William Stallings Computer Organization and Architecture

Chapter 11

Instruction Sets:

Addressing Modes and Formats

Reference to operands

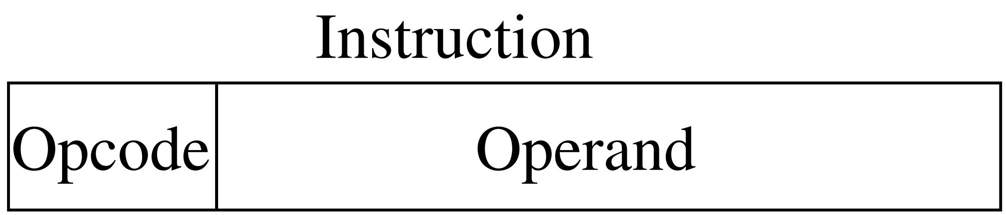
Opcode	Address Field 1	Address Field 2

- How to interpret address field values ?
- Example:
  - LOAD B can be interpreted as
    - Write into the accumulator the value B
    - Write into the accumulator the value contained in register B
    - Write into the accumulator the value contained in the memory cell with address B
    - …

Addressing Modes

- Immediate
- Direct
- Indirect
- Register
- Register Indirect
- Displacement (Indexed)
- Stack

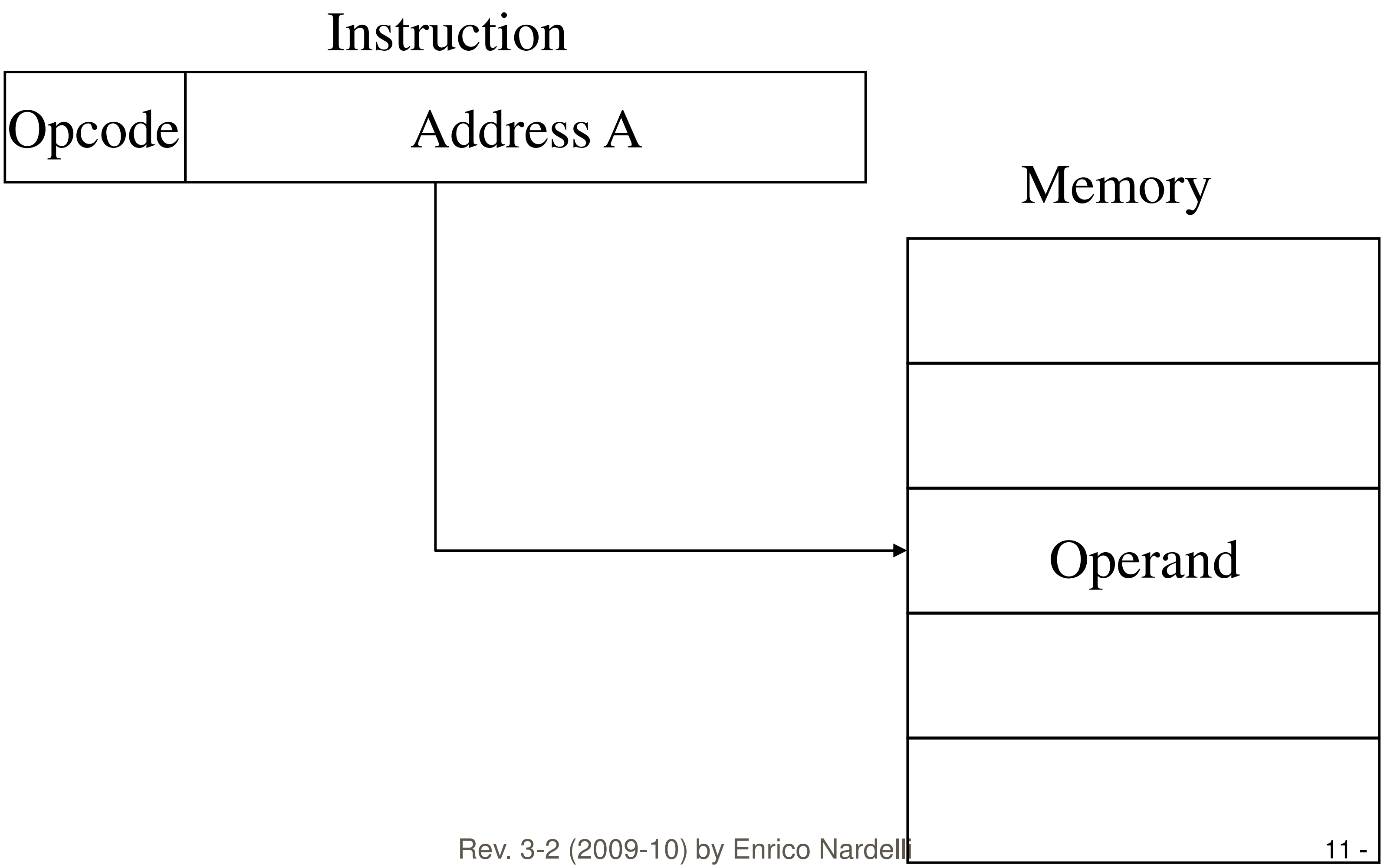
Immediate Addressing Diagram



Immediate Addressing

- Operand is part of instruction
- The value of address field is the operand
- e.g. ADD 5
  - Add 5 to contents of accumulator
  - 5 is operand
- No memory reference to fetch data
- Fast
- Limited range

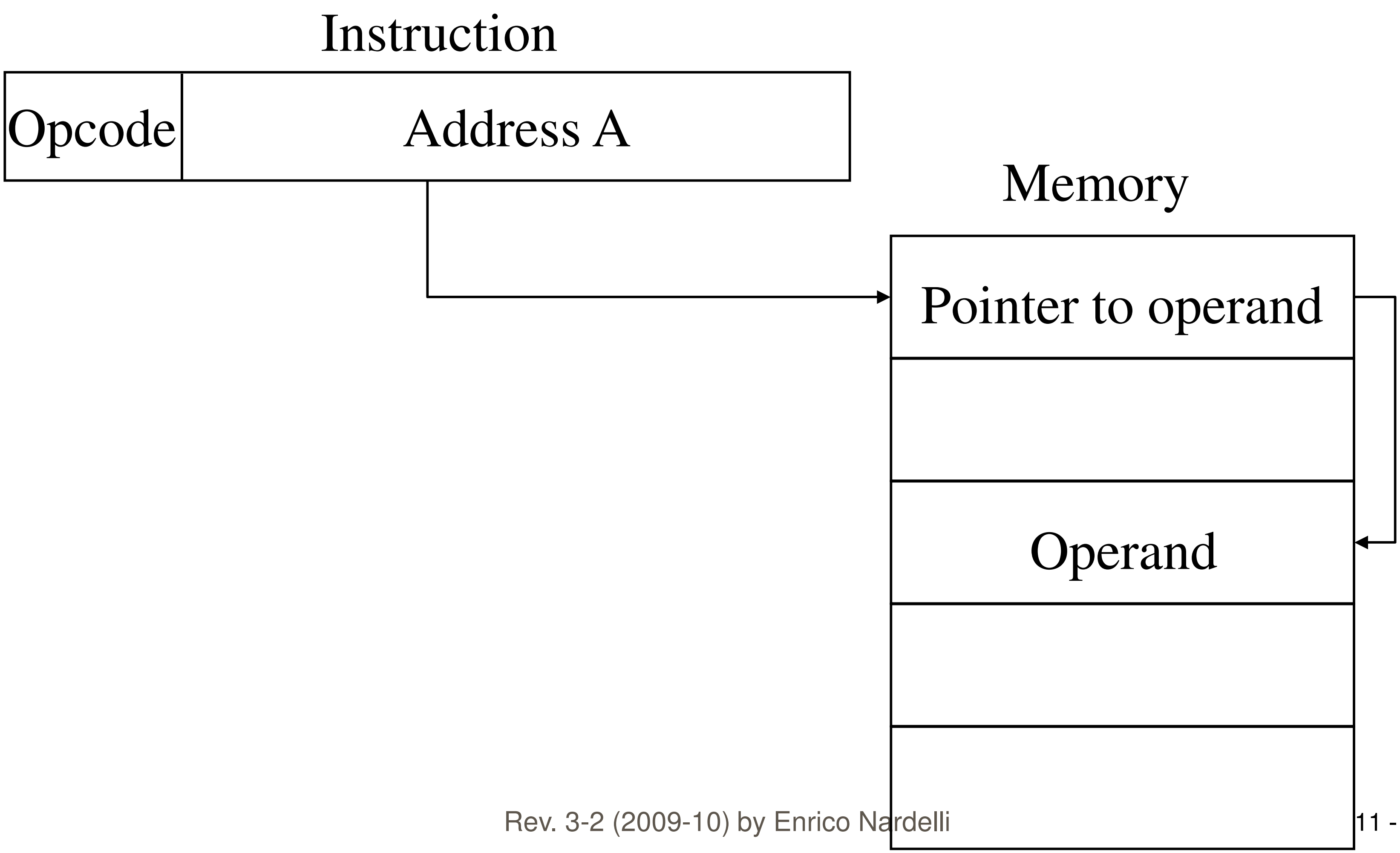
Direct Addressing Diagram



Direct Addressing

- The value of address field is the **address** of the operand
- If A is the value then (A) denotes the value contained in the memory cell with address A
- e.g. ADD @5
  - @ indicates the following values is an address
  - Look in memory at address 5 for operand
  - Add contents of cell 5 to accumulator: Acc+(5)→Acc
- Single memory reference to access data
- No additional calculations to work out effective address
- Limited address space

Indirect Addressing Diagram



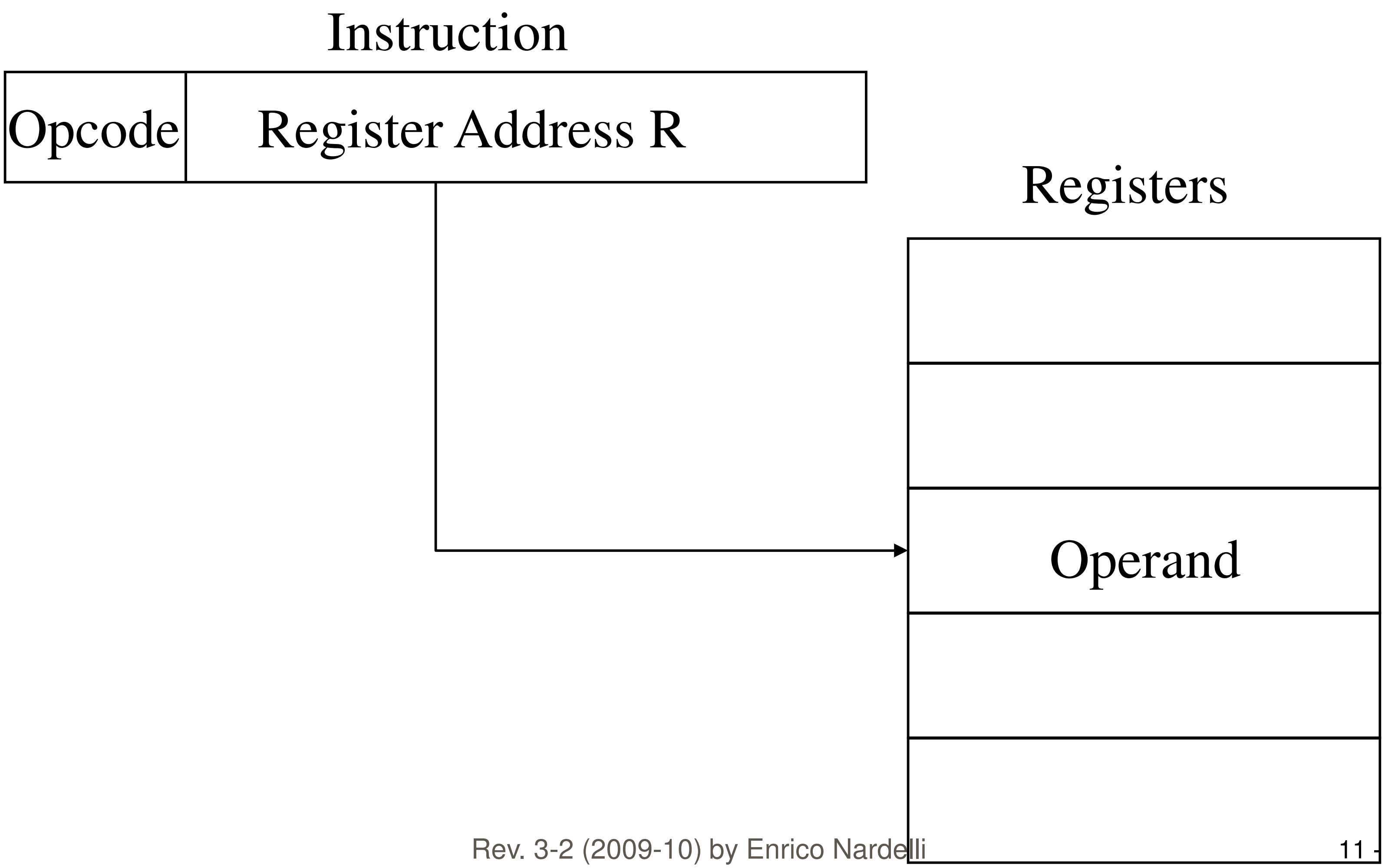
Indirect Addressing (1)

- The memory cell referenced by the address field contains the address of (i.e., the pointer to) the operand
- Let EA denote the Effective Address in memory of the operand
- If A is the value of the address field, then EA=(A)
- e.g. ADD (@5)
  - Look at address 5, then go to address (5) and look there for operand
  - Add to accumulator the content of the cell pointed to by the content of 5 (i.e., add the content of the cell at address (5))
  - $Acc+((5))\rightarrow Acc$

Indirect Addressing (2)

- Large address space
- 2^n addressable cells where n is the number of bits in the memory cell
- May be nested, multilevel, cascaded
  - e.g. EA = (((A)))
    - Draw the diagram yourself
- Multiple memory accesses to find operand
- Hence slower

Register Addressing Diagram



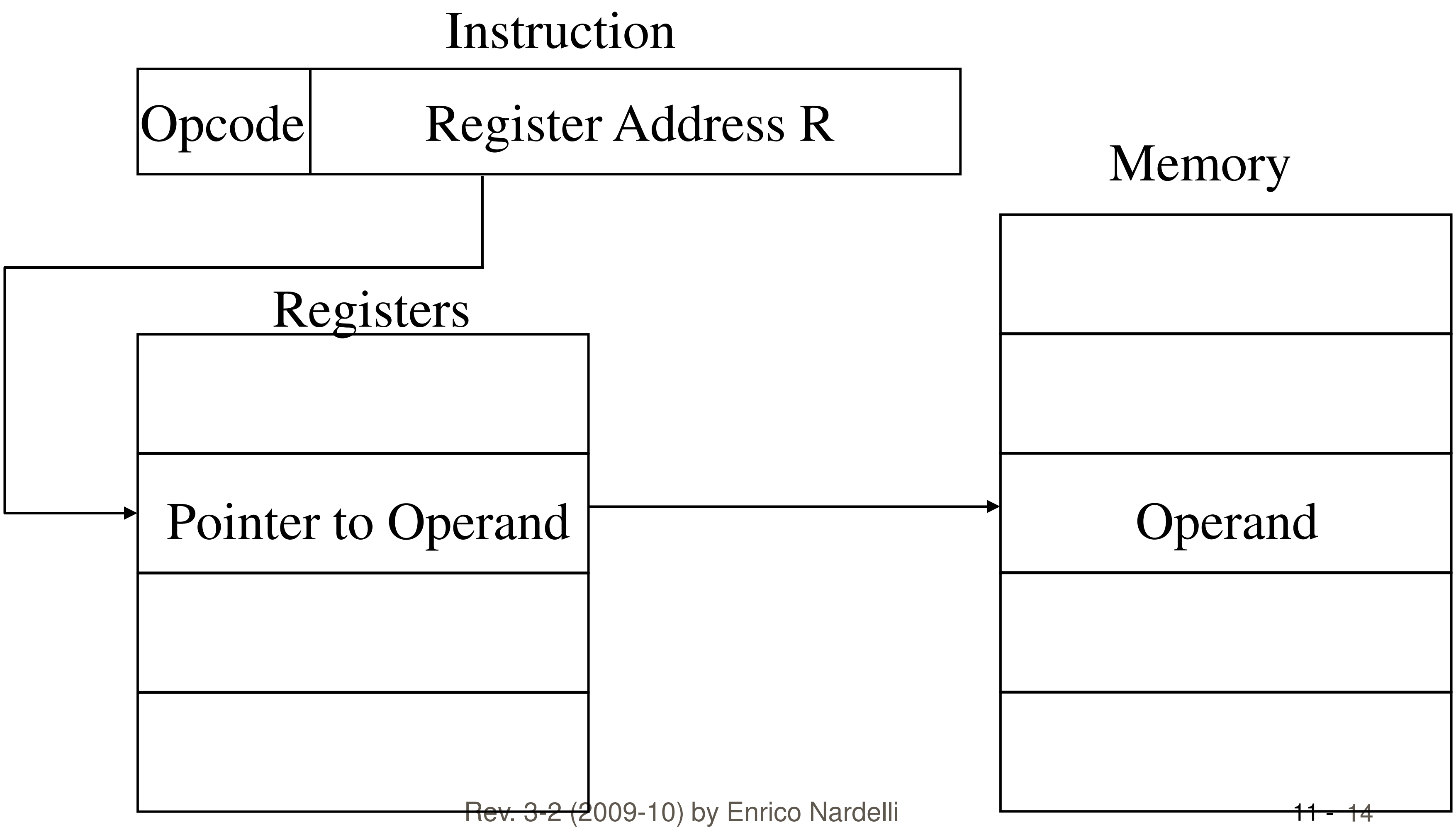
Register Addressing (1)

- Operand is contained in the register named in the address field
- If R is the register name then EA = R
- Since there is a limited number of registers, then a very small address field is needed
  - Shorter instructions
  - Faster instruction fetch
- e.g. ADD rA
  - Look into register A for operand
  - Add content of register A to accumulator
  - Acc+(rA)$\rightarrow$Acc

Register Addressing (2)

- No main memory access
- Very fast execution
- Very limited address space (= # registers)
- Multiple registers may help performance
  - Requires good assembly programming or compiler writing
  - Example: C programming
    - register int a;
- Conceptually similar to direct addressing…
- But operations on registers require fewer clock cycles

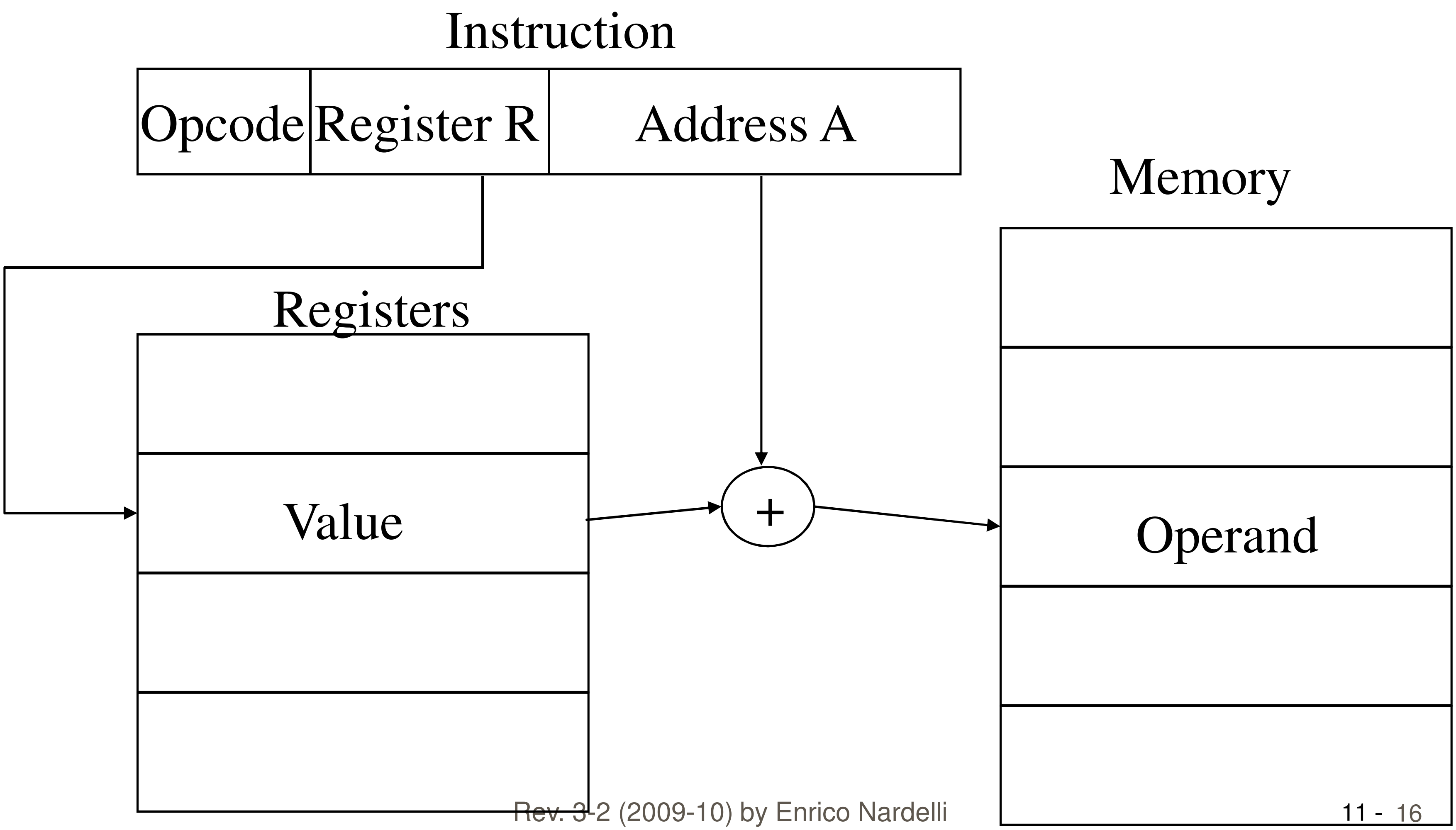
Register Indirect Addressing Diagram



Register Indirect Addressing

- Similar to indirect addressing, but passing through a register
- The register referenced by the address field contains the address of (i.e., the pointer to) the operand
- If R is the register name then EA = (R)
- e.g. ADD (rA)
  - Look into register A, then go to address (A) for operand
  - Add this operand to accumulator and store result in accumulator
  - $Acc+((rA))\rightarrow Acc$
- Large address space (2^n, where *n* is the number of bits in a register), like indirect addressing
- One fewer main memory access than indirect addressing

Displacement Addressing Diagram



Displacement Addressing

- Address field contains two values: one is a register name R and one is a value A
- The effective address is the sum of A and of the content of R
- EA = A + (R)
- It allows to implement three logically different uses
  - *Relative* addressing
  - *Base* addressing
  - *Indexed* addressing
- Slower execution, since additional time is needed for addition

Relative Addressing

- Displacement with respect to the current position in the program
- That is, R = PC, the program counter
- EA = A + (PC)
- Get operand from the cell at the address A cells away from the current location pointed to by PC

Base Addressing

- Register R holds the pointer to a *base* address
- A is the displacement value
- R may be specified explicitly or implicitly
- EA = A + (R)

Indexed Addressing

- R contains the displacement (the *index*)
- A is the base value
- EA = A + (R)
- Good for accessing all array cells in sequence (indexed access to the array)
  - First access address EA = A + (R), then increment the content of R, and repeat

Combination of displacement and indirection

- **Postindex**: first indirection on memory reference and then displacement

  EA = (A) + (R)

- **Preindex**: first displacement and then indirection on the result

  EA = (A+(R))

- Draw the diagrams yourself !

Stack Addressing

- Operand is (implicitly) on top of stack
- e.g.
  - S_ADD Pop top two items from stack and add

Instruction Formats

- Layout of bits in an instruction
- How many bits for the opcode (hence how many different operations)
- How many fields for references to operands (=address fields) and how many bits for each field
  - References may be implicit in opcodes as in the case of stack operations
- Usually the instruction set has more than one instruction format

Instruction Length

- Affected by and affects:
  - Memory size
  - Memory organization
  - Bus structure
  - CPU complexity
  - CPU speed
- Trade off between powerful instruction repertoire (i.e., more bits = more instructions) and saving space

Allocation of Bits

- Affected by and affects
  - Number of instructions
  - Number of addressing modes
  - Number of operands
  - Operands in register versus operands in memory
  - Number of registers and of register sets
  - Address range
  - Address granularity