Computer Architecture and Organization

Universidad Francisco de Vitoria UFV Madrid

TOPIC 2 Instruction Set Architecture (ISA)

Goals

General

- To know the basic characteristics of an assembly language and its corresponding machine language.
- To know the instruction set of a real processor and the methodologies for low-level programming with this ISA.

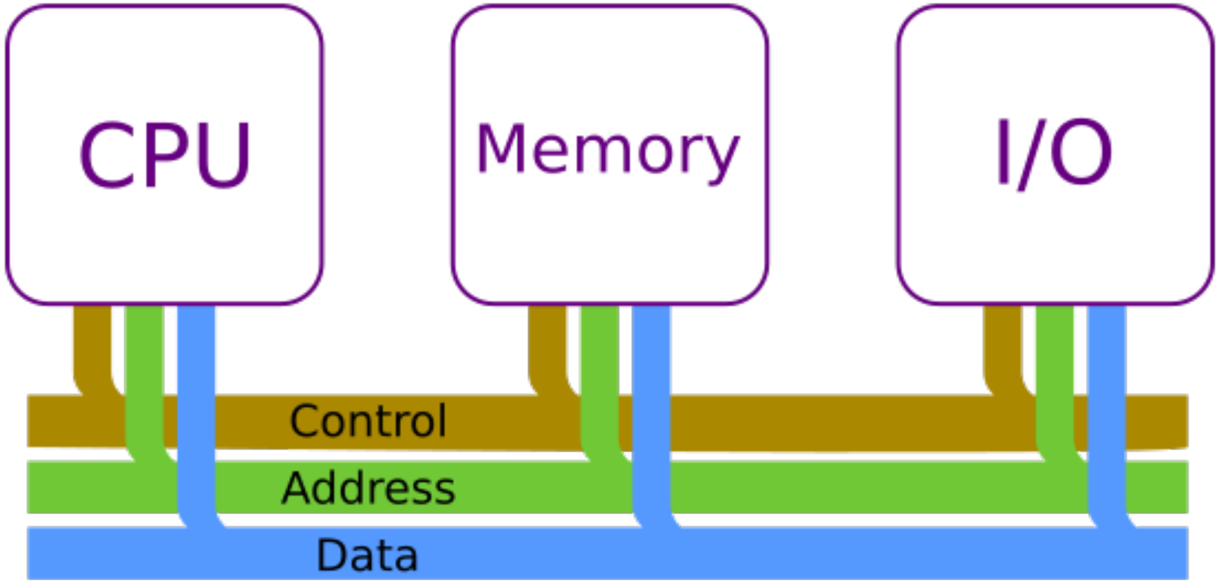
Specific

- List the characteristics of an instruction set
- Calculate the location of an operand for a given addressing mode
- Write simple assembly language programs for different machines.
- Explain the different fields that appear in the machine format of a machine instruction
- Interpret the meaning and simulate the execution of Z80 assembler programs
- State the differences between CISC and RISC approaches to instruction set design.
- Classify an instruction set as CISC or RISC

Contents

- Machine instruction characteristics
- Design elements of an ISA
- Operand types
- Operation types
- Addressing Modes
- Number of addresses
- Machine format
- CISC and RISC architecture philosophies

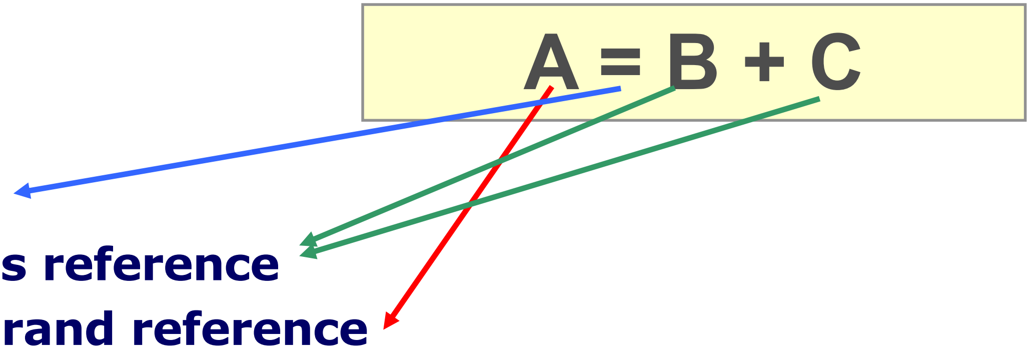
Von Neumann Architecture



Von Neumann Architecture refresh

Machine instructions characteristics

- **Machine instruction**: instruction directly executable by a processor
- **Instruction SET**: Set of all instructions that a specific processor can execute
- **Elements of a machine instruction**:



- **Operation Code**
- **Source operands reference**
- **Destination operand reference**
- **Next instruction reference**: implicit for most instructions. The next instruction is the one stored in the following memory location

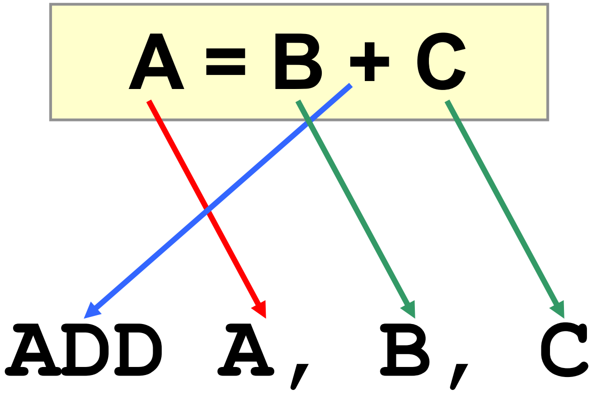
PC (**Program Counter**): internal register of the processor used to manage the implicit sequencing of instructions. It is initialized with the address of the first instruction and is incremented each time memory is accessed to read a new instruction.

Machine instructions characteristics

- **Instruction Encoding:** bit sequence, where certain groups (fields) represent each of the elements discussed before.
- **Instruction Format:** division of the bit sequence into fields. In most instruction sets there is more than one format.
- **Symbolic representations of machine instructions:** operation codes and operands are represented by symbols known as mnemonics.
- **Machine Language:** machine instruction set in binary
- **Assembly language:** set of machine instructions in symbolic representation.

Machine instructions characteristics

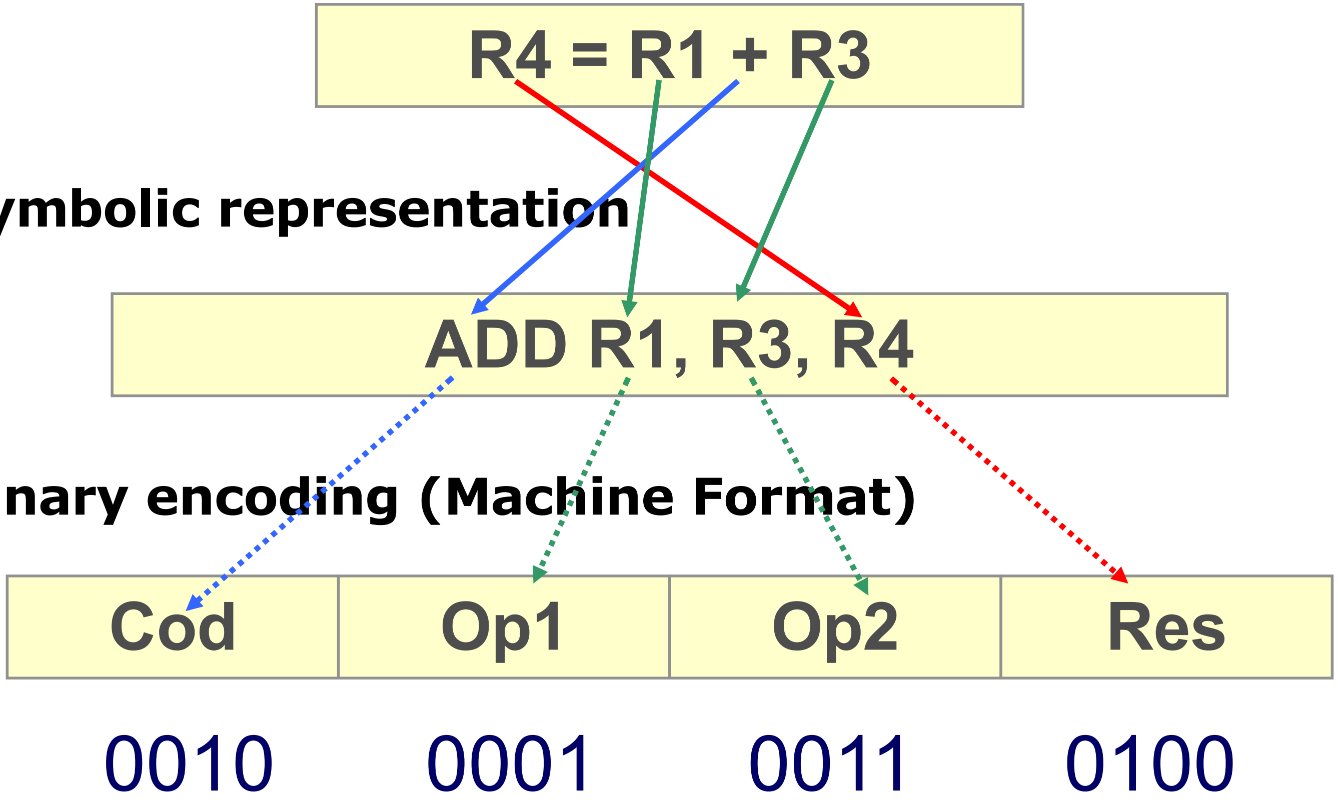
- A machine instruction must therefore indicate what operation it performs, on what data it performs it and where the result is stored.



- For this purpose, there are multiple alternatives, with different
  - Memory requirements
  - Program execution time

Machine instructions characteristics

- Example 1:
  - **High-level language instruction**



R4 = R1 + R3

  - **Symbolic representation**

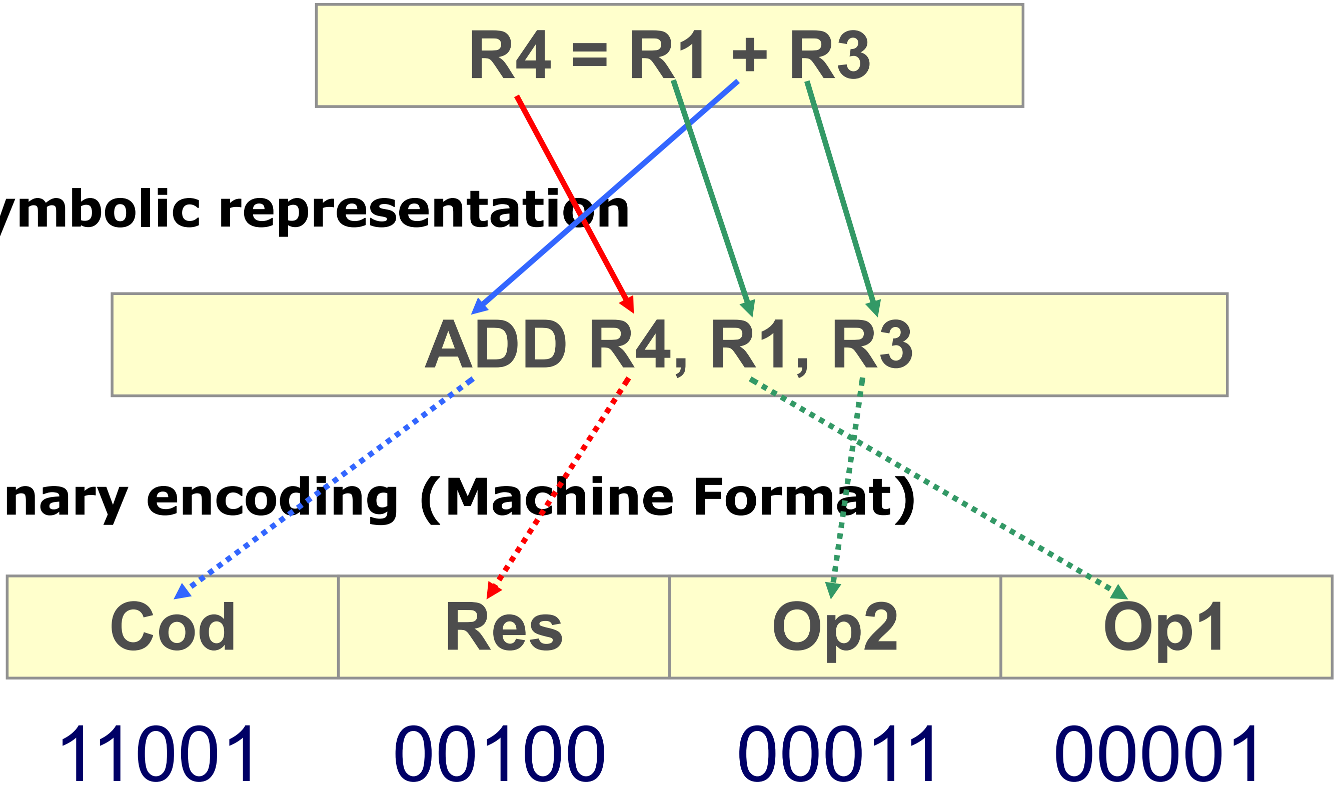
ADD R1, R3, R4

  - **Binary encoding (Machine Format)**

Cod	Op1	Op2	Res
0010	0001	0011	0100

Machine instructions characteristics

- Example 2:
  - **High-level language instruction**
  - **Symbolic representation**
  - **Binary encoding (Machine Format)**



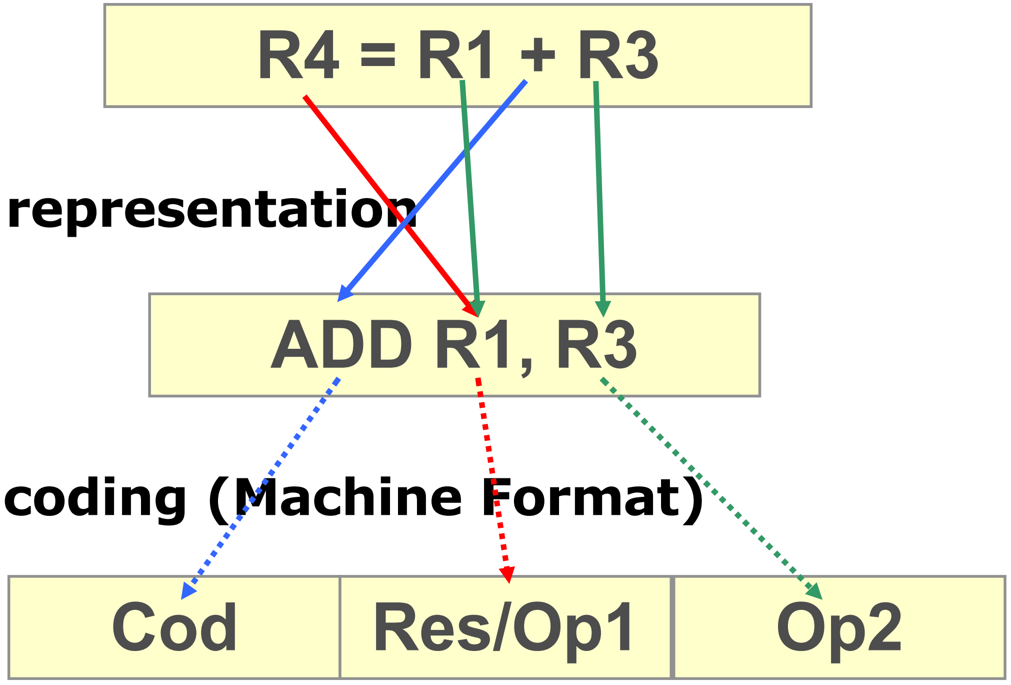
R4 = R1 + R3

ADD R4, R1, R3

Cod	Res	Op2	Op1
11001	00100	00011	00001

Machine instructions characteristics

- Example 3:
  - **High-level language instruction**



R4 = R1 + R3

  - **Symbolic representation**

ADD R1, R3

  - **Binary encoding (Machine Format)**

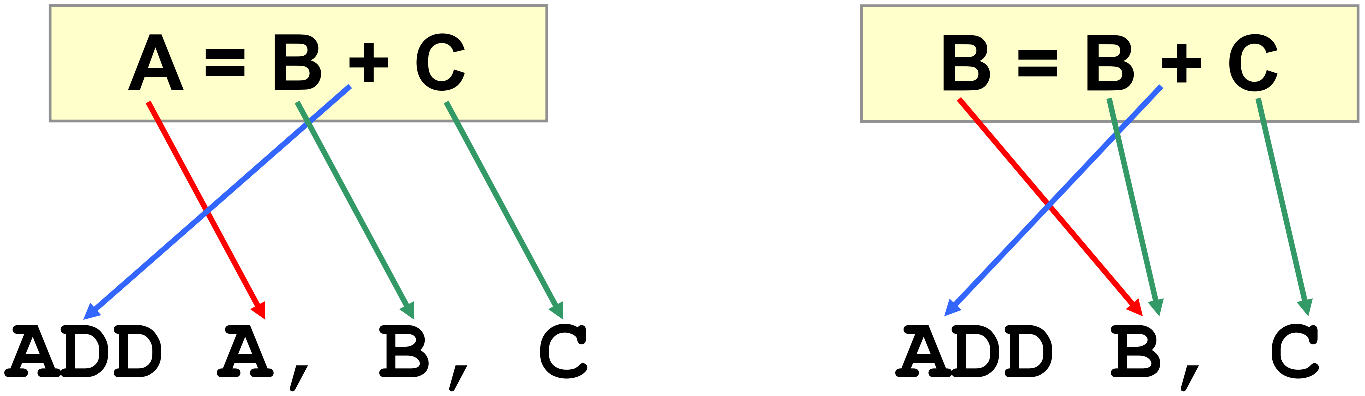
Cod	Res/Op1	Op2
00001	0001	0011

Design elements of an ISA

- There are **six basic elements** (decisions that are taken when designing a machine language) that differentiate some instruction sets from others :
  1. Number of addresses
  2. Operand location
  3. Addressing Modes
  4. Instruction number and types
  5. Data size and types
  6. Machine format

Design elements of an ISA

- Design Element 1:
  - **Number of addresses**: number of explicit references to operands in arithmetic or logical operations
  - ISAs may be of 0, 1, 2 and 3 addresses
  - Example: 3 and 2 addresses instructions



A = B + C

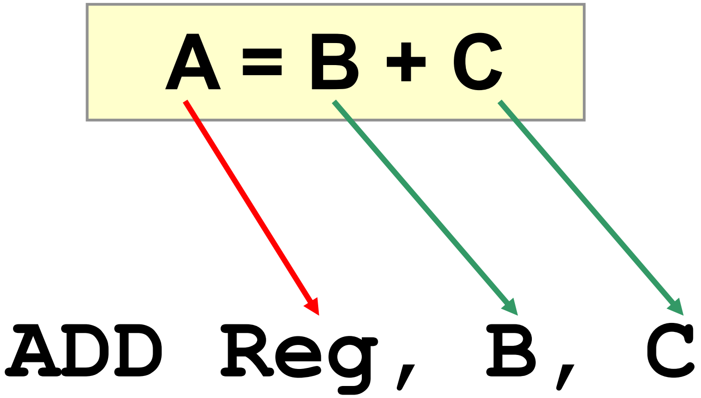
```
ADD  A,  B,  C
```

B = B + C

```
ADD  B,  C
```

Design elements of an ISA

- Design Element 2:
  - **Operand location**: the data to be operated on may be in memory, but also in internal registers of the processor or even in the instruction itself.
  - The location of the data is specified by using an **addressing mode.**



```
ADD Reg, B, C
```

Design elements of an ISA

- Design Element 3:
  - The location of the data is specified by using an **addressing mode**.
  - For accessing data in memory, there are multiple addressing modes, which differ in the size needed to be encoded in the instruction and the time it takes to fetch the data.

- Design Element 4:
  - **Instruction number and types**
    - Arithmetic instructions: ADD, SUB, MULT, DIV, ...
    - Logic instructions: AND, OR, COMP, ...
    - Data transfer instructions (also called Load/Store instructions): LOAD, STORE, MOVE,
    - Flow control instructions: JUMP, BEQ, BNE, ...
    - I/O Instructions
    - ...

Design elements of an ISA

- Design Element 5:
  - **Data size and types**:
    - Integers of 8, 16, 32, or 64 bits
    - Floating point of 32, 64 or 128 bits
    - Binary Coded Decimals (BCD)
    - ASCII Chars....

- Design Element 6:
  - The instructions in the ISA must be encoded in binary in what is called **machine format**, which can be of:
    - Fixed size
    - Variable size

Number of addresses (Design Element 1)

- **3-address architecture:**
  - The instruction explicitly specifies the location of the two operands and the result. Each one can therefore be in **three different places**.
  - Examples:

INSTRUCTION	MEANING
`ADD R, A, B`	`R= A + B`
`SUB R, A, B`	`R= A - B`
`MUL R, A, B`	`R= A * B`
`DIV R, A, B`	`R= A / B`

Note: Destination is commonly placed first, as per the syntax of the original formula

Number of addresses (Design Element 1)

- **2-address architecture**:
  - The instruction specifies two operands explicitly, one of which also acts as the result at the same time.
  - Shorter instructions, but less compact programs, since in each operation one of the operands is lost and it may be necessary to save it beforehand.
  - Examples:

INSTRUCTION	MEANING
ADD A, B	A= A + B
SUB C, D	C= C - D
MUL B, C	B= B * C
DIV B, A	B= B / A

Number of addresses (Design Element 1)

- **1-address architecture**:
  - Uses a special-purpose register called Accumulator (AC).
  - The instruction explicitly specifies the location of one operand; the other operand and the result are placed in the accumulator.
  - Shorter instructions but less compact programs. Since the accumulator can store a single piece of data, this architecture forces many data movements to perform the operations.
  - Examples:

INSTRUCTIONS	MEANING
ADD B	AC= AC + B
SUB B	AC= AC - B
XOR B	AC= AC XOR B (bitwise XOR)
AND B	AC= AC AND B (bitwise AND)

Z80 is a 1-address architecture for 8-bit operations.
The Accumulator Register is "A"

Number of addresses
(Design Element 1)

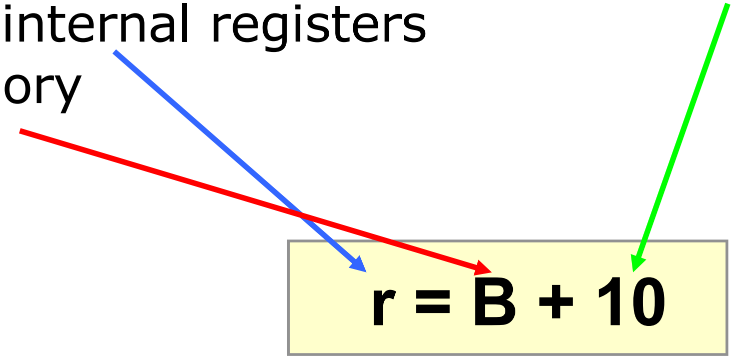
- **0-address architecture:**
  - Uses a **Stack structure** to temporarily store data.
  - All three operands must be placed in the Stack in a specific order, which may be different from one CPU to another.
  - It is not very flexible, and does not allow reusing operands, only intermediate calculations.
  - Examples:

INSTRUCTION	MEANING
ADD	TOS = TOS + (TOS-1)
SUB	TOS = TOS – (TOS-1)
MUL	TOS = TOS * (TOS-1)
DIV	TOS = TOS / (TOS-1)

* Note: TOS = ”Top Of Stack” upper element of the stack

Operand (data) location
(Design Element 2)

- Data can be found in three different places
  - In the very instruction itself (constant data)
  - In CPU internal registers
  - In Memory



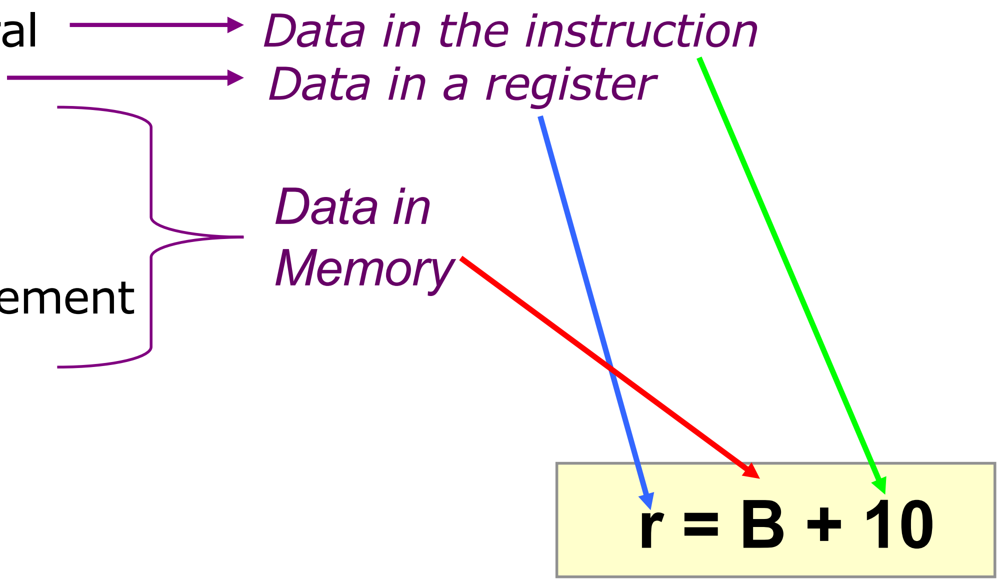
r = B + 10

- All ISAs handle data in any of these three locations. The difference from one language to another lies in the possible locations for the data on which arithmetic and logic operations are performed:
  - Some languages only allow to operate with data that are stored in the internal registers of the processor (they must be fetched from memory in advance).
  - Other languages allow to operate with data stored in memory (Operands are retrieved from memory just when they are needed for the operation).

Addressing modes (Design Element 3)

- **Addressing mode**: mode in which the **effective address** of an operand is specified, this is, its location.

- **Modes**:
  - Immediate or literal
  - Register
  - Direct
  - Indirect
  - Register Indirect
  - Indirect + Displacement
  - Etc.



Data in the instruction

Data in a register

Data in Memory

r = B + 10

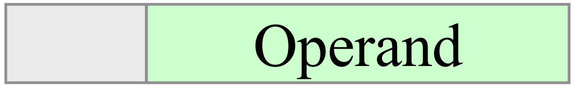
Addressing Modes (Design Element 3)

- **Immediate or Literal**:
  - Data is located in the very instruction, so it is a constant
  - It usually does not require an extra memory access to get the data
  - The range of representable data depends on the length of the corresponding field in the instruction, and on the data type
  - **Design question: How many bits dedicated to this field?**
    - Integer number range with n bits: $[-2^{n-1}, 2^{n-1}-1]$
    - Natural number range with n bits: $[0, 2^{n}-1]$
  - Examples:

LD A, 0xF3

JP 0x80BF

AND 0x07



Addressing Modes (Design Element 3)

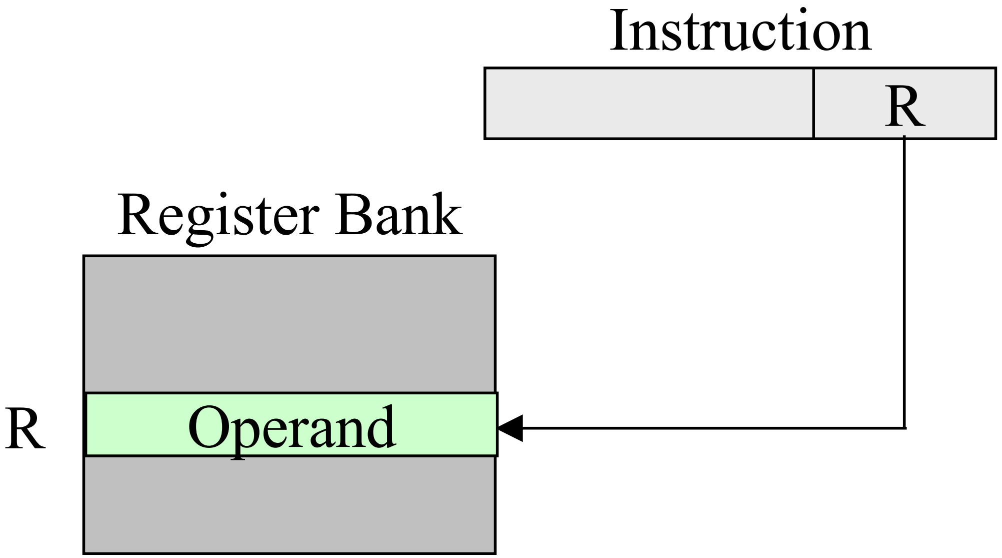
- **Register**:
  - Data is stored in a CPU register
  - The instruction includes the identifier for the used register (Since there are just a few registers, a small number of bits are needed for register-id)
  - Memory access is not required
  - Data access is very fast (registers are inside the CPU)

  - Examples:

    LD A, C

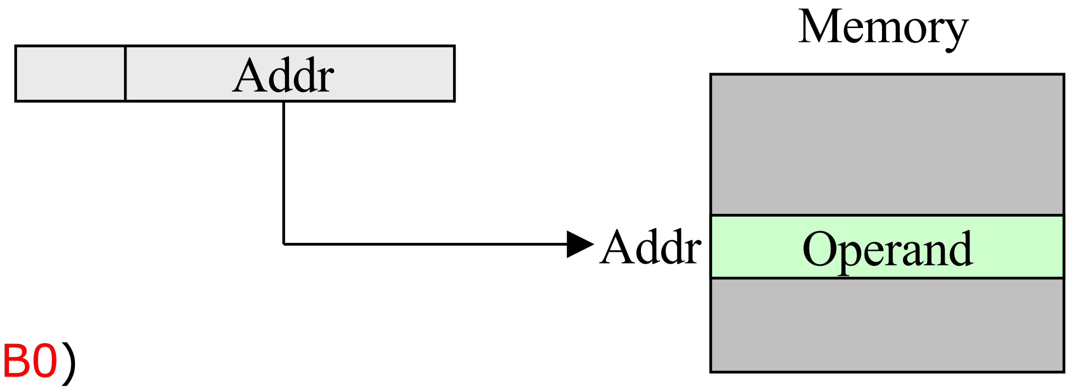
    JP (HL)

    AND B



Addressing Modes (Design Element 3)

- **Direct** (or absolute):
  - Requires just one access to memory
  - The instruction contains the address of the memory location where the data is located
  - Reachable memory range: 2^k, *k* is the lenght in bits of the field dedicated to address

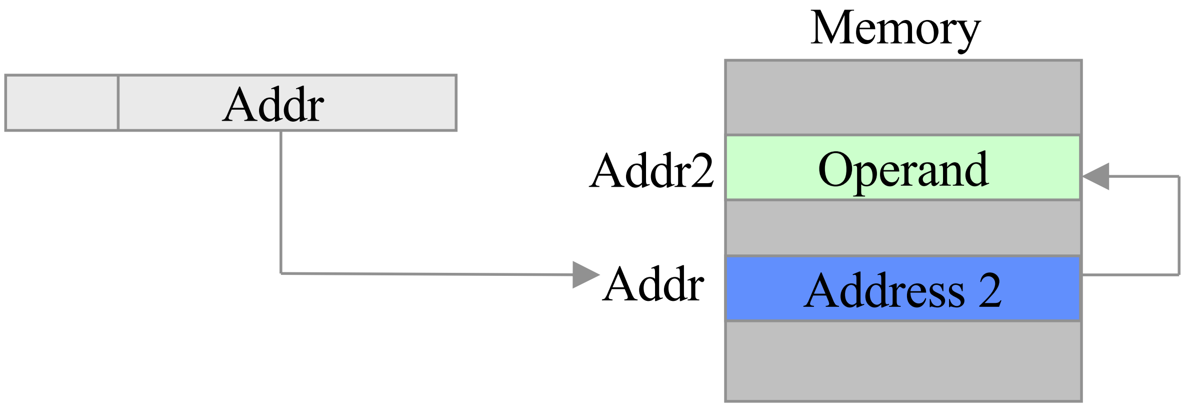


  - Example:

    LD A, (0x41B0)

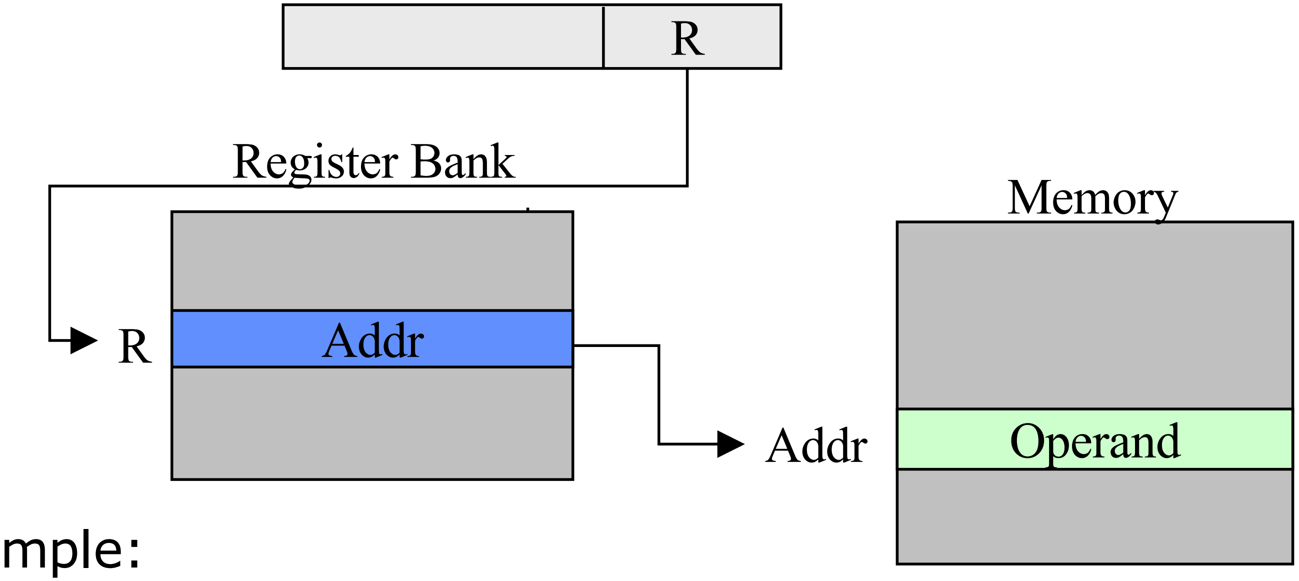
Addressing Modes (Design Element 3)

- **Indirect (also called Memory Indirect)**:
  - Requires two accesses to memory
  - The instruction contains the address of the memory location where the address of the data is located. So it contains the address of a pointer.
  - Memory range in which pointers can be located: 2^k, *k* is the lenght in bits of the Addr field in the instruction.



Addressing Modes
(Design Element 3)

- **Register Indirect**:
  - Requires one register access and one memory access
  - the instruction contains the register number where the data address (pointer) is found
  - Range of reachable memory: 2^N, N is the bit width of the register

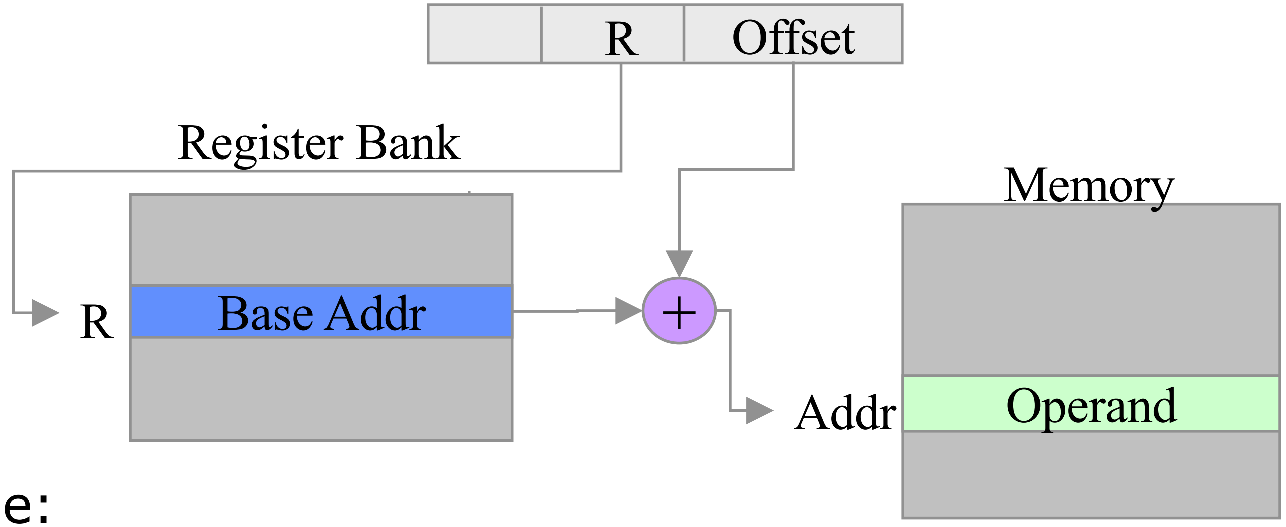


  - Example:

    LD A, (HL)

Addressing Modes
(Design Element 3)

- **Base plus Displacement**:
  - The instruction contains two fields: a register number and a constant called offset. The memory address of the data is calculated as the sum of this constant plus the register contents.
  - Requires one Access to a register and one memory access
  - Range of reachable memory: 2^N+k , *N* is the bit width of the register, k is the range of the offset.



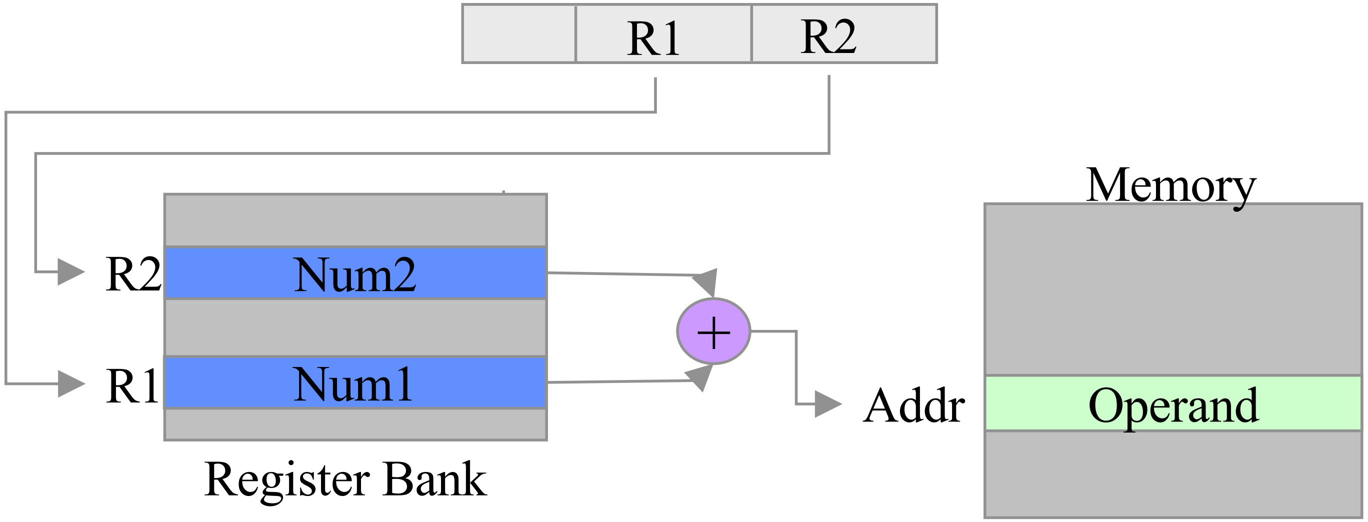
  - Example:

    LD A, **(IX+34)**

Addressing Modes
(Design Element 3)

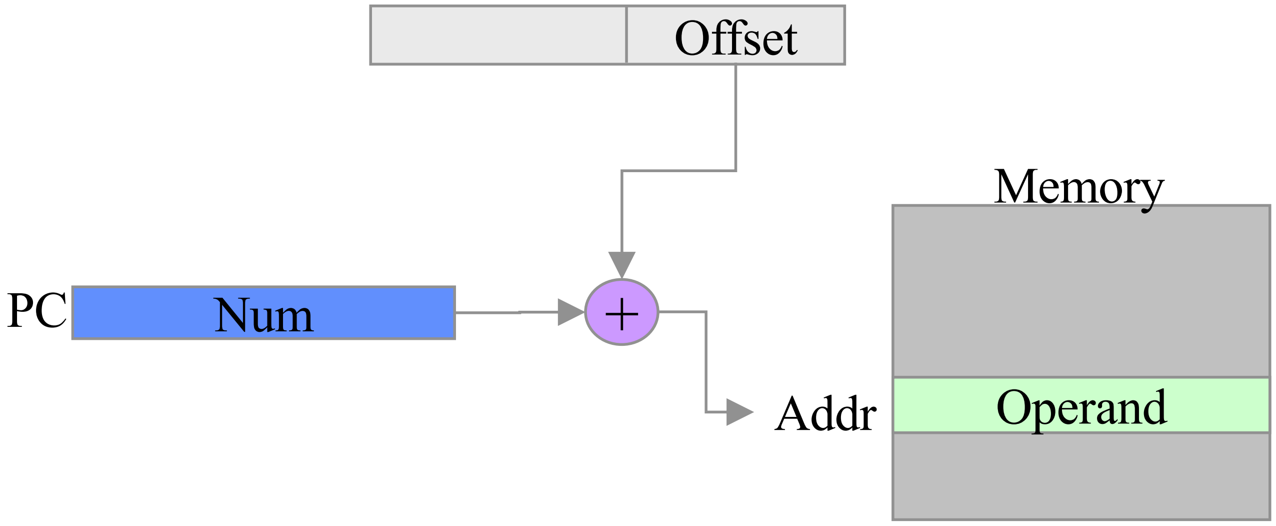
- **Indexed**:
  - The instruction contains two register fields. The memory address of the data is calculated as the sum of the contents of both registers.
  - Requires two accesses to the registers
  - Example (mode not available in Z80):

    Add R3, **(R1+R2)**



Addressing Modes (Design Element 3)

- **Relative or PC Relative**:
  - The instruction contains a field with a constant called offset. The memory address of the data is calculated as the sum of this constant plus the contents of the PC (program counter)..
  - Requires an access to a specific register (PC)
  - It is very important for **relocatable code**



  - Examples:

    JR +45

    DJNZ -10

Instruction Types
(Design Element 4)

Common types for all ISAs:

- **Data Transfer**: copy a piece of data from one location to another.
- **Arithmetic**: addition, subtraction, multiplication and division for natural numbers, fixed-point integers, floating-point reals and decimals (in BCD)
- **Logic**: AND, OR, NOT, XOR, Bitwise shift and rotation
- **I/O**: for data transfer between the CPU and peripheral devices
- **Flow Control**: allow the implicit sequencing to be changed
  - Conditional branch
  - Unconditional branch
  - Subroutine call

Instruction Types (Design Element 4)

- Examples: Some common assembly mnemonics (Z80)

  - **Data Transfer**

    MOVE STORE LOAD

    CLEAR **SET** **LD**

  - **Arithmetic**

    ADD **SUB** MULT DIV

  - **Logic**

    AND **OR** **NOT** **XOR**

    CMP SHIFT ROTATE **RR**

  - **Flow Control**

    JUMP BEQ BNQ BGT

    BGE BLT BLE **JR**

Data Types (Design Element 5)

Machine code instructions usually deal with:

- **Memory addresses**: Unsigned integer numbers (Naturals)
- **Data:**
  - **Numeric**:
    - Integers of several sizes: 8 bit, 16 bit, 32 bit, …
      - *Unsigned*: natural numbers in binary
      - Signed: Two’s complement binary numbers
    - Floating Point Real numbers (IEEE 754 format), 32 bit, 64 bit, …
    - Binary Coded Decimals (BCD)
  - **Characters: letters, numbers and other symbols represented in a specific code** (ASCII, Unicode, …)
- **Logic data**
  - Any data when manipulated bitwise (bit by bit)

Machine Format
(Design Element 6)

Two alternatives for encoding (machine format)

- **Variable size** instructions:
  - Normally used when there are many different instructions and many addressing modes

Op + Addr Num	Addr Mode 1	Operand 1	•••	Addr Mode n	Operand n

- **Fixed size** instructions:
  - Normally used when only a few instructions and addressing modes are available

Op + Addr num + mode	Operand 1	Operand 2	Operand 3

Current ISAs

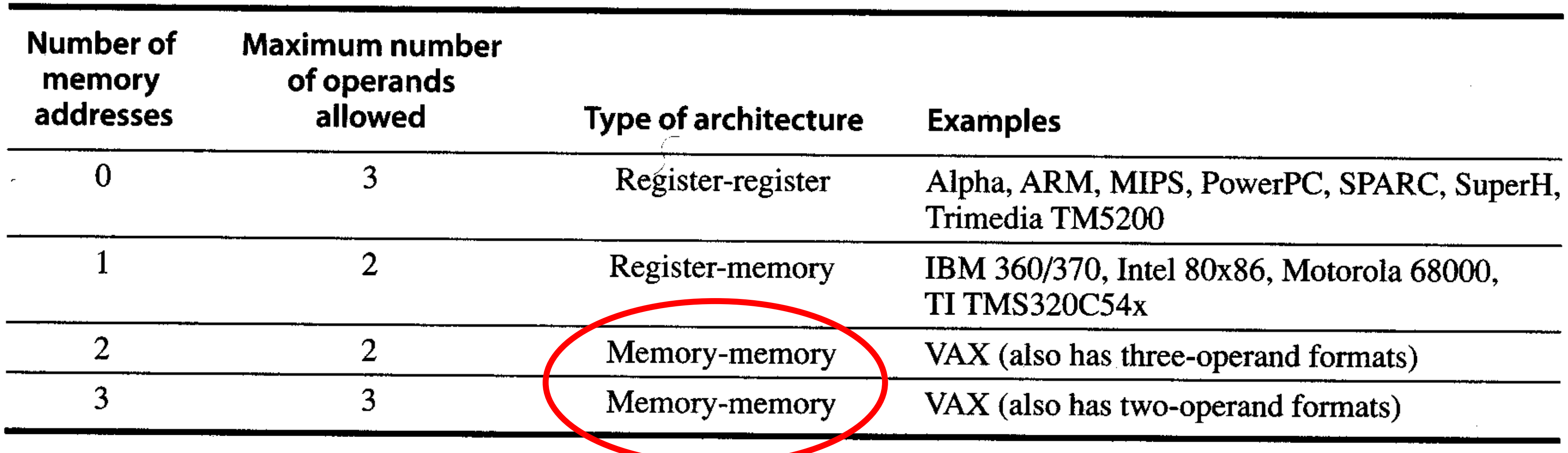
- Since the 1980s, general purpose processors (used in PCs, servers, etc.) have only implemented 2-address and 3-address architectures, called **General Purpose Register** Architectures (GPR).

- The first computers (EDSAC, UNIVAC I, etc.) had **accumulator** type architectures, i.e. 1-address. Today, accumulator architectures are present in some microcontrollers used in embedded systems.

- In the 1960s and 1970s there were several computers with **stack** architectures (0-address). Today, most processors incorporate a stack and instructions to access to it.

Current ISAs

- **3-Address**
  - They operate with data in registers only
  - Examples: RISC-V, ARMv8, PowerPC by IBM-Apple-Motorola, 88000 by Motorola, etc.
- **2-Address**
  - Operate with data in registers and with data in memory
  - Up to a single memory access is allowed in each instruction.
  - Examples: Z80 by Zilog, x86 by Intel, 68000 by Motorola, etc.



Number of memory addresses	Maximum number of operands allowed	Type of architecture	Examples
0	3	Register-register	Alpha, ARM, MIPS, PowerPC, SPARC, SuperH, Trimedia TM5200
1	2	Register-memory	IBM 360/370, Intel 80x86, Motorola 68000, TI TMS320C54x
2	2	Memory-memory	VAX (also has three-operand formats)
3	3	Memory-memory	VAX (also has two-operand formats)

Not in any current ISA

CISC vs. RISC Architectures

- There are two opposing philosophies for the definition of processor instruction set architectures (ISA): CISC (Complex Instruction Set Computer) and RISC (Reduce Instruction Set Computer).

- CISC architectures provide a very extensive and rich machine language, with many different instruction types and addressing modes for data access.

- RISC architectures, on the other hand, provide a much simpler machine language. They appeared much later than CISC (1980s), with the aim of simplifying processor microarchitectures and thus increasing their performance.

CISC vs. RISC Architectures approach

CISC Architecture

- Variable size instructions
- Many instructions and addressing modes
- Any instruction may obtain operands from memory
- Reduced register bank
- Complexity resides in the microcode of the CPU

IBM 360, IA32, IA64, M68k, Z80

RISC Architecture

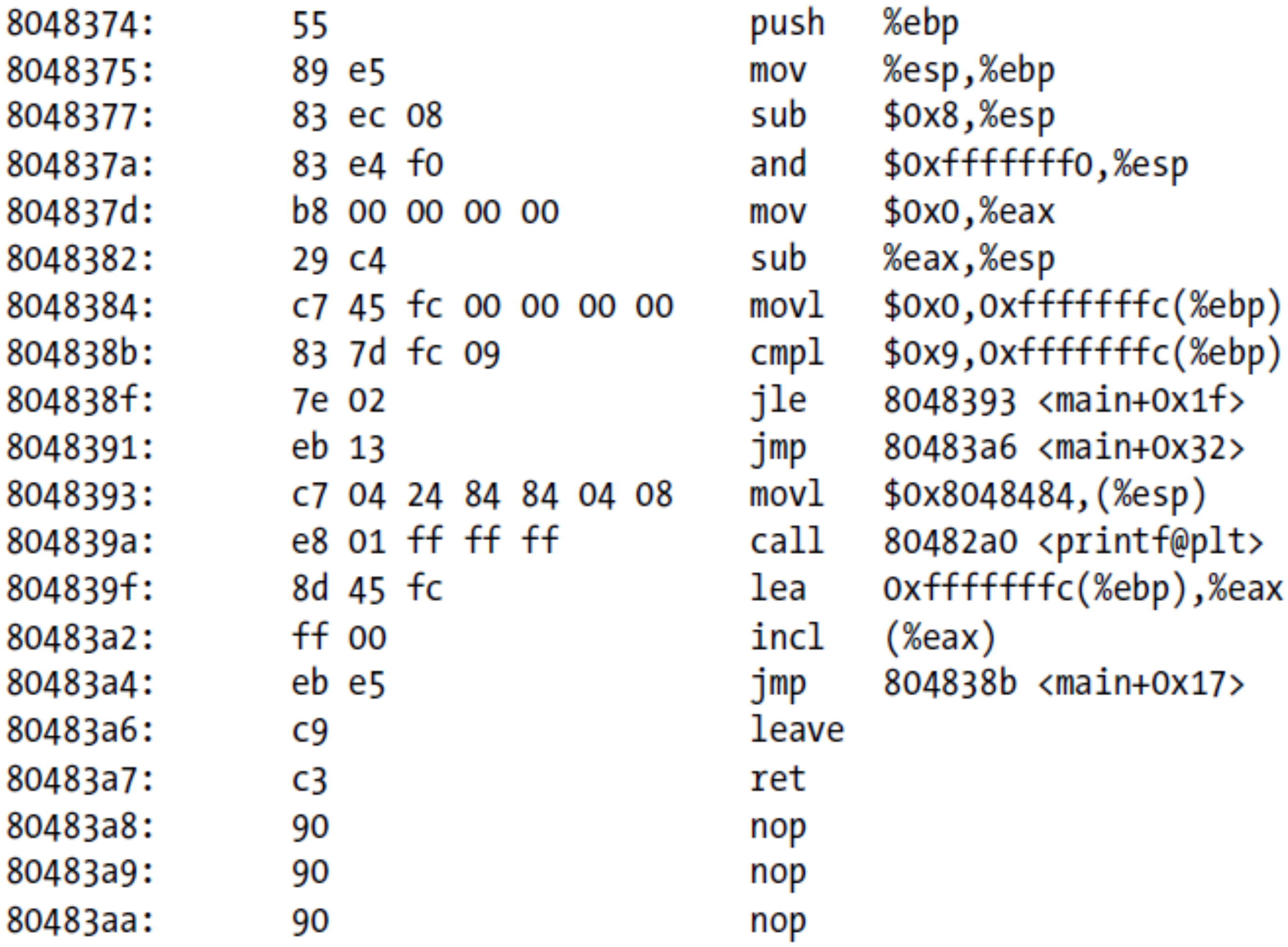
- Fixed size instructions
- Small number of instructions and addressing modes
- Only Load/Store instructions can access to main memory
- Many registers
- Complexity resides in the compiler

PA-RISC, PowerPC, Sparc, M88k, MIPS, ARMv8, RISC-V

What does this mean?

CISC vs. RISC Architectures

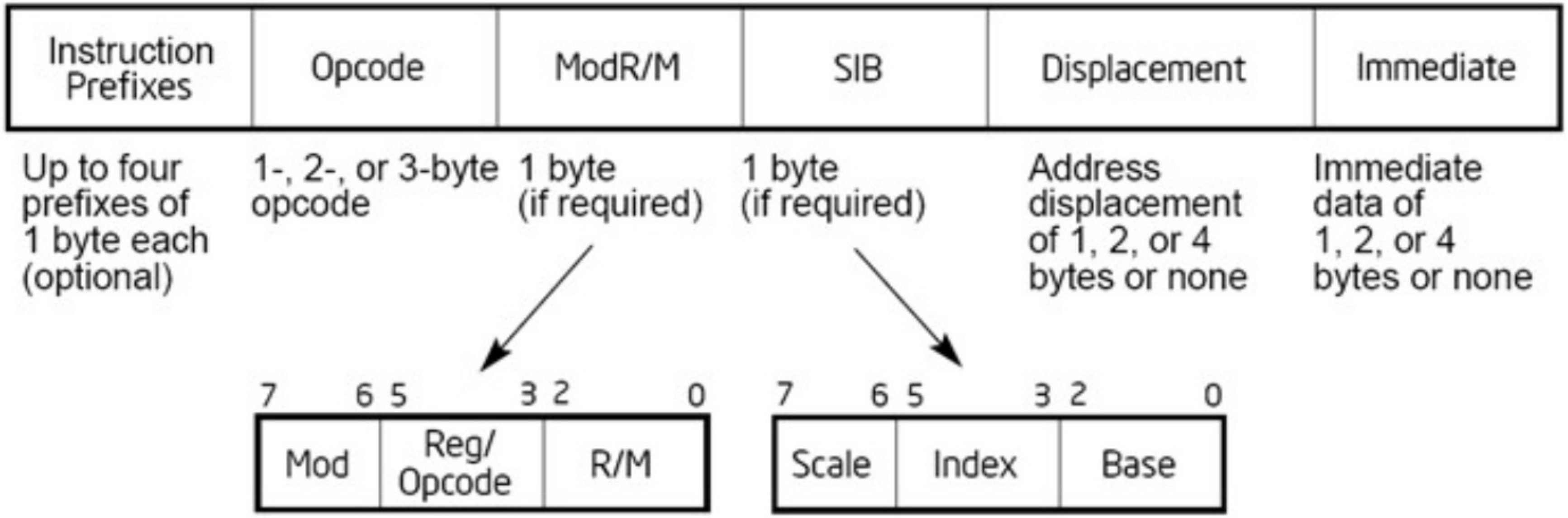
- Example: Machine Code and Assembly IA-32



```
8048374:    55                      push   %ebp
8048375:    89 e5                   mov    %esp,%ebp
8048377:    83 ec 08                sub    $0x8,%esp
804837a:    83 e4 f0                and    $0xfffffff0,%esp
804837d:    b8 00 00 00 00          mov    $0x0,%eax
8048382:    29 c4                   sub    %eax,%esp
8048384:    c7 45 fc 00 00 00 00    movl   $0x0,0xfffffffc(%ebp)
804838b:    83 7d fc 09             cmpl   $0x9,0xfffffffc(%ebp)
804838f:    7e 02                   jle    8048393 <main+0x1f>
8048391:    eb 13                   jmp    80483a6 <main+0x32>
8048393:    c7 04 24 84 84 04 08    movl   $0x8048484,(%esp)
804839a:    e8 01 ff ff ff          call   80482a0 <printf@plt>
804839f:    8d 45 fc                lea    0xfffffffc(%ebp),%eax
80483a2:    ff 00                   incl   (%eax)
80483a4:    eb e5                   jmp    804838b <main+0x17>
80483a6:    c9                      leave
80483a7:    c3                      ret
80483a8:    90                      nop
80483a9:    90                      nop
80483aa:    90                      nop
```

CISC vs. RISC Architectures

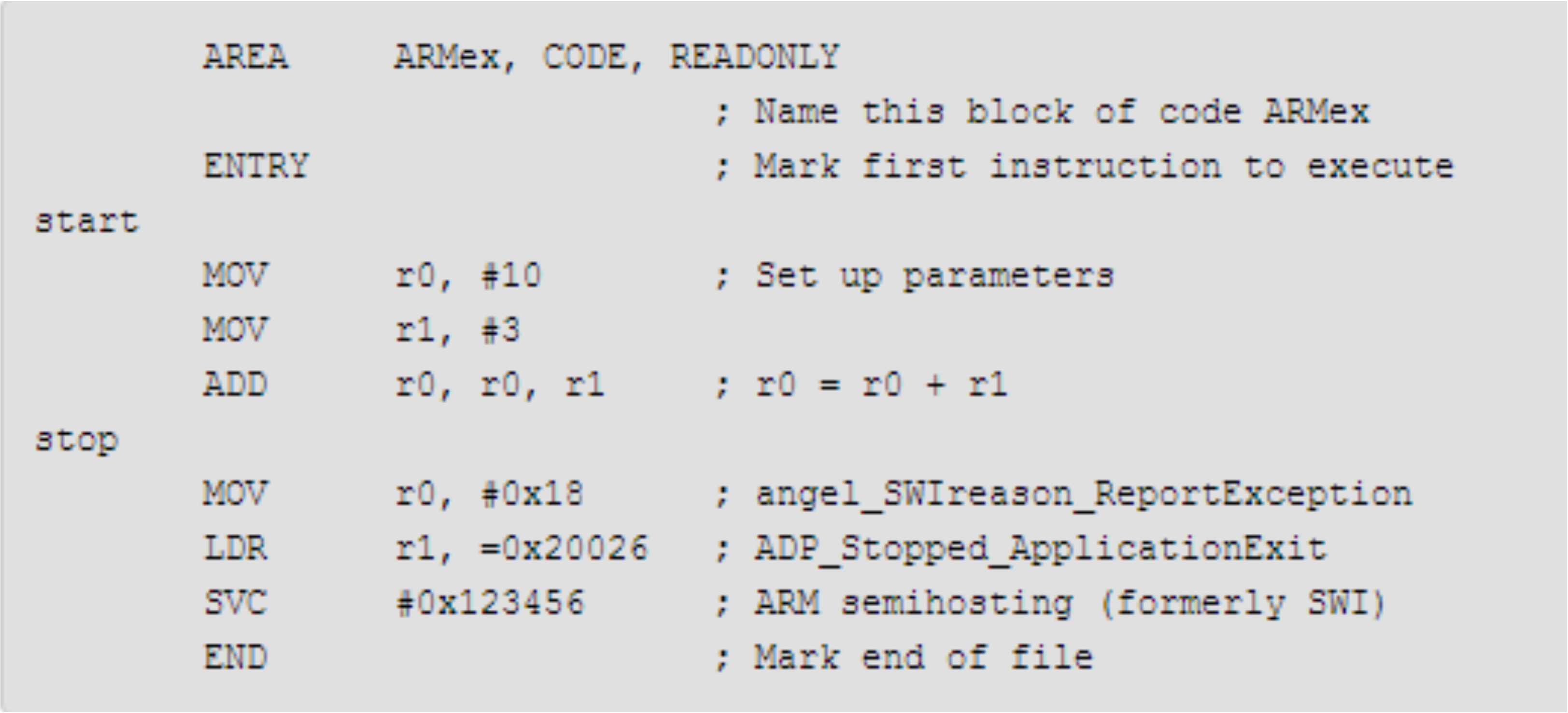
- Example: machine format IA-32



Instruction Prefixes	Opcode	ModR/M	SIB	Displacement	Immediate
Up to four prefixes of 1 byte each (optional)	1-, 2-, or 3-byte opcode	1 byte (if required)	1 byte (if required)	Address displacement of 1, 2, or 4 bytes or none	Immediate data of 1, 2, or 4 bytes or none

CISC vs. RISC Architectures

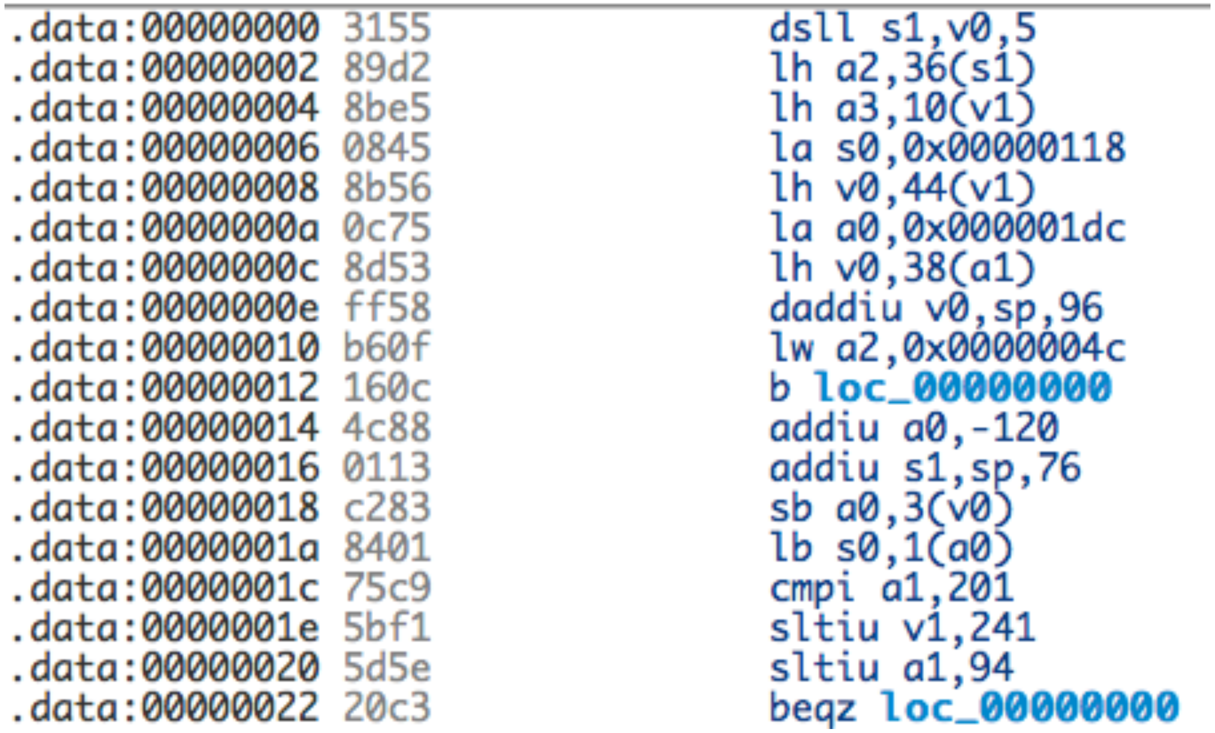
- Example: ARM Assembly



```
        AREA    ARMex, CODE, READONLY
                                ; Name this block of code ARMex
        ENTRY                   ; Mark first instruction to execute
start
        MOV     r0, #10         ; Set up parameters
        MOV     r1, #3
        ADD     r0, r0, r1      ; r0 = r0 + r1
stop
        MOV     r0, #0x18       ; angel_SWIreason_ReportException
        LDR     r1, =0x20026    ; ADP_Stopped_ApplicationExit
        SVC     #0x123456       ; ARM semihosting (formerly SWI)
        END                     ; Mark end of file
```

CISC vs. RISC Architectures

- Example MIPS 16: Machine Code and Assembly



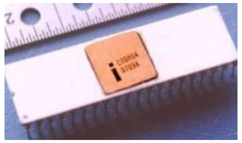
```
.data:00000000 3155                 dsll s1,v0,5
.data:00000002 89d2                 lh a2,36(s1)
.data:00000004 8be5                 lh a3,10(v1)
.data:00000006 0845                 la s0,0x00000118
.data:00000008 8b56                 lh v0,44(v1)
.data:0000000a 0c75                 la a0,0x000001dc
.data:0000000c 8d53                 lh v0,38(a1)
.data:0000000e ff58                 daddiu v0,sp,96
.data:00000010 b60f                 lw a2,0x0000004c
.data:00000012 160c                 b loc_00000000
.data:00000014 4c88                 addiu a0,-120
.data:00000016 0113                 addiu s1,sp,76
.data:00000018 c283                 sb a0,3(v0)
.data:0000001a 8401                 lb s0,1(a0)
.data:0000001c 75c9                 cmpi a1,201
.data:0000001e 5bf1                 sltiu v1,241
.data:00000020 5d5e                 sltiu a1,94
.data:00000022 20c3                 beqz loc_00000000
```

Intel ISA Architecture Evolution

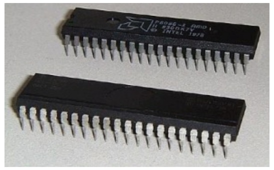
Architecture Evolution

- Example: Intel microprocessors

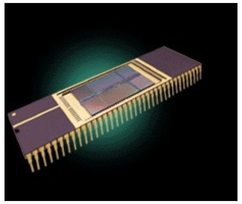
1971: **4004**
4 bits



1974: **8080**
8 bits



1978: **8086**
16 bits



1982: **80286**
16 bits

1985: **80386**
32 bits (IA32)

2001: **Itanium**

IA-64

2004: **Pentium 4, Xeon**

64 bits (EM64T)

Intel ISA Architecture Evolution

Current Intel Processor

Introduced 2017: **Core i9**

(high-performance, high core count x86 desktop processor)

64 bits (Skylake)

Public Datasheet: https://cdrdv2.intel.com/v1/dl/getContent/743844?explicitVersion=true