, since register access is currently still much faster than memory access.
    - Registers are easier for compilers to use.
    - Shorter, simpler instructions.
- **Load-Store Machines**: GPR machines where memory addresses are only included in data movement instructions between memory and registers (all machines after 1980).
## ISA Examples

<table>
<thead>
<tr>
<th>Machine</th>
<th>Number of General Purpose Registers</th>
<th>Architecture</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDSAC</td>
<td>1</td>
<td>accumulator</td>
<td>1949</td>
</tr>
<tr>
<td>IBM 701</td>
<td>1</td>
<td>accumulator</td>
<td>1953</td>
</tr>
<tr>
<td>CDC 6600</td>
<td>8</td>
<td>load-store</td>
<td>1963</td>
</tr>
<tr>
<td>IBM 360</td>
<td>16</td>
<td>register-memory</td>
<td>1964</td>
</tr>
<tr>
<td>DEC PDP-11</td>
<td>8</td>
<td>register-memory</td>
<td>1970</td>
</tr>
<tr>
<td>DEC VAX</td>
<td>16</td>
<td>register-memory</td>
<td>1977</td>
</tr>
<tr>
<td>Motorola 68000</td>
<td>16</td>
<td>register-memory</td>
<td>1980</td>
</tr>
<tr>
<td>MIPS</td>
<td>32</td>
<td>load-store</td>
<td>1985</td>
</tr>
<tr>
<td>SPARC</td>
<td>32</td>
<td>load-store</td>
<td>1987</td>
</tr>
</tbody>
</table>
## Examples of GPR Machines

<table>
<thead>
<tr>
<th>Number of memory addresses</th>
<th>Maximum number of operands allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2 or 3</td>
<td>2 or 3</td>
</tr>
</tbody>
</table>

SPARK, MIPS  
PowerPC, ALPHA  
Intel 80x86, Motorola 68000  
VAX
Complex Instruction Set Computer (CISC)

- Emphasizes doing more with each instruction.
- Motivated by the high cost of memory and hard disk capacity when original CISC architectures were proposed:
  - When M6800 was introduced: 16K RAM = $500, 40M hard disk = $55,000
  - When MC68000 was introduced: 64K RAM = $200, 10M HD = $5,000
- Original CISC architectures evolved with faster, more complex CPU designs, but backward instruction set compatibility had to be maintained.
- Wide variety of addressing modes:
  - 14 in MC68000, 25 in MC68020
- A number instruction modes for the location and number of operands:
  - The VAX has 0- through 3-address instructions.
- Variable-length or hybrid instruction encoding is used.
Example CISC ISAs
Motorola 680X0

18 addressing modes:

- Data register direct.
- Address register direct.
- Immediate.
- Absolute short.
- Absolute long.
- Address register indirect.
- Address register indirect with postincrement.
- Address register indirect with predecrement.
- Address register indirect with displacement.
- Address register indirect with index (8-bit).
- Address register indirect with index (base).
- Memory indirect postindexed.
- Memory indirect preindexed.
- Program counter indirect with index (8-bit).
- Program counter indirect with index (base).
- Program counter indirect with displacement.
- Program counter memory indirect postindexed.
- Program counter memory indirect preindexed.

Operand size:

- Range from 1 to 32 bits, 1, 2, 4, 8, 10, or 16 bytes.

Instruction Encoding:

- Instructions are stored in 16-bit words.
- the smallest instruction is 2- bytes (one word).
- The longest instruction is 5 words (10 bytes) in length.
Example CISC ISA:
Intel X86, 386/486/Pentium

12 addressing modes:
- Register.
- Immediate.
- Direct.
- Base.
- Base + Displacement.
- Index + Displacement.
- Scaled Index + Displacement.
- Based Index.
- Based Scaled Index.
- Based Index + Displacement.
- Based Scaled Index + Displacement.
- Relative.

Operand sizes:
- Can be 8, 16, 32, 48, 64, or 80 bits long.
- Also supports string operations.

Instruction Encoding:
- The smallest instruction is one byte.
- The longest instruction is 12 bytes long.
- The first bytes generally contain the opcode, mode specifiers, and register fields.
- The remainder bytes are for address displacement and immediate data.
Reduced Instruction Set Computer (RISC)

- Focuses on reducing the number and complexity of instructions of the machine.
- Reduced number of cycles needed per instruction.
  - Goal: At least one instruction completed per clock cycle.
- Designed with CPU instruction pipelining in mind.
- Fixed-length instruction encoding.
- Only load and store instructions access memory.
- Simplified addressing modes.
  - Usually limited to immediate, register indirect, register displacement, indexed.
- Delayed loads and branches.
- Prefetch and speculative execution.
- Examples: MIPS, HP-PA, UltraSpark, Alpha, PowerPC.
Example RISC ISA:

PowerPC

8 addressing modes:

- Register direct.
- Immediate.
- Register indirect.
- Register indirect with immediate index (loads and stores).
- Register indirect with register index (loads and stores).
- Absolute (jumps).
- Link register indirect (calls).
- Count register indirect (branches).

Operand sizes:

- Four operand sizes: 1, 2, 4 or 8 bytes.

Instruction Encoding:

- Instruction set has 15 different formats with many minor variations.
- All are 32 bits in length.
Example RISC ISA:

HP Precision Architecture, HP-PA

7 addressing modes:

- Register
- Immediate
- Base with displacement
- Base with scaled index and displacement
- Predecrement
- Postincrement
- PC-relative

Operand sizes:

- Five operand sizes ranging in powers of two from 1 to 16 bytes.

Instruction Encoding:

- Instruction set has 12 different formats.
- All are 32 bits in length.
Example RISC ISA:

SPARC

5 addressing modes:
- Register indirect with immediate displacement.
- Register indirect indexed by another register.
- Register direct.
- Immediate.
- PC relative.

Operand sizes:
- Four operand sizes: 1, 2, 4 or 8 bytes.

Instruction Encoding:
- Instruction set has 3 basic instruction formats with 3 minor variations.
- All are 32 bits in length.
Example RISC ISA:

DEC/Compaq Alpha AXP

4 addressing modes:
• Register direct.
• Immediate.
• Register indirect with displacement.
• PC-relative.

Operand sizes:
• Four operand sizes: 1, 2, 4 or 8 bytes.

Instruction Encoding:
• Instruction set has 7 different formats.
• All are 32 bits in length.
RISC ISA Example:

MIPS R3000

Instruction Categories:
- Load/Store.
- Computational.
- Jump and Branch.
- Floating Point (using coprocessor).
- Memory Management.
- Special.

Instruction Encoding: 3 Instruction Formats, all 32 bits wide.

<table>
<thead>
<tr>
<th>OP</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>sa</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP</td>
<td>rs</td>
<td>rt</td>
<td>immediate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OP</td>
<td>jump target</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4 Addressing Modes:
- Base register + immediate offset (loads and stores).
- Register direct (arithmetic).
- Immediate (jumps).
- PC relative (branches).

Operand Sizes:
- Memory accesses in any multiple between 1 and 8 bytes.

Registers

R0 - R31
PC
HI
LO
Evolution of Instruction Set Architectures

Single Accumulator (EDSAC 1950)

Accumulator + Index Registers
(Manchester Mark I, IBM 700 series 1953)

Separation of Programming Model from Implementation

High-level Language Based (B5000 1963)

Concept of an ISA Family (IBM 360 1964)

General Purpose Register (GPR) Machines

Complex Instruction Sets (CISC) (Vax, Motorola 68000, Intel x86 1977-80)

Load/Store Architecture (CDC 6600, Cray 1 1963-76)

RISC (MIPS, SPARC, HP-PA, IBM RS6000, ... 1987)