Instruction Set Architecture

ICT 2203 Computer Architecture

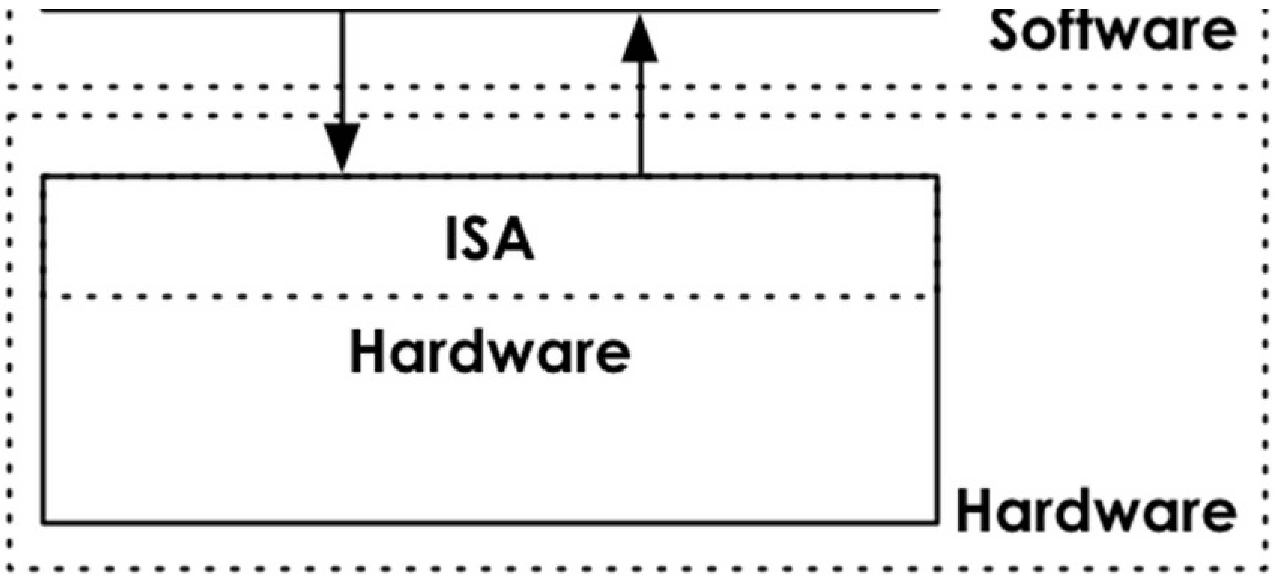
Based on Computer Organization and Architecture, 6th Edition, by William Stallings

What is an Instruction Set?

- The complete collection of instructions that are understood by a CPU
- Machine language:
  - Binary representation of operations and (addresses of) arguments
- Assembly language:
  - Memorable representation for humans

Instruction Set Architecture (ISA)

- Serves as an interface between software and hardware.
- Specific to the hardware of a CPU and its internal structure.
- Provides a mechanism for software to tell hardware what to do.



Elements of an Instruction

- Operation code (opcode)
  - Do this: ADD, SUB, MPY, DIV, LOAD, STOR
- Source operand reference
  - To this: (address of) argument of op, e.g. register, memory location
- Result operand reference
  - Put the result here
- Next instruction reference
  - When you have done that

Design Decisions

- ISA Design
  - How many operations?
  - What can they do?
  - How complex are they?
- Data types
  - length of words
  - integer representation
- Instruction formats
  - Length of op code field
  - Length and number of addresses

Design Decisions

- Registers
  - Number of CPU registers available
  - Which operations can be performed on which registers? General purpose and specific registers
- Addressing modes (see later)
- RISC (Reduced Instruction Set Computer)
- CISC (Complex Instruction Set Computer)

Instruction Types

- Data transfer:
  - registers, main memory, stack or I/O
- Data processing:
  - arithmetic, logical
- Control:
  - systems control, transfer of control

Data Transfer

- Store, load, exchange, move, clear, set, push, pop
- Specifies: source and destination (memory, register, stack), amount of data
- May be different instructions for different (size, location) movements,

Input/Output

- May be specific instructions
  - e.g. INPUT, OUTPUT
- May be done using data movement instructions
  - (memory mapped I/O)
- May be done by a separate controller (DMA)
  - Start I/O, Test I/O

Arithmetic

- Add, Subtract, Multiply, Divide for signed integer (+ floating point and packed decimal)
  - may involve data movement
- May include
  - Absolute (|a|)
  - Increment (a++)
  - Decrement (a--)
  - Negate (-a)

Logical

- Bitwise operations:
  - AND, OR, NOT, XOR, TEST, CMP, SET
- Shifting and rotating functions, e.g.
  - logical right shift for unpacking: send 8-bit character from 16-bit word
  - arithmetic right shift: division and truncation for odd numbers
  - arithmetic left shift: multiplication without overflow

Transfer of Control

- Skip
  - e.g., increment and skip if zero
  - ISZ Reg1, cf. jumping out from loop
- Branch instructions:
  - BRZ X (branch to X if result is zero)
  - BRP X (positive)
  - BRN X (negative)
  - BRE X,R1,R2 (equal)
- Procedure (economy and modularity)
  - call and return

Instruction Formats

- Layout of bits in an instruction
- Includes opcode
- Includes (implicit or explicit) operand(s)
- Usually more than one instruction format in an instruction set

Instruction Length

- Affected by and affects:
  - Memory size
  - Memory organization - addressing
  - Bus structure, e.g., width
  - CPU complexity
  - CPU speed
- Trade off between powerful instruction repertoire and saving space

Types of Operand

- Addresses:
  - immediate, direct, indirect, stack
- Numbers:
  - integer or fixed point (binary, twos complement), floating point (sign, exponent), (packed) decimal
- Characters:
  - ASCII (128 printable and control characters + bit for error detection)
- Logical Data:
  - bits or flags, e.g., Boolean 0 and 1

Allocation of Bits

- Number of addressing modes: implicit or additional bits specifying it
- Number of operands
- Register (faster, limited size and number, 32) versus memory
- Number of register sets, e.g., data and address (shorter addresses)
- Address range
- Address granularity (e.g., by byte)

Number of Addresses

- More addresses
  - More complex (powerful?) instructions
  - More registers - inter-register operations are quicker
  - Less instructions per program
- Fewer addresses
  - Less complex (powerful?) instructions
  - More instructions per program, e.g. data movement
  - Faster fetch/execution of instructions
- Example:
  - Y=(A-B):[(C+(DxE)]

Three addresses

- Operation Result, Operand 1, Operand 2
  - Not common
  - Needs very long words to hold everything

SUB Y,A,B	Y <- A-B
MPY T,D,E	T <- DxE
ADD T,T,C	T <- T+C
DIV Y,Y,T	Y <- Y:T

Two addresses

- One address doubles as operand and result
  - Reduces length of instruction
  - Requires some extra work: temporary storage

MOVE Y,A	Y <- A
SUB Y,B	Y <- Y-B
MOVE T,D	T <- D
MPY T,E	T <- TxE
ADD T,C	T <- T+C
DIV Y,T	Y <- Y:T

One address

- Implicit second address, usually a register (accumulator, AC)

LOAD D	AC <- D
MPY E	AC <- ACxE
ADD C	AC <- AC+C
STOR Y	Y <- AC
LOAD A	AC <- A
SUB B	AC <- AC-B
DIV Y	AC <- AC:Y
STOR Y	Y <- AC

0 (zero) addresses

- All addresses implicit, e.g. ADD
  - Uses a stack, e.g. pop a, pop b, add
  - c = a + b

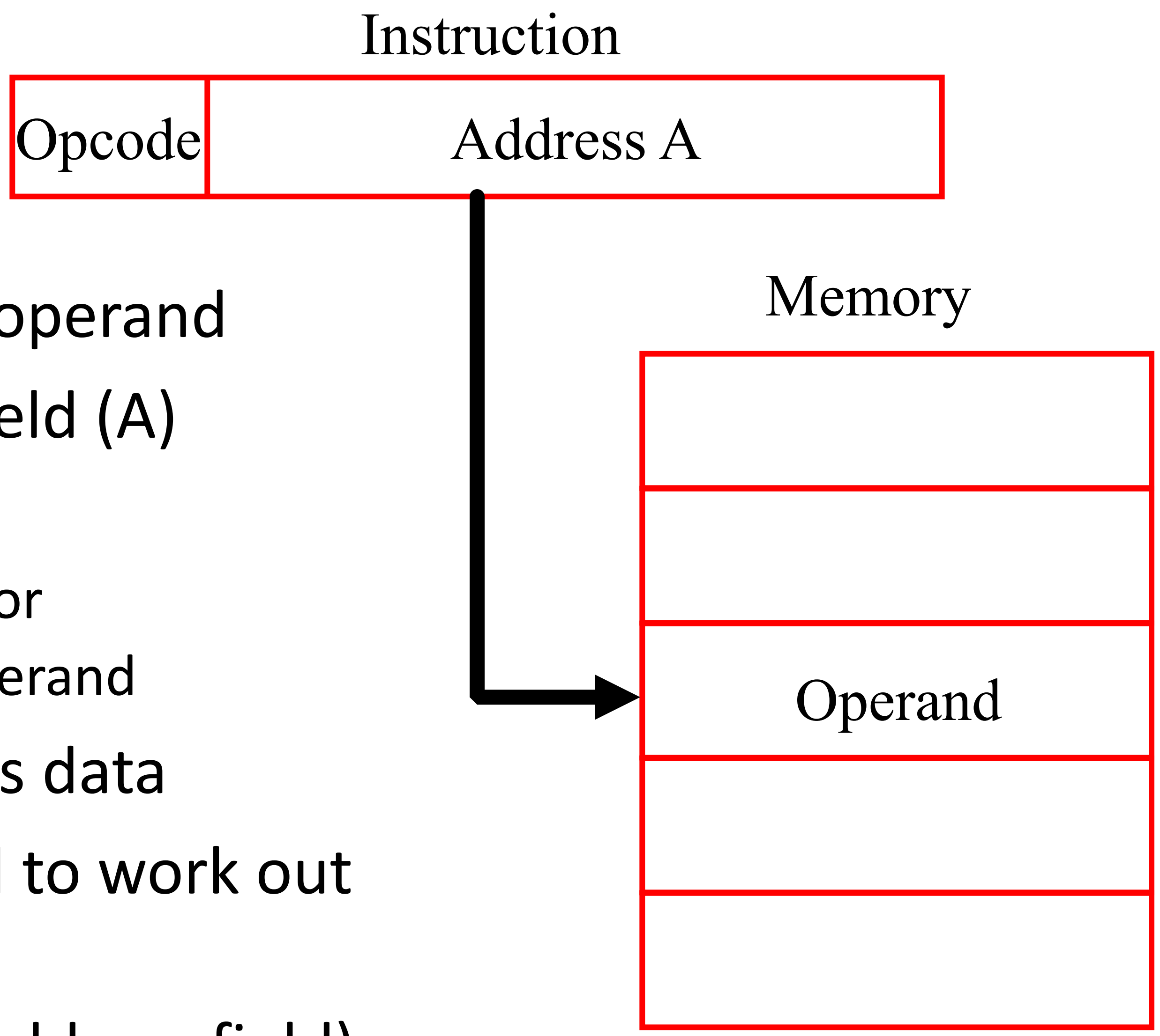
Addressing Modes

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

Immediate Addressing

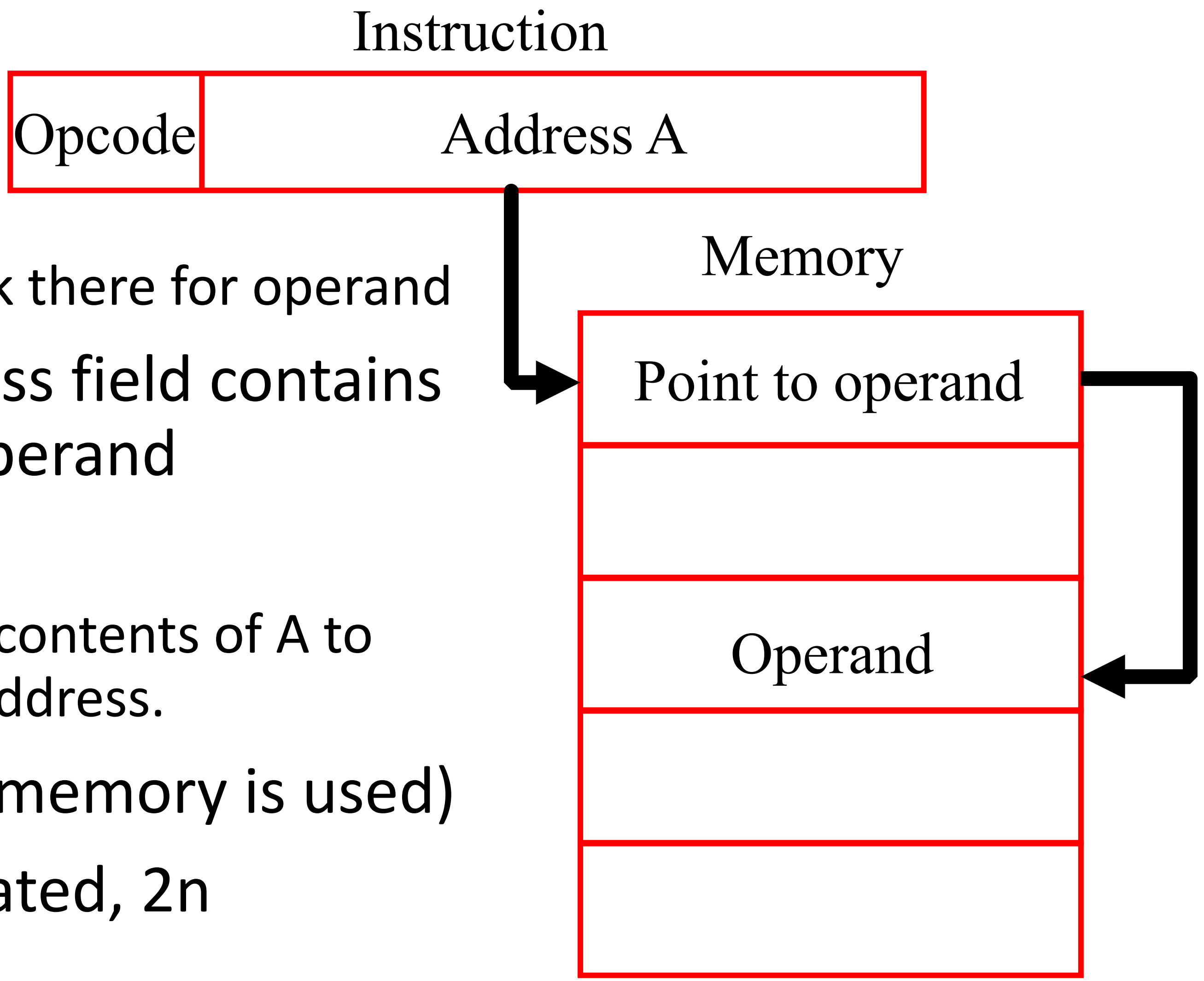
- Operand is part of instruction
- Operand = address field
- e.g., ADD 5 or ADD #5
  - Add 5 to contents of accumulator
  - 5 is operand
- No memory reference to fetch data
- Fast
- Limited range

Direct Addressing



- Address field contains address of operand
- Effective address (EA) = address field (A)
- e.g., ADD A
  - Add contents of cell A to accumulator
  - Look in memory at address A for operand
- Single memory reference to access data
- No additional calculations needed to work out effective address
- Limited address space (length of address field)

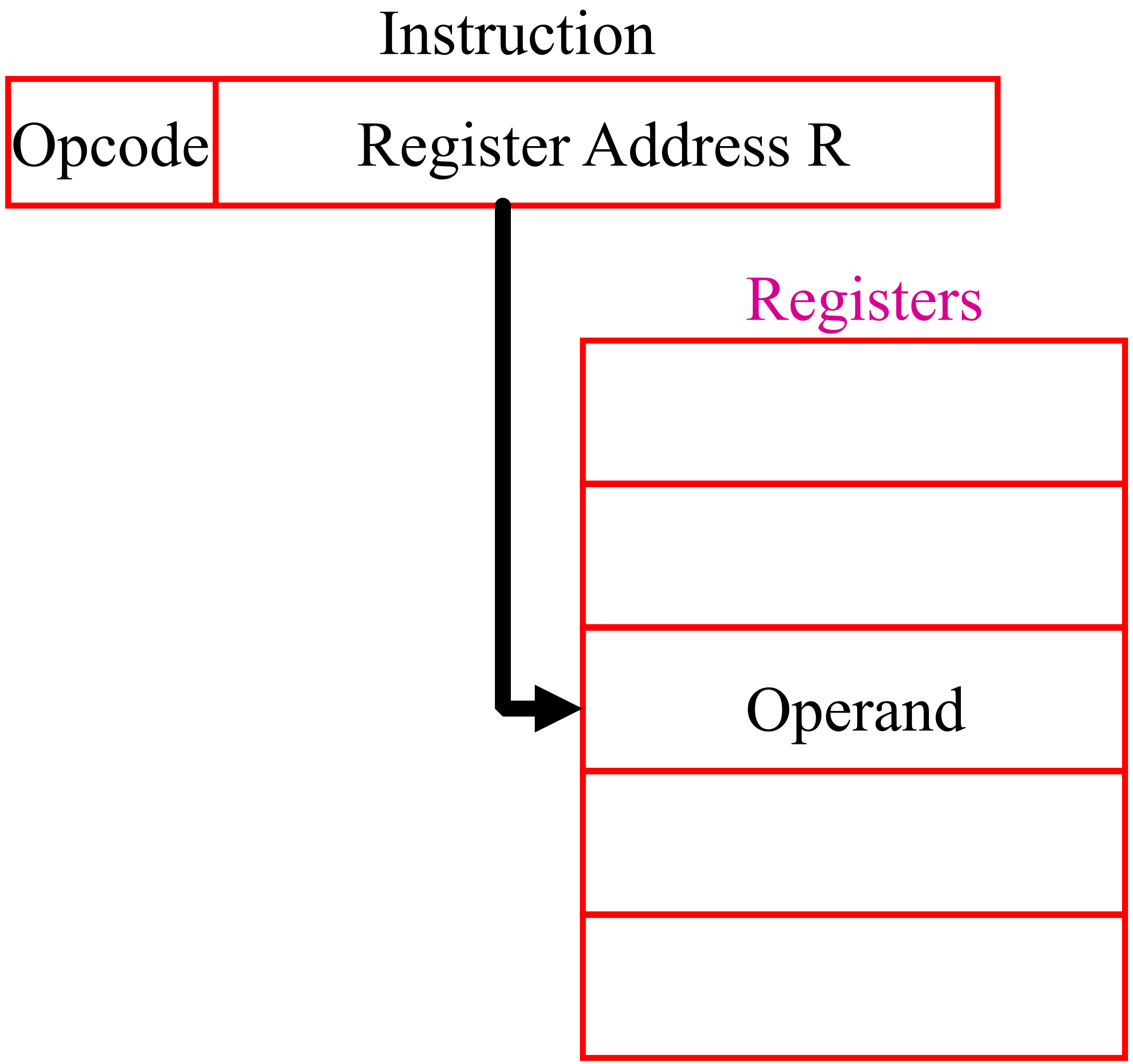
Indirect Addressing



- EA = (A)
  - Look in A, find address (A) and look there for operand
- Memory cell pointed to by address field contains the address of (pointer to) the operand
- E.g. ADD (A)
  - Add contents of cell pointed to by contents of A to accumulator. () denotes memory address.
- Large address space (since main memory is used)
- Number of addresses can be created, 2n
  - where n = word length
- Multiple memory accesses, hence slower

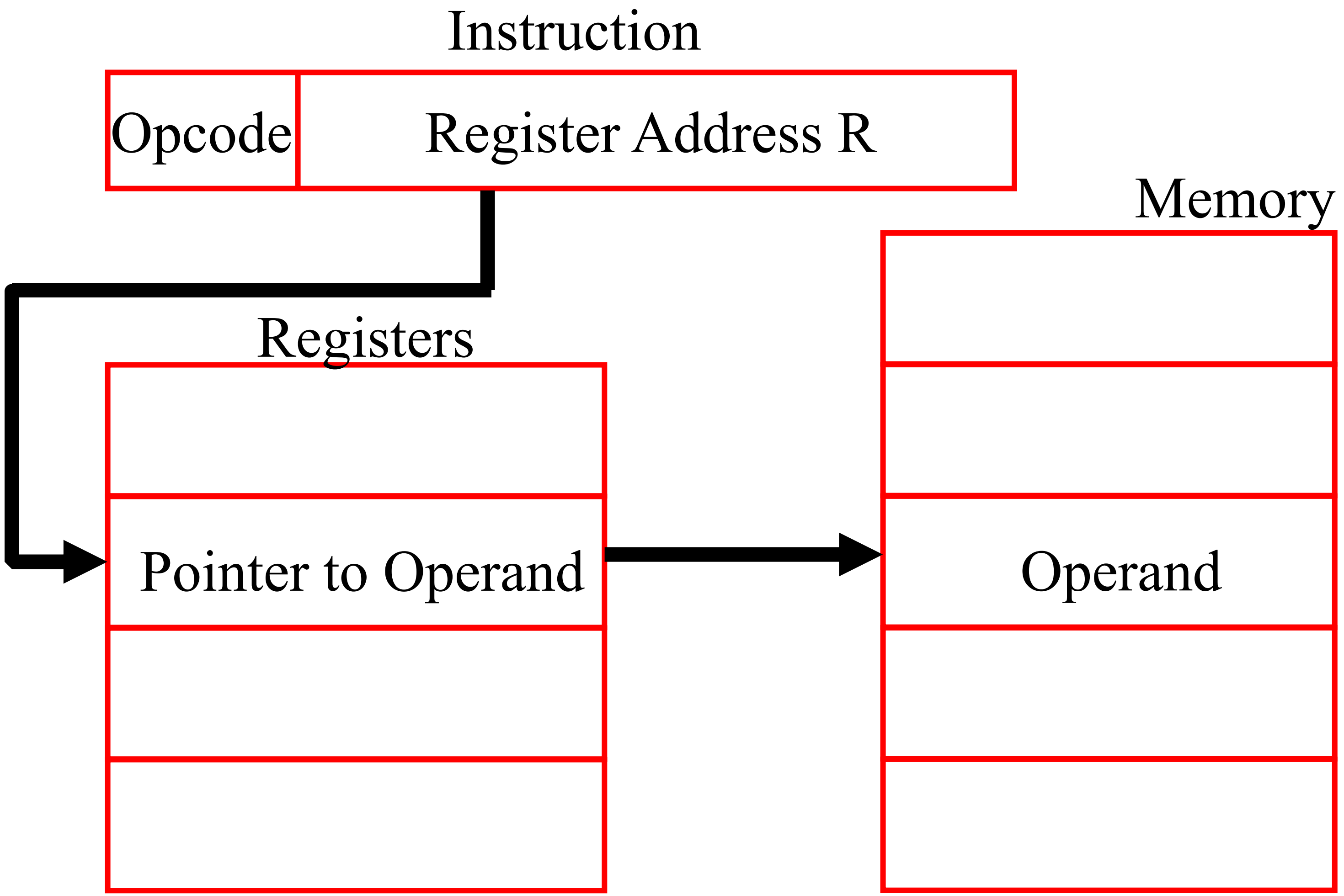
Register Addressing

- EA = R
- Operand is held in register named in address filed
- Limited number of registers
- Very small address field needed
  - Shorter instructions
  - Faster instruction fetch
- No memory access



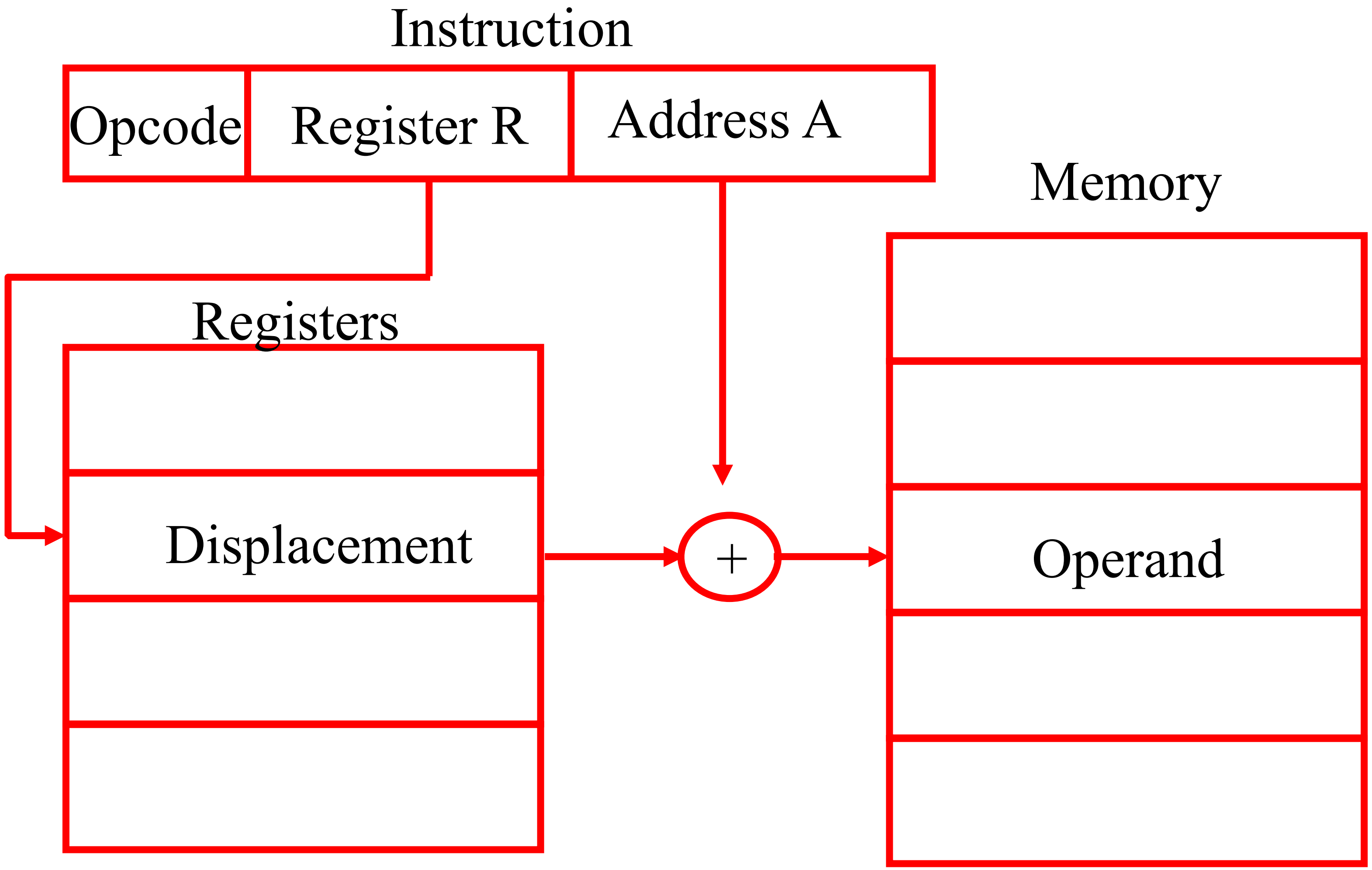
Register Indirect Addressing

- EA = (R)
- Operand is in memory cell pointed to by contents of register R
- Large address space (2n)
- One fewer memory access than indirect addressing



Displacement Addressing

- EA = A + (R)
- Address field hold two values
- A = base value
- R = register that holds displacement



Relative Addressing

- Relative addressing means that the next instruction to be carried out is an offset number of locations away, relative to the address of the current instruction.
- A version of displacement addressing
- R = Program counter, PC
- EA = A + (PC)
- i.e., get operand from A cells away from current location pointed to by PC

Next:

Assembly Language Programming