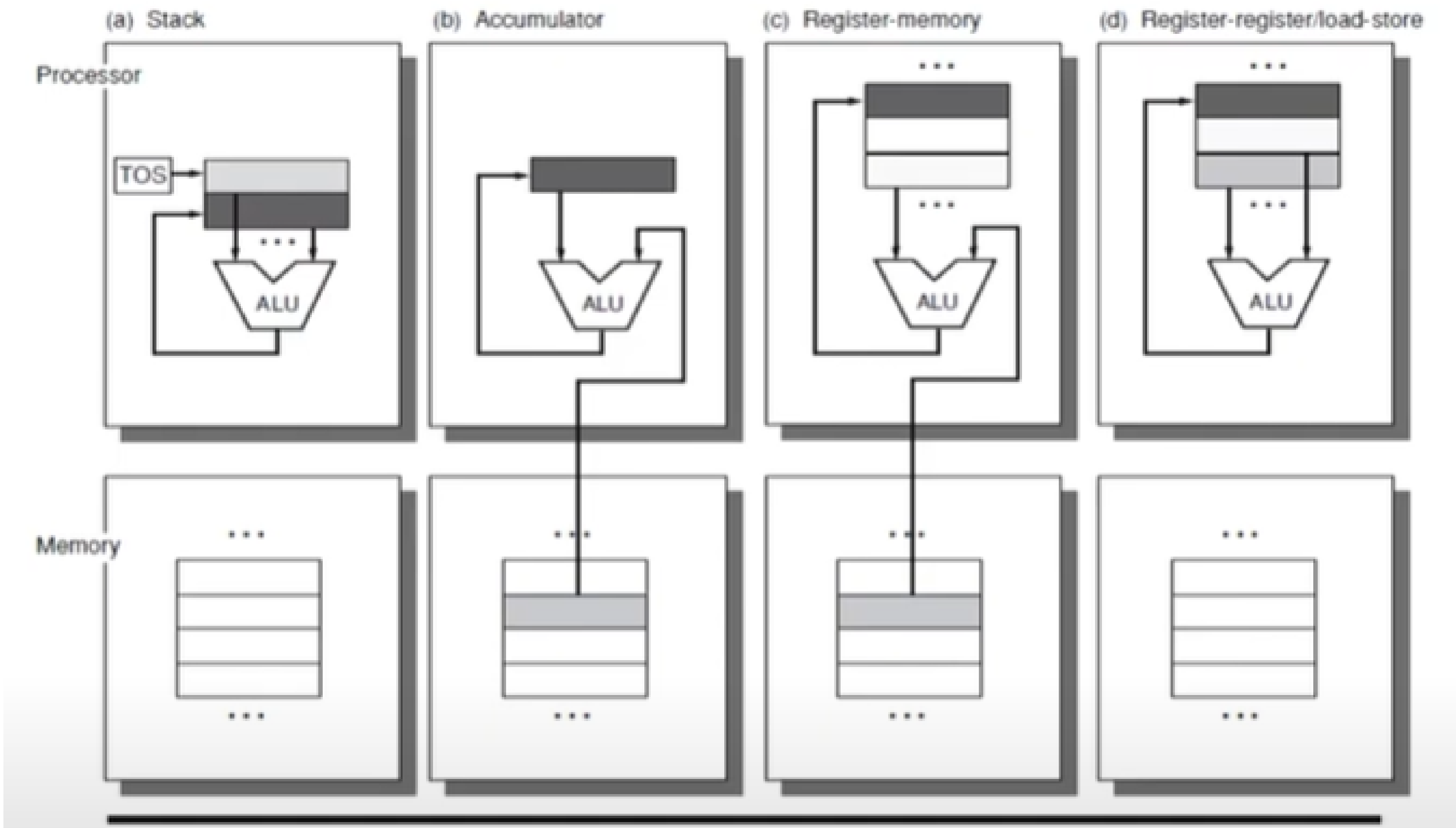
Instruction Sets: Addressing Modes and Formats

Introduction to Instruction Set Design

Introduction to Instruction Set Design

- Instruction set design aims to:
  - Minimize instruction length
  - Maximize flexibility (in CISC design)
  - Accommodate compiler needs
- Addressing modes: Key component in instruction set design
- Balance between instruction length, addressing flexibility, and complexity


(a) Stack
(b) Accumulator
(c) Register-memory
(d) Register-register/load-store
Processor
TOS
ALU
ALU
ALU
ALU
Memory

Common Addressing Modes

- Different types of addresses involve tradeoffs between instruction length, addressing flexibility and complexity of address calculation
  - Immediate
  - Direct
  - Indirect
  - Register
  - Register Indirect
  - Displacement (Indexed)
  - Implied (Stack and others)

Immediate Addressing

- Definition: Operand value is part of the instruction
- Example: ADD eax,5 (Add 5 to contents of accumulator)
- Advantages:
  - No memory reference needed to fetch data
  - Fast execution
- Limitation:
  - Potential range restrictions in fixed-length instructions

Immediate Addressing: Small Operands

- Many immediate mode instructions use small operands (8 bits)
- Challenge:
  - Space waste in 32- or 64-bit machines with fixed-length instructions
- Solution:
  - Some instruction formats include a bit for small operands
- ALU function:
  - Zero-extend or sign-extend the operand to register size

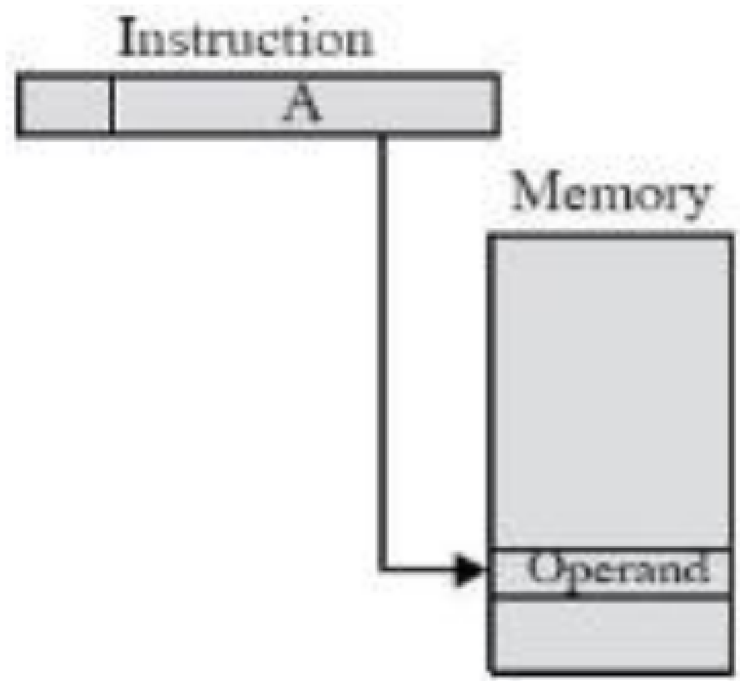
Direct Addressing

- Definition: Address field contains the address of the operand
- Effective Address (EA) = Address field (A)
- Example: add ax, count or add ax,[10FC]
- Characteristics:
  - Single memory reference to access data
  - No additional calculations for effective address

Direct Addressing in x86 Architecture

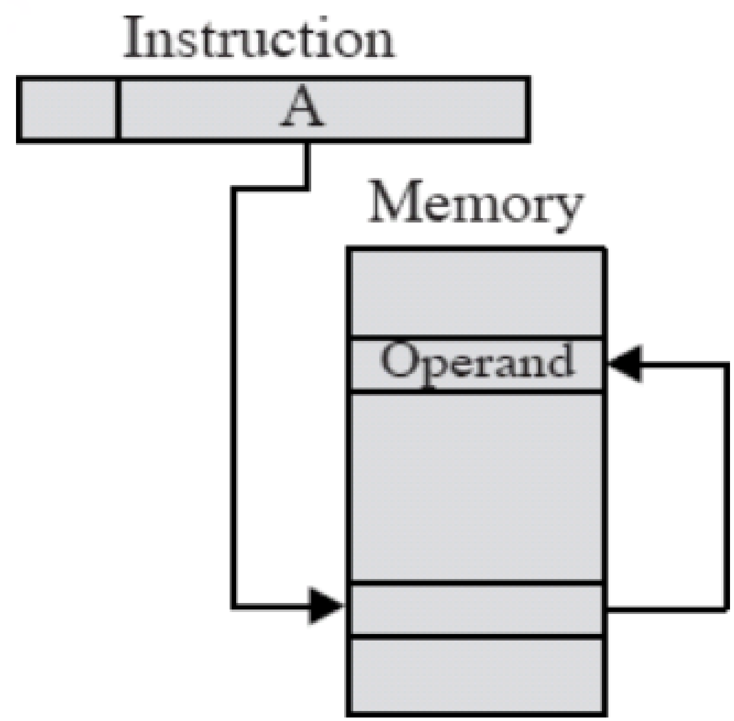
- x86: Segmented architecture
- Segment register involved in EA computation (even in flat memory model)
- Examples:

```
mov [0344], bx

add [00C018A0], edx

pushd [09820014]

inc byte ptr [45AA]

cmp es:[0342], 1
```



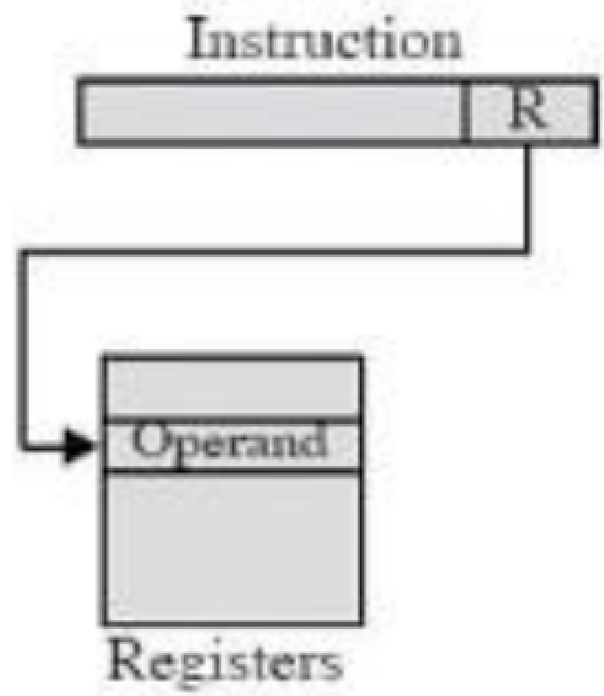
Memory-Indirect Addressing

- Definition: Memory cell pointed to by address field contains the address of the operand
- EA = (A)
- Process: Look in A, find address (A), then look there for operand
- Advantage: Flexible for accessing data through pointers



Register Addressing (Part 1)

- Definition: Operand(s) is/are registers; EA = R
- Characteristics:
  - Register R is EA (not contents of R)
  - Limited number of registers
  - Small address field needed (e.g., x86: 3 bits for 8 registers)
- Advantages:
  - Shorter instructions
  - Faster instruction fetch
  - Eg: X86: 3 bits used to specify one of 8 registers

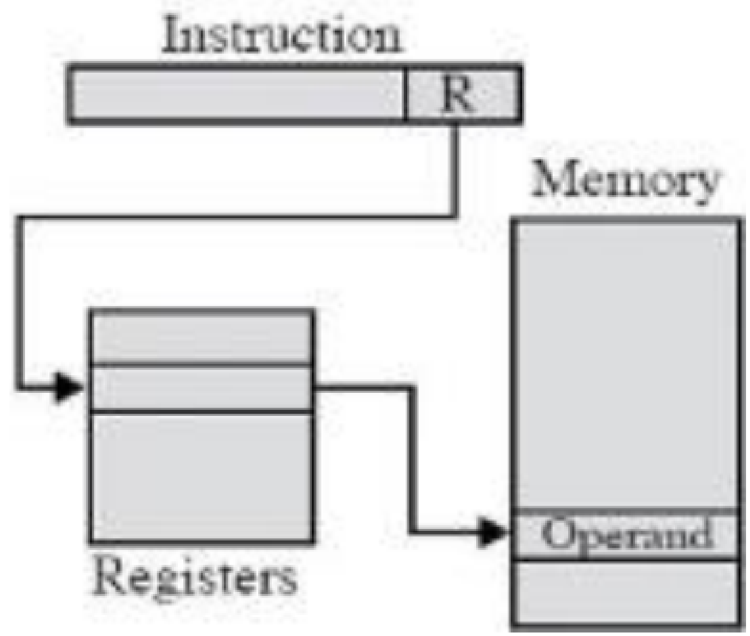


Register Addressing (Part 2)

- Advantages:
  - No memory access needed to fetch EA
  - Very fast execution
- Limitation: Very limited address space
- Multiple registers can help performance
  - Requires good assembly programming or compiler writing
  - Note: in C you can specify register variables register int a;
  - This is only advisory to the compiler
  - No guarantees

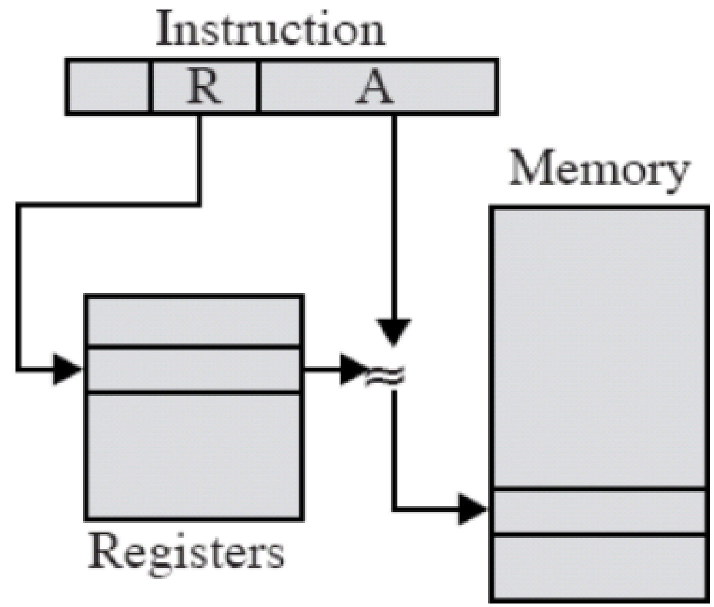
Register Indirect Addressing

- Definition: EA = (R)
- Process: Operand is in memory cell pointed to by contents of register R
- Advantages:
  - Large address space (2^n)
  - One fewer memory access than indirect addressing
- Common use: Accessing data structures through pointers



Displacement Addressing

- Formula: EA = A + (R)
- Combines register indirect addressing with direct addressing
- Address field holds two values:
  - A = base value
  - R = register that holds displacement (or vice versa)
- Types:
  - Relative Addressing
  - Base-register addressing
  - Indexing



Relative Addressing

- Also known as PC-relative addressing
- Formula: EA = A + (PC)
- Characteristics:
  - Address field A treated as 2's complement integer (allows backward references)
  - Fetch operand from PC+A
- Efficiency: Good for locality of reference & cache usage
- Challenge: In large programs, code and data may be widely separated in memory

Base-Register Addressing

- A holds displacement
- R holds pointer to base address
- R may be explicit or implicit
- Example: Segment registers in 80x86 (base registers involved in all EA computations)
- x86 base addressing format examples:
    - `mov eax,[edi + 4 * ecx]`
    - `sub [bx+si-12],2`

Indexed Addressing

- Formula: EA = A + R
    - A = base
    - R = displacement
- Useful for accessing arrays
- Iterative access process:
    - EA = A + R
    - R++
- Some architectures provide auto-increment or auto-decrement:
    - Preindex: EA = A + (R++)
    - Postindex: EA = A + (++R)

Pentium Addressing Modes (Part 1)

- Virtual or effective address: Offset into segment
  - Linear address = Starting address + offset
  - Goes through page translation if paging enabled
- 12 available addressing modes:
  - Immediate
  - Register operand
  - Displacement
  - Base
  - Base with displacement
  - Scaled index with displacement
  - Base with index and displacement
  - Base scaled index with displacement
  - Relative

Activity: Addressing Mode Identification

- Instructions: Identify the addressing mode used in each of the following x86 assembly instructions:
    1. mov eax, 5
    2. add [0x1000], ebx
    3. push [esi]
    4. sub eax, [ebx + 8]
    5. inc dword ptr [eax + 4*ecx + 100]

Example: Array Access Using Indexed Addressing

Consider an array of integers starting at memory address 1000:

```
mov esi, 1000 ; Base address

mov ecx, 0 ; Index

mov eax, [esi + 4*ecx] ; Load first element

inc ecx

mov ebx, [esi + 4*ecx] ; Load second element
```

This example demonstrates how indexed addressing simplifies array access.

Summary: Addressing Modes and Instruction Set Design

- Addressing modes are crucial for instruction set efficiency
- Each mode offers trade-offs between:
  - Instruction length
  - Addressing flexibility
  - Complexity of address calculation
- Choice of addressing modes impacts:
  - Program size
  - Execution speed
  - Memory access patterns
- Modern architectures often support multiple addressing modes for versatility