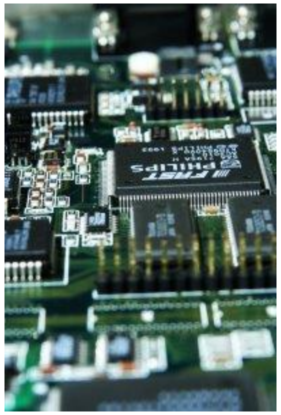
CSE 321a

Computer Organization (1)

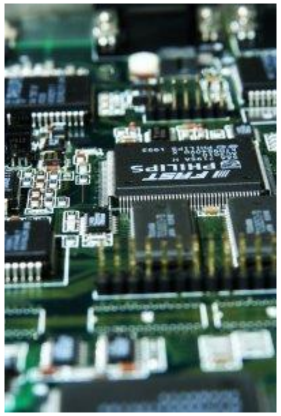
تنظيم الحاسبات (1)



3rd year, Computer Engineering

Fall 2017

Lecture #9



Dr. Hazem Ibrahim Shehata

Dept. of Computer & Systems Engineering

Credits to Dr. Ahmed Abdul-Monem Ahmed for the slides

Administrivia

- Midterm:
  - Solution to posted today.
  - Marks to be announced later on this week.

Website: http://hshehata.github.io/courses/zu/cse321a/
Office hours: Sunday 1:00pm – 2:00pm

Chapter 13. Instruction Sets: Addressing Modes and Formats

How the address of an operand is specified

How the bits of an instruction are organized

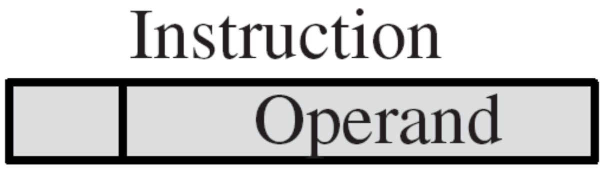
Addressing Modes

- What: how address of operand is specified.
- Why: address field in an instruction is small!
- Tradeoff: address range/addressing flexibility **vs.** number of memory references/complexity of address calculation.

1. Immediate
2. Direct
3. Indirect
4. Register
5. Register Indirect
6. Displacement
7. Stack

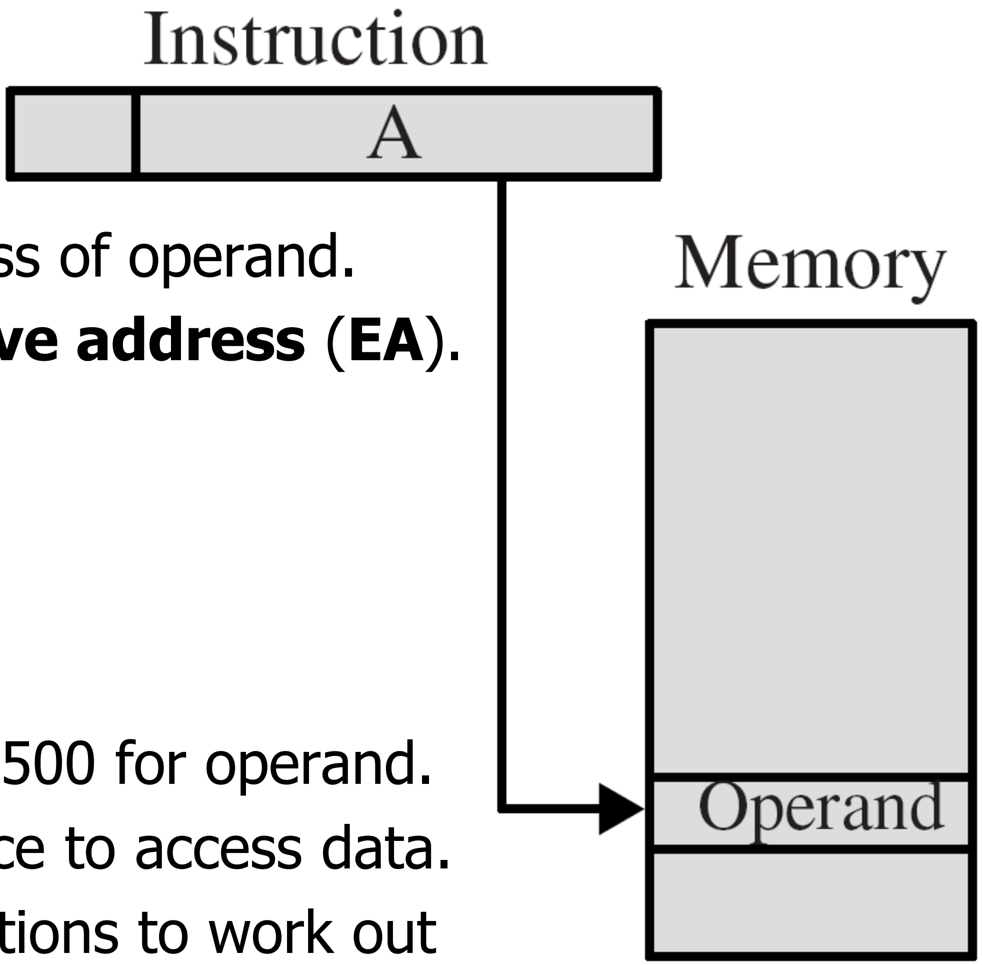
Instruction
A

1. Immediate Addressing



- Address field **A** contains operand.
- Syntax: #A
- Ex.: ADD #5
  - AC 🡐 [AC] + **5**.
  - 5 is the operand.
- 👍 Fast: no memory reference to fetch data.
- 👎 Limited operand magnitude.
- Requires sign extension if loaded to larger register.

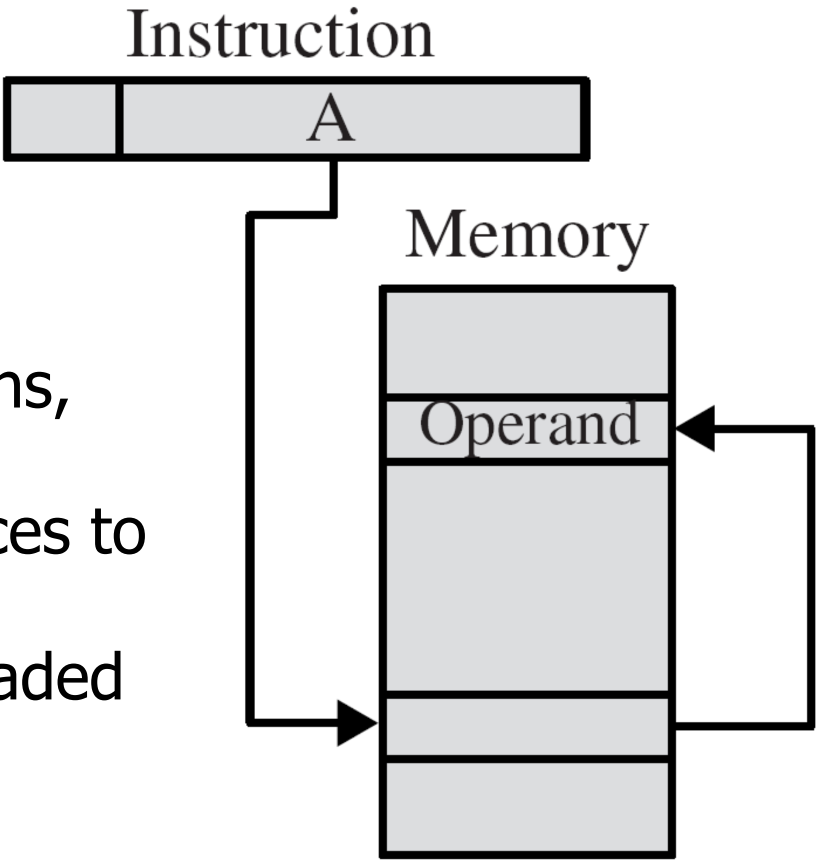
2. Direct Addressing



- Address field contains address of operand.
- Operand Address ➔ **effective address** (**EA**).
- EA = A
- Syntax: A
- Ex.: ADD 500
  - AC 🡐 [AC] + **[500]**.
  - Look in mem. at address 500 for operand.
- 👍 Fast: single memory reference to access data.
- 👍 Simple: no additional calculations to work out effective address.
- 👎 Limited address space.

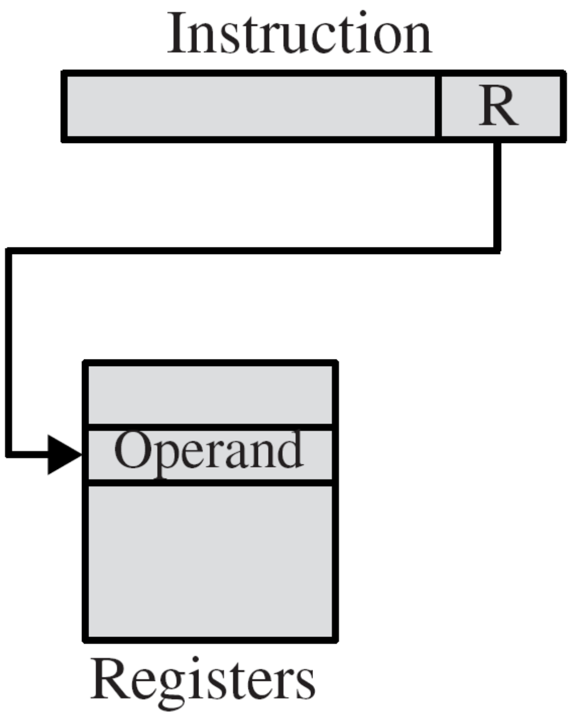
3. Indirect Addressing

- Address field contains address of memory location that contains address of operand.
- EA = [A]
- Syntax: (A)
- Ex.: ADD (1000)
  - AC ← [AC] + **[[1000]]**.
- 👍 Large address range: 2^n locations, where n = word length.
- 👎 Slow: multiple memory references to find operand.
- May be nested, multilevel, cascaded
  - Ex.: SUB (((100)))
    - EA = [[[100]]]
    - Operand = ?
    - Draw the diagram yourself!



4. Register Addressing

- Address field identifies a register that holds the operand.
- c.f. Direct addressing
- EA = R
- Notation: R
- Ex.: ADD R1
  - AC ← [AC] + **[R1]**.
- Number of registers is relatively small.
- Small address field (R: 3-5 bits).
- 👍 Fast fetch: short instruction.
- 👍 Fast execution: no memory reference.
- 👎 Very limited address space.

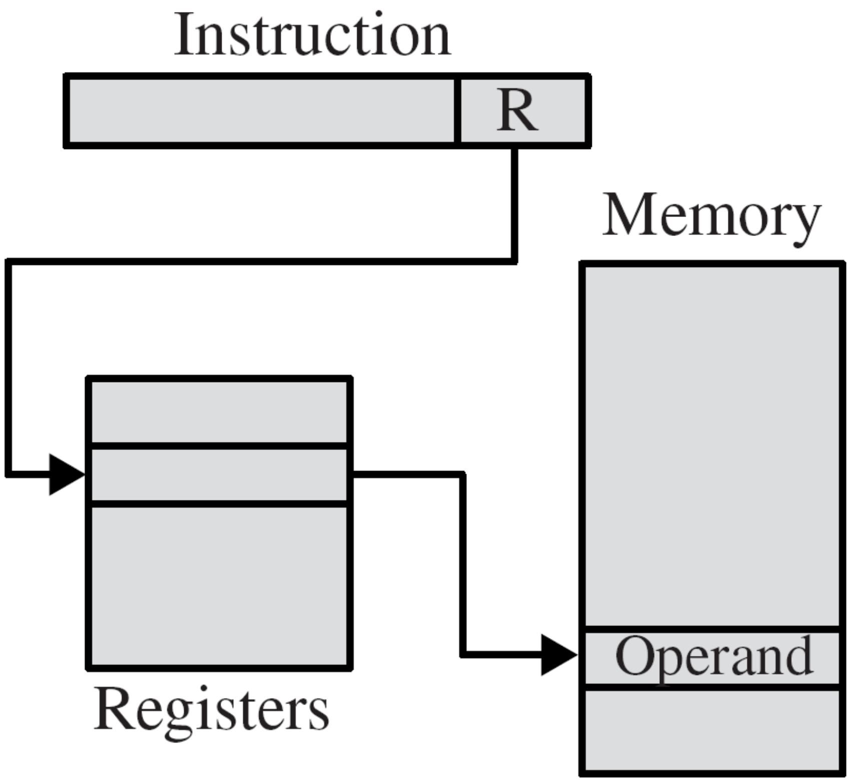


5. Register Indirect Addressing

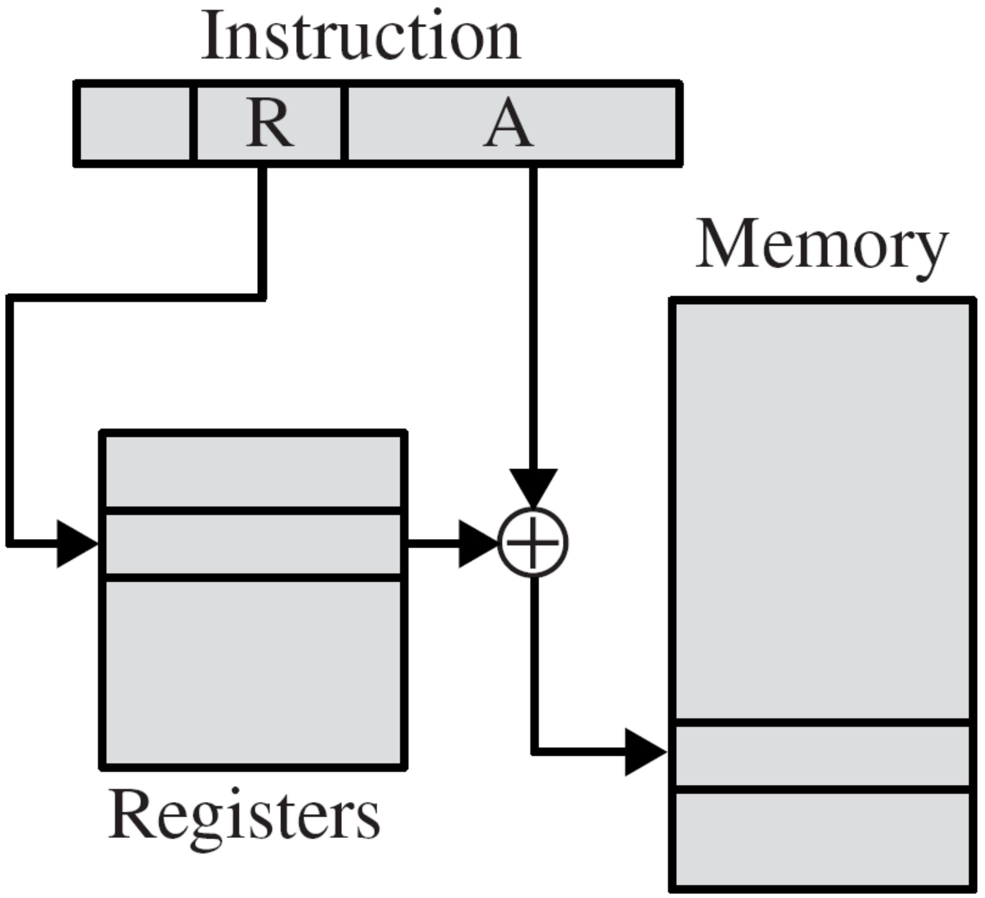
- Address field identifies register that contains the address of the operand.
- c.f. Indirect addressing.
- EA = [R]
- Notation: (R)
- Ex.: ADD (R1)
  - AC ← [AC] + **[[R1]]**.

👍 Large address space (2^n).

👍 One fewer memory access than indirect addressing.



6. Displacement Addressing



- Operand described using 2 address fields: A and R.
- One holds base value and the other holds displacement.
- EA = A + [R]

6. Displacement Addressing – Relative

- **Relative Addressing (PC-relative)**
  - R is implicitly defined as the PC ➔ R=PC
  - A is the displacement; a signed number (in 2's complement representation by default).
  - Meaning: operand found A locations far from following instruction.
  - EA = A + [PC]
  - Notation: @(A)
  - Ex.: ADD @(-100)
    - AC 🡐 [AC] + **[– 100 + [PC]]**.
  - Most common usage: defining target address in (conditional) branch instructions.
    - Ex.: "BRZ @(+30)" means "PC 🡐 30 + [PC]" if zero flag is true.

6. Displacement Addressing – Base Register

- **Base-Register Addressing**
  - Register R holds a base address.
  - A is a number representing a displacement from the base address.
  - EA = A + [R]
  - Notation: R(A)
  - Ex.: ADD R2(50)
    - AC 🡐 [AC] + **[50 + [R2]]**
  - R may be explicit or implicit (e.g., segment registers in x86).
  - With N registers and K-bit displacement A, an instruction can reference one of N areas of 2^K words.
  - Most common usage: implementing segmentation.

6. Displacement Addressing – Indexed

- **Indexed Addressing**
  - A is a memory address.
  - Register R holds displacement (index register).
  - EA = A + [R]
  - Notation: A(R)
  - Ex.: ADD R5, 40(R3); DEC R3;
    - R5 ← [R5] + **[40 + [R3]]**; **R3 ← [R3] – 1**
  - Typically, A would have more bits than it does in base-register addressing mode.
  - Most common usage: referencing arrays.

6. Displacement Addressing – Autoindexing

- **Autoindexing Addressing**
  - Typically, there is a need to increment/decrement the index register after each reference to it.
  - EA = A + [R]; R ← [R] ± 1
  - Notation: A(R)±
  - Ex.: ADD R5, 40(R3) –
    - R5 ← [R5] + **[40 + [R3]]**; **R3 ← [R3] – 1**
  - Most common usage: referencing arrays.
  - If certain registers are devoted exclusively to indexing → autoindexing can be done automatically.

6. Displacement Addressing – Pre/Post-indexing

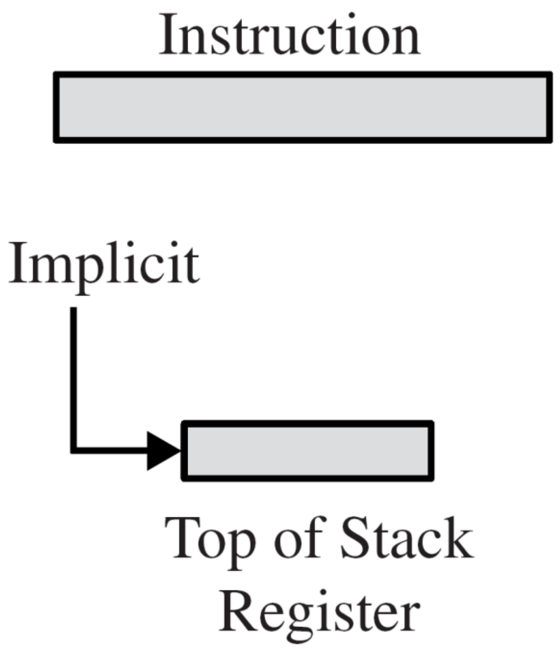
- **Preindexing / Postindexing Addressing:**
  - Sometimes both indexing and indirection are provided.
  - **Preindexing: indexing is done before indirection.**
    - EA = [A + [R]]
    - Notation: (A(R))
    - Ex.: ADD R5, (200(R4))
      - R5 ← [R5] + **[[200 + [R4]]]**
    - Usage: implement multiway branch tables. Table starts at location A. Table is indexed by [R] to select br. target address.
  - **Postindexing: indexing is done after indirection.**
    - EA = [A] + [R]
    - Notation: (A)(R)
    - Ex.: ADD R5, (200)(R4)
      - R5 ← [R5] + **[[200] + [R4]]**
    - Usage: accessing fixed-format data blocks. A block is identified by [A]. Target element in block is identified by [R].

7. Stack Addressing

- Operand is (implicitly) on top of stack.
- EA = [SP]
- Notation: N/A
- Ex.: ADD
  - —Pop top two items from stack, add them, push result back to stack

  [SP] + 1 ← [[SP]] + [[SP] + 1];

  SP ← [SP] + 1



Addressing Modes – Summary

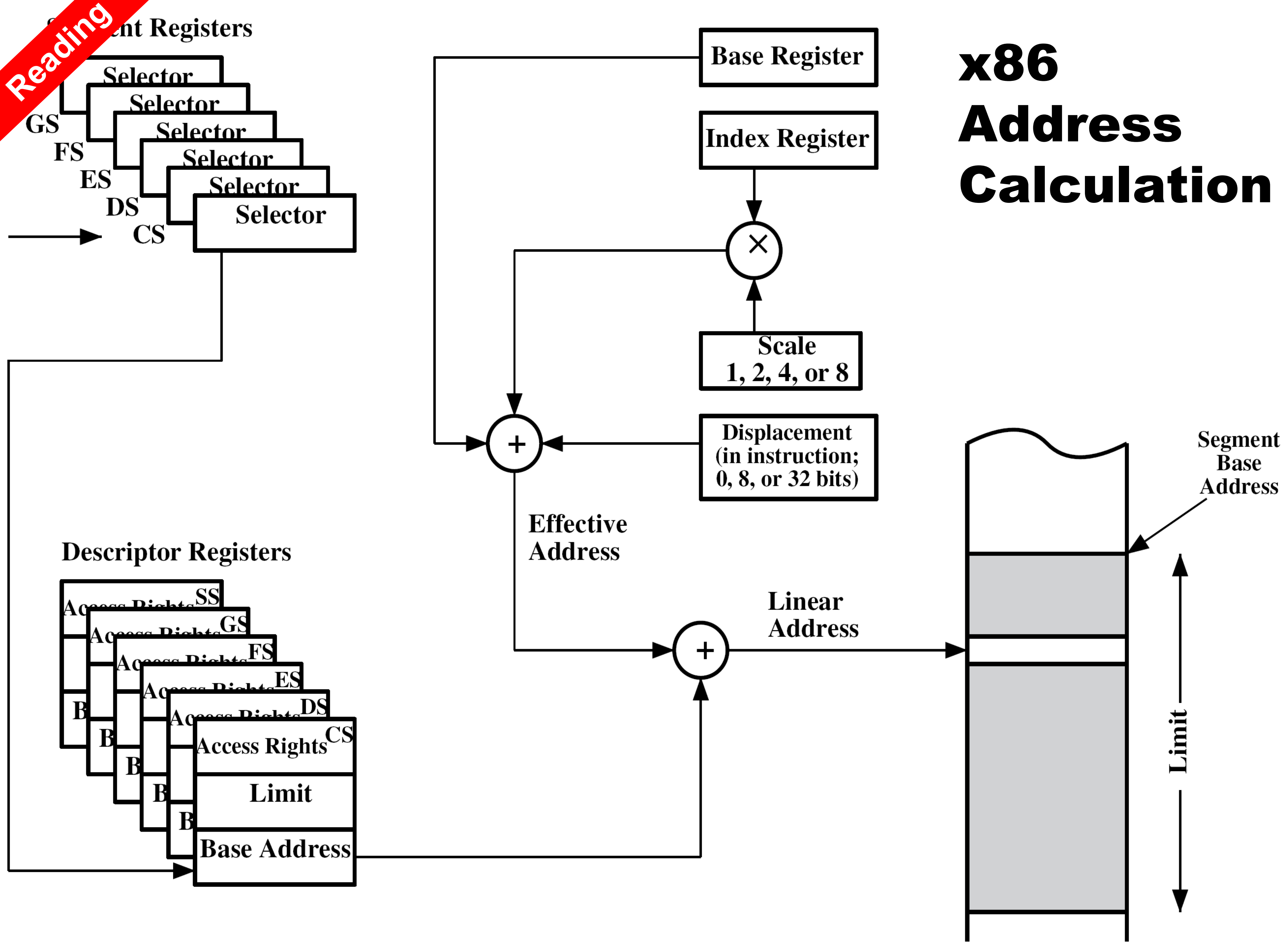
Mode	Algorithm	Principal Advantage	Principal Disadvantage
Immediate	Operand = A	No memory reference	Limited operand magnitude
Direct	EA = A	Simple	Limited address space
Indirect	EA = [A]	Large address space	Multiple memory references
Register	EA = R	No memory reference	Limited address space
Register indirect	EA = [R]	Large address space	Extra memory reference
Displacement	EA = A + [R]	Flexibility	Complexity
Stack	EA = top of stack	No memory reference	Limited applicability

Reading

X86 Addressing Modes

- Virtual or effective address is offset into segment.
  - —Starting address plus offset gives linear address.
  - —This goes through page translation if paging is enabled.
- Addressing modes available
  - —Immediate
  - —Register operand: 8-bit, 16-bit, 32-bit, 64-bit GPR’s.
  - —Displacement: equivalent to indirect.
  - —Base: equivalent to register indirect.
  - —Base with displacement: equivalent to base-register disp.
  - —Scaled index with displacement: Indexed disp. + scaling (to handle word sizes greater than 1 byte).
  - —Base with index and displacement: 2D arrays, array in SF
  - —Base with scaled index & displacement
  - —Relative

x86 Address Calculation



Reading

x86 Addressing Modes – Summary

Mode	Algorithm
Immediate	Operand = A
Register Operand	LA = R
Displacement	LA = [SR] + A
Base	LA = [SR] + [B]
Base with Displacement	LA = [SR] + [B] + A
Scaled Index with Displacement	LA = [SR] + [I] * S + A
Base with Index and Displacement	LA = [SR] + [B] + [I] + A
Base with Scaled Index and Displacement	LA = [SR] + [I] * S + [B] + A
Relative	LA = [PC] + A

LA	=	linear address
[X]	=	contents of X
SR	=	segment register
PC	=	program counter
A	=	contents of an address field in the instruction
R	=	register
B	=	base register
I	=	index register
S	=	scaling factor

General-purpose register

Instruction Formats

- Defines layout of bits of instruction, in terms of its constituent fields.
- Includes opcode and (implicit or explicit) zero or more operands.
- Each explicit operand is referenced using one of the addressing modes.
- Most instruction sets use more than one instruction format.
- Key design issues:
  - —Instruction length
  - —Allocation of bits
  - —Variable-length instructions

Instruction Length

- Instruction format length affects/affected by: memory size/organization, bus structure, CPU complexity/speed.
- Tradeoff: Instruction capabilities **vs.** storage requirements.
  - Programmers need **powerful instructions** ➔ more opcodes, operands, addr. modes ➔ longer instructions ➔ **bigger storage**!!
- Also, instruction length should:
  - Suit **memory-transfer rate**.
    - Long instruction & small memory-transfer rate ➔ CPU fetches less instructions than it executes ➔ memory becomes bottleneck ➔ CPU becomes less busy ➔ performance is hurt!!
  - Equal length of **memory-transfer unit** (or multiple)
  - Equal multiple of **character** length.
  - Equal **word** length.
  - Equal length of **fixed-point numbers**.

Allocation of Bits (1)

- Tradeoff: number of opcodes **vs.** power of addressing.
  - —Work around: **variable-length opcodes** ➔ more opcodes for operations that require less operands and/or addressing.
- Factors go into determining use of addressing bits:

1. Number of addressing modes
   - —Implicit: opcode ➔ particular addressing mode.
   - —Explicit: some bits to specify addressing mode.
2. Number of operands
   - —Fewer addresses ➔ longer programs.
   - —Each operand could need its mode indicator, or just one.
3. Register versus memory
   - —Accumulator ➔ no bits, but longer program.
   - —More registers ➔ used instead of memory ➔ less bits.

Allocation of Bits (2)

- … (Cont.)

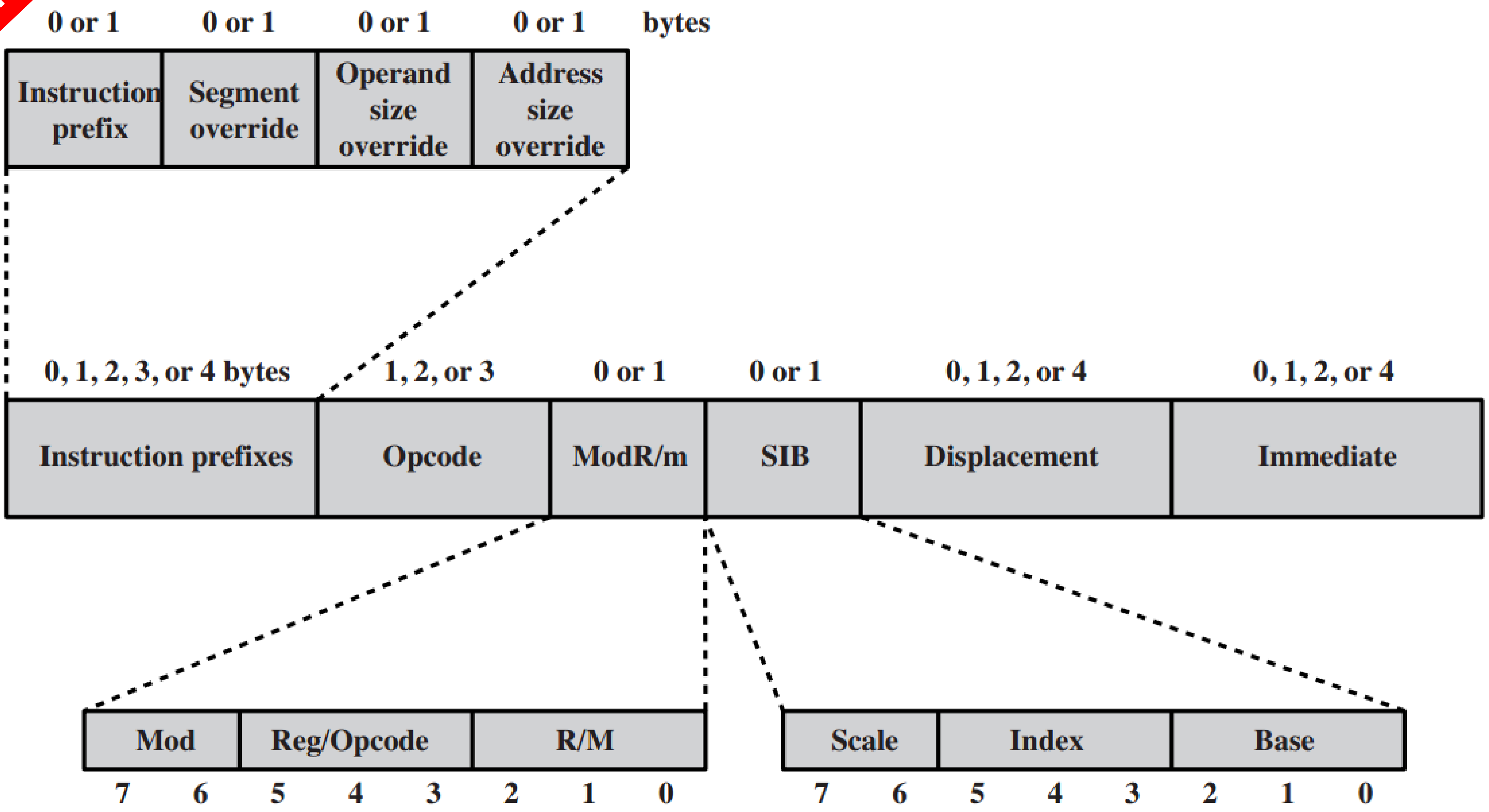
4. Number of register sets
   - A set of general-purpose vs. 2^+ specialized sets
   - e.g., a set for data and another set for displacement.
   - 2 sets of 8 registers ➔ 3 bits are needed, opcode determines which set.
5. Address range (for addresses that reference memory)
   - Direct addressing is rarely used.
   - Displacement addressing: large disp. ➔ more bits.
6. Address granularity (for addresses that reference memory)
   - Byte addressing vs. word addressing.
   - Byte addressing is convenient for characters, but needs large number of bits.

Variable-Length Instructions

- A variety of instruction formats of different lengths.
- Pros
  - Larger repertoire of opcodes.
  - More flexible addressing, i.e., various combinations of reg/mem references.
- Cons
  - Increase processor complexity.
- Instruction lengths should be integrally related to word length.
- Typically, processor fetches a number of words equal to longest possible instruction.

Reading

x86 Instruction Format: variable-length (1-15B)



- Inst. prefix: LOCK or repeat prefixes to repeat operations on strings. # in CX.
- Segment override: which segment register to use.
- Operand size override: specifies 16- or 32-bit operands.
- Address size override: specifies 16- or 32-bit addresses ➔ displacement size.

Assembly Language (N=I+J+K)

Address	Contents			
101	0010	0010	0000	0001
102	0001	0010	0000	0010
103	0001	0010	0000	0011
104	0011	0010	0000	0100
201	0000	0000	0000	0010
202	0000	0000	0000	0011
203	0000	0000	0000	0100
204	0000	0000	0000	0000

(a) Binary program

Address	Instruction	
101	LDA	201
102	ADD	202
103	ADD	203
104	STA	204
201	DAT	2
202	DAT	3
203	DAT	4
204	DAT	0

(b) Symbolic program

Address	Contents
101	2201
102	1202
103	1203
104	3204
201	0002
202	0003
203	0004
204	0000

(c) Hexadecimal program

Label	Operation	Operand
FORMUL	LDA	I
	ADD	J
	ADD	K
	STA	N
I	DATA	2
J	DATA	3
K	DATA	4
N	DATA	0

(d) Assembly program

Reading Material

- Stallings, Chapter 13:
  - Pages 452 – 461
  - Pages 464 – 467
  - Page 469
  - Pages 473 – 475
  - Pages 477 – 479