Instruction Set

- Common instruction sets can be classified into two basic types:
  - **load – store (*L/S*)** *architecture and*
  - ***register – memory (R/M)*** *architecture:*

Instruction Set

- The **L/S instruction set** includes the RISC microprocessors.
- **Arguments** are **in registers** before their execution.
- An **ALU instruction has** both source operands and result specified as registers.
- The **advantages of the L/S architecture** are:
  - regularity of execution and
  - ease of instruction decode.

Instruction Set

- The **R/M architectures** include instructions that operate on **operands in registers** or with one of the **operands in memory.**

- In the R/M architecture, an ADD instruction might sum a register value and a value contained in memory, with the result going to a register.

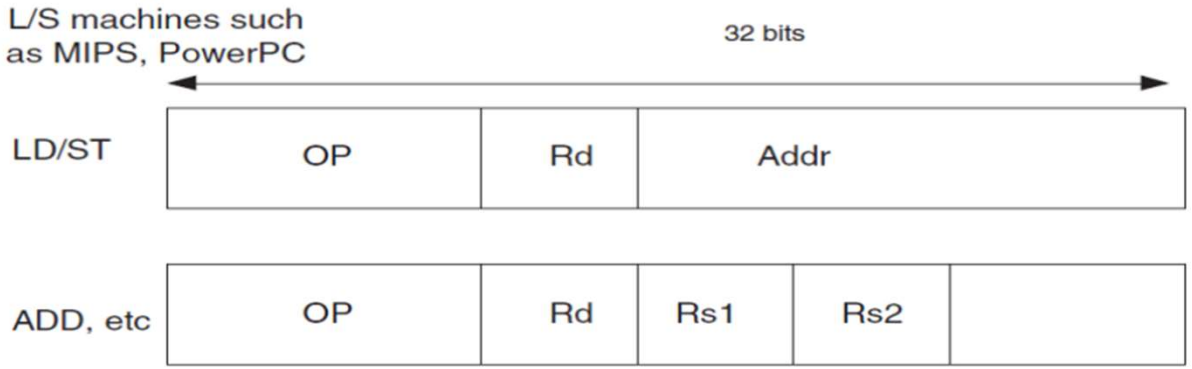
Instruction Set

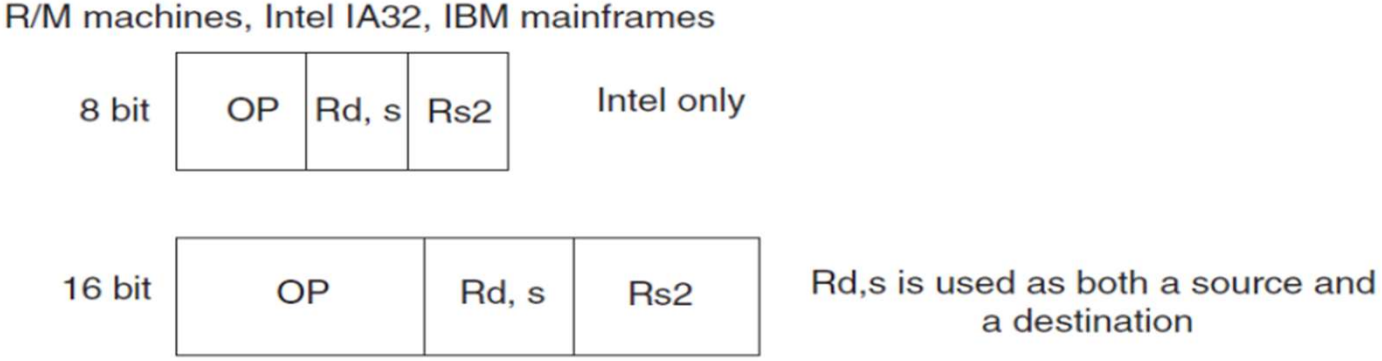
Feature	Load-Store Architecture (L/S)	Register-Memory Architecture (R/M)
Memory Access	Memory access is restricted to load and store instructions.	Memory can be accessed directly by arithmetic and logical instructions.
Registers	ALU operations only work with registers.	ALU operations can work with both memory and registers.
Instruction Simplicity	Instructions are simpler and uniform.	Instructions are more complex.
Code Size	Typically requires more instructions, leading to larger code size.	Requires fewer instructions, leading to more compact code.
Performance	Faster clock speed, more efficient pipelining.	May have slower clock speed due to complex instructions.
Example Architectures	ARM, MIPS, RISC-V	x86, IBM System/360

Instruction Set

- The **trade - off in instruction sets** is an **area – time compromise**.
- The **R/M approach** offers a program representation using **fewer instructions of variable size** compared with L/S.
- The **variable instruction size** makes **decoding more difficult.**
- The decoding of multiple instructions requires predicting the starting point of each. The R/M processors require more circuitry (and area) to be devoted to instruction fetch and decode.

Basic Concepts In Processor Architecture





Basic Concepts In Processor Architecture

- The instruction format defines how the bits within an instruction are organized. Generally, instructions are broken down into several fields, which may include the following:
- **Opcode**: Specifies the operation (e.g., ADD, SUB, LOAD) that the instruction will perform.
- **Operands**: The data or addresses on which the operation is performed (these could be registers, memory locations, or immediate values).
- **Addressing Mode**: Specifies how to interpret the operands (e.g., direct, indirect, immediate addressing).

Instruction Set Mnemonic Operations

Mnemonic	Operation
ADD	Addition
SUB	Subtraction
MPY	Multiplication
DIV	Division
CMP	Compare
LD	Load (a register from memory)
ST	Store (a register to memory)
LDM	Load multiple registers
STM	Store multiple registers
MOVE	Move (register to register or memory to memory)
SHL	Shift left
SHR	Shift right
BR	Unconditional branch
BC	Conditional branch
BAL	Branch and link

Some Instruction Set Conventions

- To indicate the data type that the operation specifies, the operation mnemonic is extended by a data - type indicator:
- **OP.W** might indicate an OP for integers, while
- **OP.F** indicates a floating - point operation.

Modifier	Data Type
B	Byte (8 bits)
H	Halfword (16 bits)
W	Word (32 bits)
F	Floating point (32 bits)
D	Double-precision floating point (64 bits)
C	Character or decimal in an 8-bit format

Typical data - type modifiers are shown in above table.
A typical instruction has the form ***OP.M destination, source 1, source 2.***
The source and destination specification has the form of either a register or a memory location

Branches

- ***Branches (or jumps)*** *manage* ***program control flow.***

- *They typically consist of* **unconditional BR**, **conditional BC**, and **subroutine call and return** (link).

- Typically, the **CC is set by an ALU instruction** to record **one of several results**, for example, specifying whether the instruction has generated
    1. a positive result,
    2. a negative result,
    3. a zero result, or
    4. an overflow.

Interrupts and Exceptions:

- Many embedded SOC controllers have external interrupts and internal exceptions
- These facilities can be managed and supported in various ways:

1. ***User Requested versus Coerced*** *(Forcefully)**: The former often covers executions like* divide by zero, while the latter is usually triggered by external events.
2. ***Maskable versus Nonmaskable:*** *The former type of event can be ignored*, while the latter cannot be ignored.
3. ***Terminate versus Resume:*** *An event such as divide by zero would terminate* ordinary processing, while a processor resumes operation.
4. ***Asynchronous versus Synchronous:*** *Interrupt events can occur in asynchrony* with the processor clock by an external agent or not, as when caused by a program's execution.
5. ***Between versus Within Instructions:*** *Interrupt events can be recognized* only between instructions or within an instruction execution.

Interrupts and Exceptions

- Interrupts are generally generated by external hardware or software signals (i.e., events that are not directly related to the current execution of a program).

- Exceptions are generated by the CPU itself due to errors or special conditions during the execution of instructions.

- Once the exception is handled, the latter instructions are restarted from scratch

Conditional and Unconditional JUMP instructions in 8085 Microprocessor

Opcode	Operand	Meaning
JMP	16-bit address	Jump unconditionally

Opcode	Description	Flag Status	Operand	Meaning
JC	Jump on Carry	CY=1	16-bit address	Jump conditionally
JNC	Jump on no Carry	CY=0		
JP	Jump on positive	S=0		
JM	Jump on minus	S=1		
JZ	Jump on zero	Z=1		
JNZ	Jump on no zero	Z=0		
JPE	Jump on	P=1		

BASIC CONCEPTS IN PROCESSOR MICROARCHITECTURE

- Almost all modern processors use an instruction execution pipeline design.
- Simple processors issue only one instruction for each cycle; others issue many.

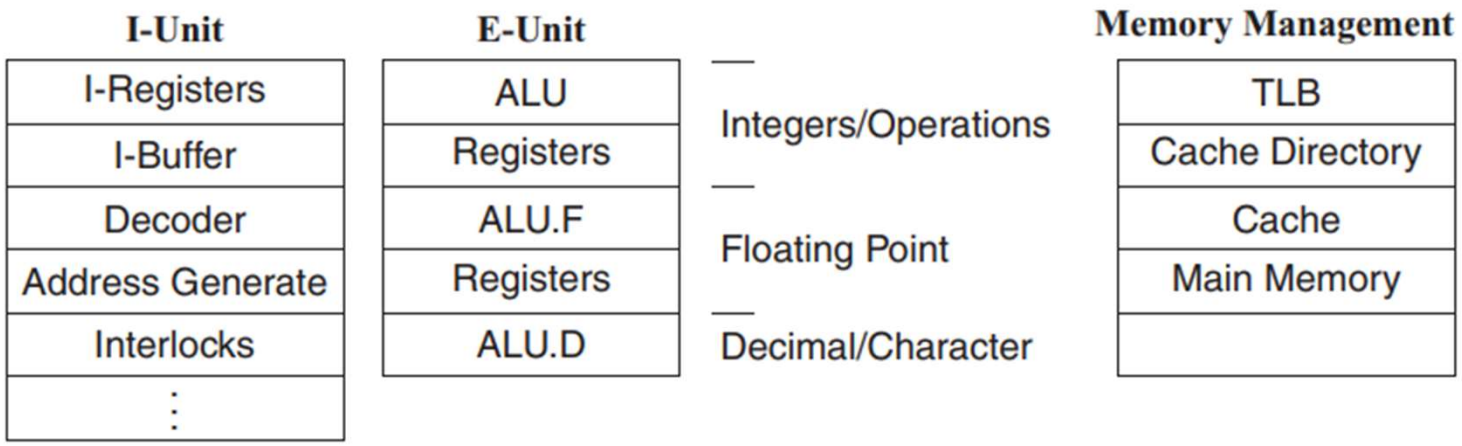
- Many embedded and some signal processors use a simple issue - one - instruction - per - cycle design approach.

- But the bulk of modern desktop, laptop, and server systems issue multiple instructions for each cycle.

BASIC CONCEPTS IN PROCESSOR MICROARCHITECTURE

- Every processor has a memory system, execution unit (data paths), and instruction unit.
- The faster the cache and memory, the smaller the number of cycles required for fetching instructions and data (IF and DF).
- The more extensive the execution unit, the smaller the number of execution cycles (EX).
- The control of the cache and execution unit is done by the instruction unit.

BASIC CONCEPTS IN PROCESSOR MICROARCHITECTURE



Processor units.

BASIC CONCEPTS IN PROCESSOR MICROARCHITECTURE

- The pipeline mechanism or control has many possibilities. Potentially, it can execute one or more instructions for each cycle.
- Instructions may or may not be decoded and/or executed in program order.
- Indeed, instructions from several different programs can be executed in the same cycle in multithreaded pipelines.
- Regardless of the type of pipeline, “ breaks ” or delays are the major limit on performance. Pipeline delays or breaks generally arise from one of three causes

BASIC CONCEPTS IN PROCESSOR MICROARCHITECTURE

1. ***Data Conflicts -***

   Unavailability of a Source Operand. This can occur for several reasons; typically, the current instruction requires an operand that is the result of a preceding uncompleted instruction.

2. ***Resource Contention-***

   Multiple successive instructions use the same resource or an instruction with a long execution time delays a successor instruction's execution. Additional resources (floating - point units, register ports, and out - of - order execution) contribute to reducing contention.

BASIC CONCEPTS IN PROCESSOR MICROARCHITECTURE

3. Run - On Delays (in Order Execution Only)

When instructions must complete the WB (writeback) in program order, any delay in execution (as in the case of multiply or divide) necessarily delays the instruction execution in the pipeline.

4. Branches

The pipeline is delayed because of branch resolution and/or delay in the fetching of the branch target instruction before the pipeline can resume execution.

Branch prediction, branch tables, and buffers all can be used to minimize the effect of branches.