Akdeniz University
Computer Engineering Department

CSE206 Computer Organization

Week13: Addressing Modes

Assoc.Prof.Dr. Taner Danışman

tdanisman@akdeniz.edu.tr

Addressing Modes

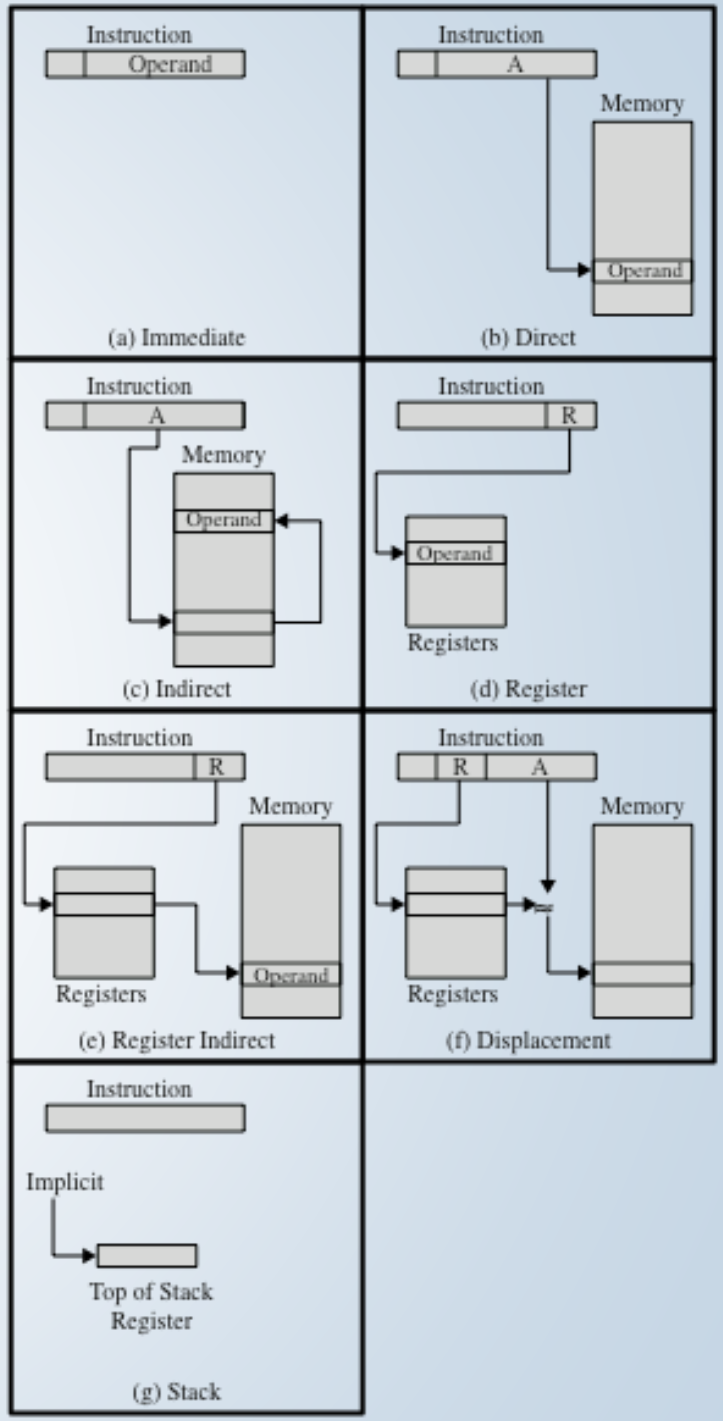
- Immediate
- Direct
- Indirect
- Register
- Register indirect
- Displacement
- Stack

- We would like to be able to reference a large range of locations in main memory or, for some systems, virtual memory.
- To achieve this objective, a variety of addressing techniques has been employed.

Addressing Modes

A = contents of an address field in the instruction
R = contents of an address field in the instruction that refers to a register

- EA = actual (effective) address of the location containing the referenced operand
- (X) = contents of memory location X or register X

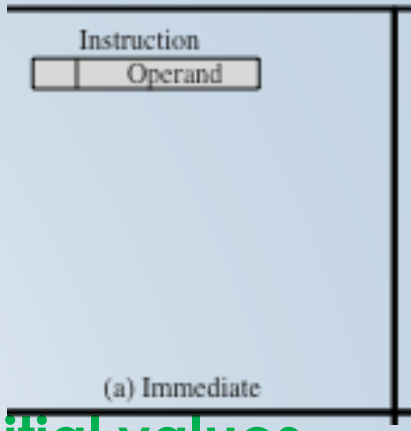


Basic Addressing Modes

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

Table 13.1 Basic Addressing Modes

Immediate Addressing



- Simplest form of addressing
- **Operand = A**
- This mode can be used to define and use constants or set **initial values of variables**
  - Typically the number will be stored in twos complement form
  - The leftmost bit of the operand field is used as a sign bit
- **Advantage:**
  - no memory reference other than the instruction fetch is required to obtain the operand, thus saving one memory or cache cycle in the instruction cycle
- **Disadvantage:**
  - The size of the number is restricted to the size of the **address field,** which, in most instruction sets, is small compared with the word length

Direct Addressing

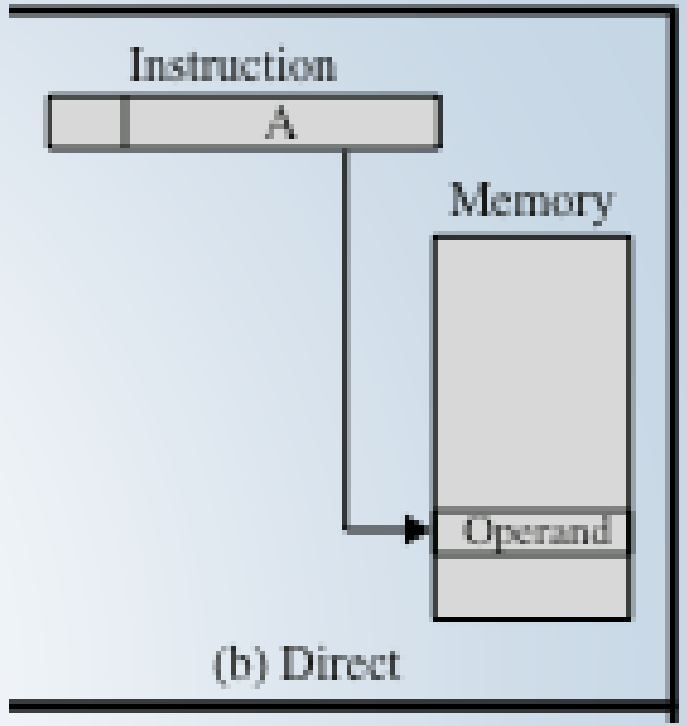
Address field contains the effective address of the operand

Effective address (EA) = address field (A)

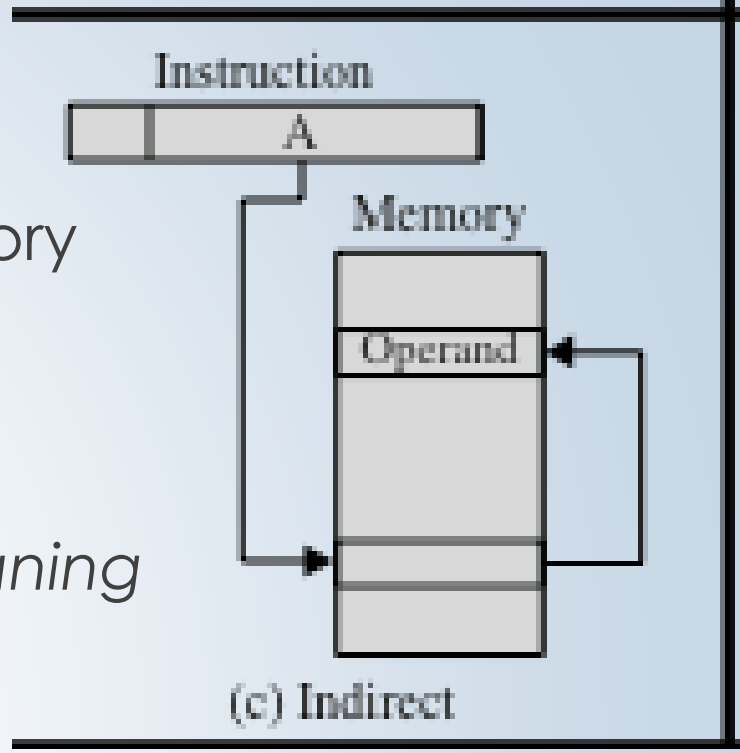
Was common in earlier generations of computers

Requires only one memory reference and no special calculation

Limitation is that it provides only a limited address space



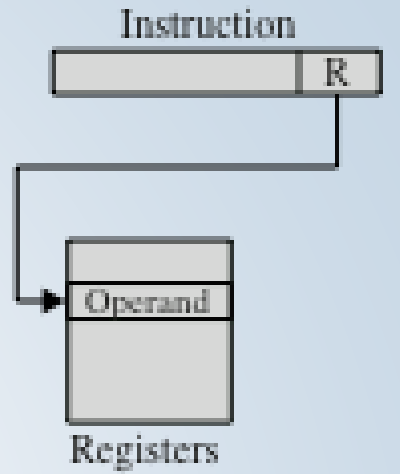
Indirect Addressing



- Reference to the address of a word in memory which contains a full-length address of the operand
- **EA = (A)**
  - Parentheses are to be interpreted as *meaning contents of*
- **Advantage:**
  - For a word length of *N* an address space of 2^N is now available
- **Disadvantage:**
  - Instruction execution requires **two memory references** to fetch the operand
    - ***One to get its address*** and a ***second to get its value***

Register Addressing

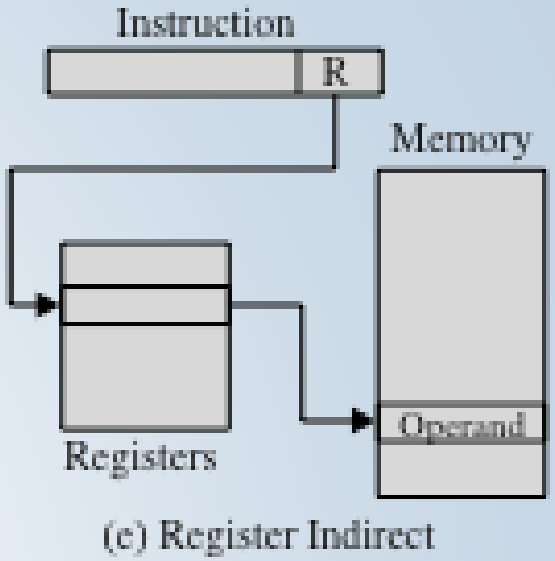
- ***Address field*** refers to a register rather than a main memory address
- **EA = R**
- **Advantages:**
  - Only a small address field is needed in the instruction
  - No time-consuming memory references are required
- **Disadvantage:**
  - The address space is very limited



(d) Register

- If the contents of a register address field in an instruction is 5, then register R5 is the **intended address**, and the operand value is contained in R5
- Typically, an **address field** that references registers will have from 3 to 5 bits, so that a total of from 8 to 32 general-purpose registers can be referenced.

Register Indirect Addressing

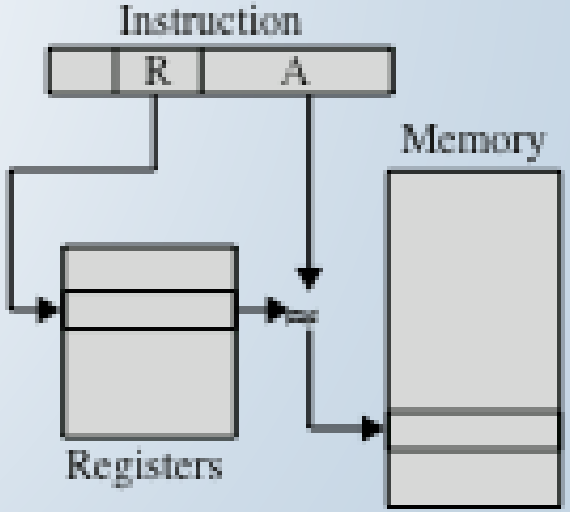


(e) Register Indirect

- Analogous to indirect addressing
  - The only difference is whether the address field refers to a memory location or a register
- **EA = (R)**
- *Address space limitation of the address field is overcome* by having that field refer to a word-length location containing an address
- **Uses one less memory reference than indirect addressing**

Displacement Addressing

- Combines the capabilities of **direct addressing** and **register indirect addressing**
- **EA = A + (R)**
- Requires that the instruction have **two address fields**, at least one of which is explicit
  - The value contained in one address field (value = A) is used directly
  - The other address field refers to a register whose contents are added to A to produce the **effective address**
- **Most common uses:**
  - **Relative addressing**
  - **Base-register addressing**
  - **Indexing**



(f) Displacement

Relative Addressing

- **The implicitly referenced register is the program counter (PC)**
  - The next instruction address is added to the address field to produce the EA
  - Typically the address field is treated as a twos complement number for this operation
  - Thus the effective address is a displacement relative to the address of the instruction
- Exploits the concept of locality
- Saves address bits in the instruction if most memory references are relatively **near to the instruction being executed**

Base-Register Addressing

- The referenced register contains a **main memory address** and the **address field** contains a **displacement from that address**
- The register reference may be explicit or implicit
- Exploits the locality of memory references
- Convenient means of implementing **segmentation**
  - In some implementations **a single segment** base register is employed and is used implicitly
  - In others the programmer **may choose a register** to hold the base address of a segment and the instruction must reference it explicitly

Indexed Addressing

- The address field references **a main memory address** and the referenced register contains **a positive displacement** from that address
- The method of calculating the EA is the ***same as for base-register addressing***
- An important use is to provide an efficient mechanism for performing iterative operations
- Autoindexing
  - Automatically increment or decrement the index register after each reference to it
  - EA = A + (R)
  - (R) ← (R) + 1
- Postindexing
  - Indexing is performed after the indirection
  - EA = (A) + (R)
- Preindexing
  - Indexing is performed before the indirection
  - EA = (A + (R))

Consider, for example, a list of numbers stored starting at location A. Suppose that we would like to add 1 to each element on the list. We need to fetch each value, add 1 to it, and store it back. The sequence of effective addresses that we need is A, A + 1, A + 2,..., up to the last location on the list. With indexing, this is easily done. The value A is stored in the instruction's address field, and the chosen register, called an *index register*, is initialized to 0. After each operation, the index register is incremented by 1.

Stack Addressing

- A stack is a linear array of locations
  - Sometimes referred to as a ***pushdown list*** or ***last-in-first-out queue***
- A stack is a reserved block of locations
  - Items are **appended to the top of the stack** so that the block is partially filled
- Associated with the stack is a **pointer** whose value is the **address of the top of the stack**
  - **The stack pointer is maintained in a register**
  - Thus references to stack locations in memory are in fact **register indirect addresses**
- Is a form of implied addressing
- The machine instructions need **not include a memory reference but implicitly operate on the top of the stack**

x86 Addressing Mode Calculation

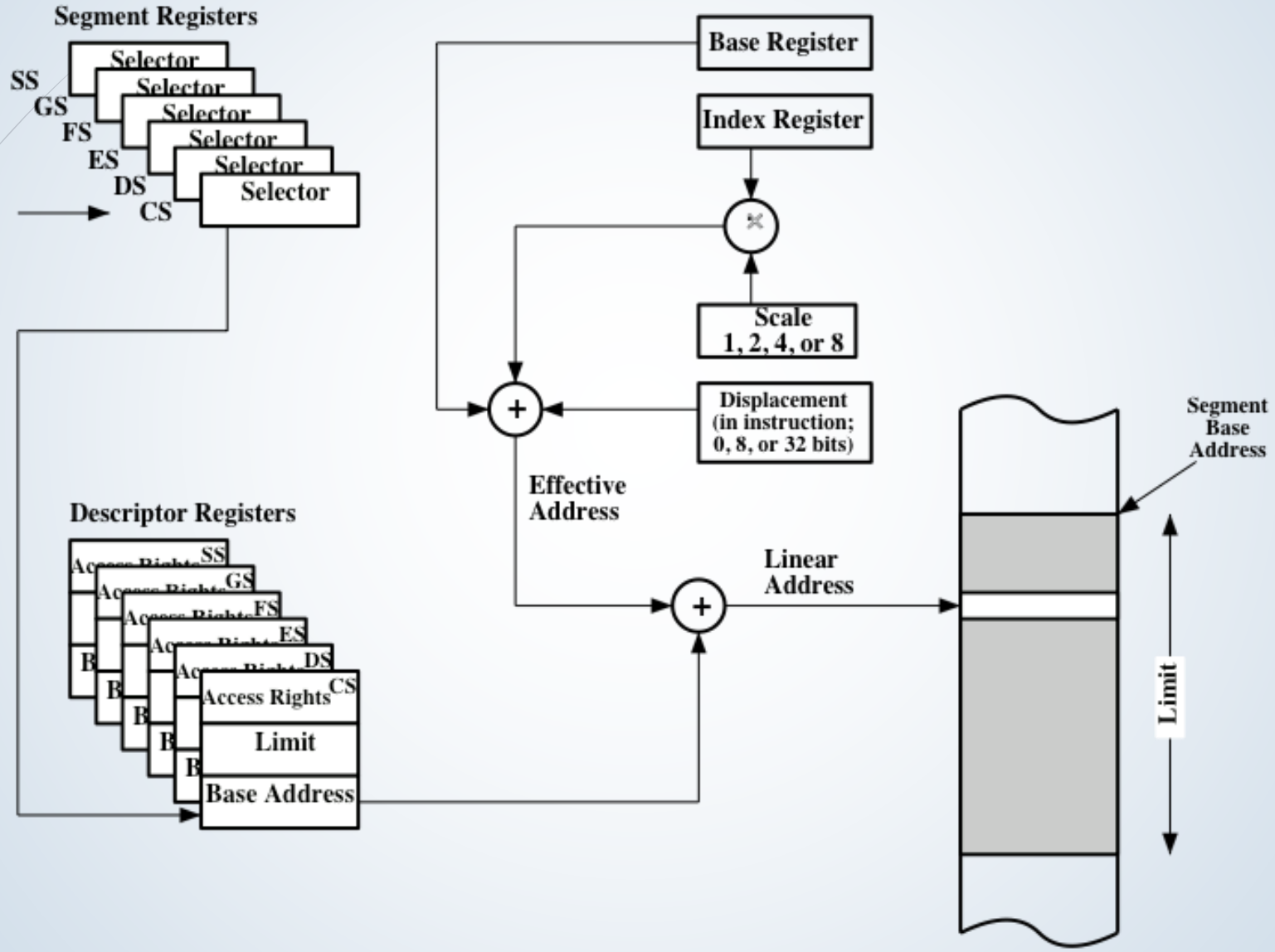


Figure 13.2 x86 Addressing Mode Calculation

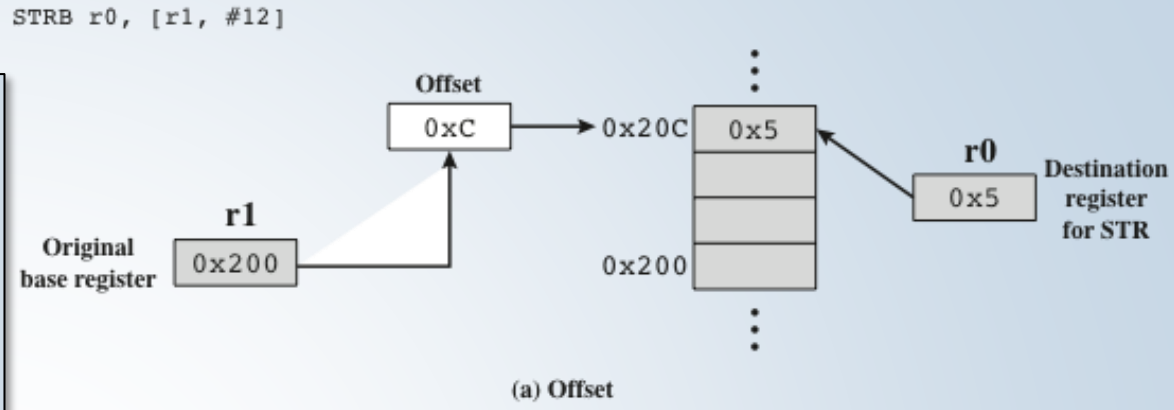
x86 Addressing Modes

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

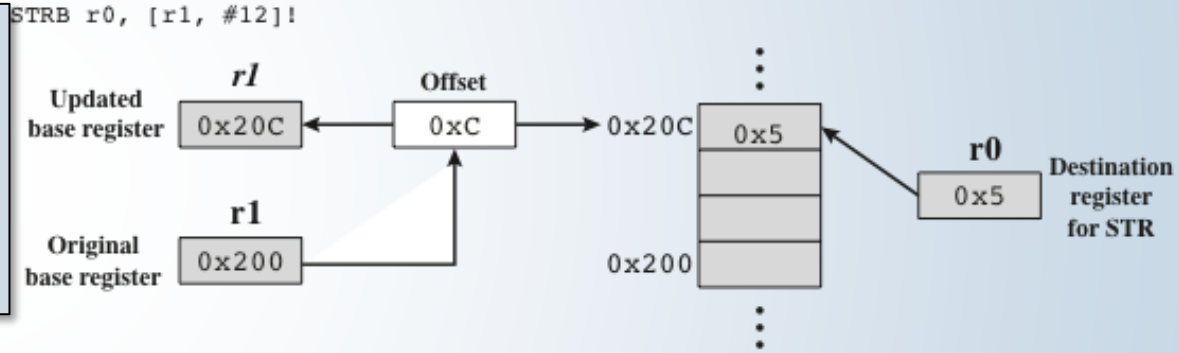
ARM Indexing Methods

An offset value is added to or subtracted from the value in the base register to form the memory address. STRB r0, [r1, #12] is the **store byte instruction**. The **base address** is in register **r1** and the **displacement** is an **immediate value** of decimal 12. The resulting address (**base plus offset**) is the location where the least significant byte from r0 is to be stored.



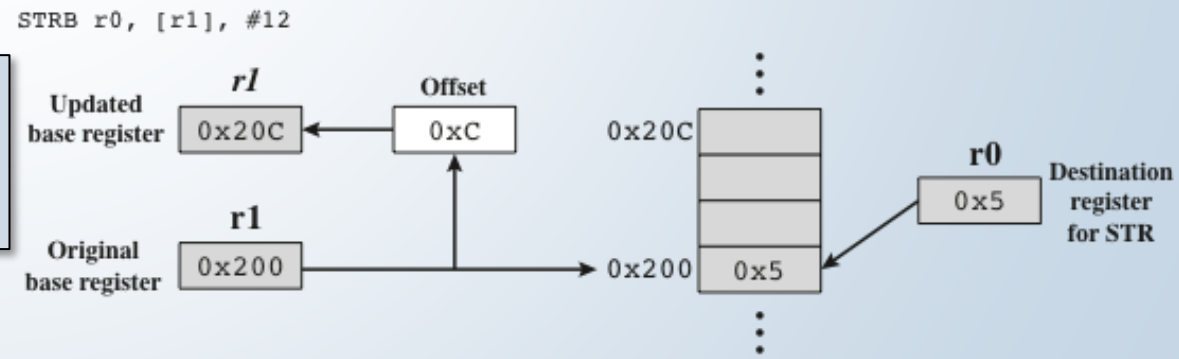
(a) Offset

The memory address is formed in the same way as for off set addressing. The memory address is also written back to the base register. In other words, the base register value is incremented or decremented by the offset value. The exclamation point signifies preindexing.



(b) Preindex

The memory address is the base register value. An offset is added to or subtracted from the base register value and the result is written back to the base register.



(c) Postindex

Instruction Formats

Define the layout of the bits of an instruction, in terms of its constituent fields

Must include an opcode and, implicitly or explicitly, indicate the addressing mode for each operand

For most instruction sets more than one instruction format is used

Instruction Length

- Most basic design issue
- Affects, and is affected by:
  - Memory size
  - Memory organization
  - Bus structure
  - Processor complexity
  - Processor speed
- **Should be equal to the memory-transfer length** or one should be a multiple of the other
- **Should be a multiple of the character length**, which is usually **8 bits**, and of the length of fixed-point numbers

Allocation of Bits

- Number of addressing modes
- Number of operands
- Register versus memory
- Number of register sets
- Address range
- Address granularity

For a given instruction length, there is clearly a trade-off between the number of opcodes and the power of the addressing capability.

PDP-8 Instruction Format

Memory Reference Instructions

Opcode			D/I	Z/C	Displacement						
0		2	3	4	5						11

Input/Output Instructions

1	1	0	Device						Opcode		
0		2	3					8	9		11

Register Reference Instructions

Group 1 Microinstructions

1	1	1	0	CLA	CLL	CMA	CML	RAR	RAL	BSW	IAC
0	1	2	3	4	5	6	7	8	9	10	11

Group 2 Microinstructions

1	1	1	1	CLA	SMA	SZA	SNL	RSS	OSR	HLT	0
0	1	2	3	4	5	6	7	8	9	10	11

Group 3 Microinstructions

1	1	1	1	CLA	MQA	0	MQL	0	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11

D/I = Direct/Indirect address
Z/C = Page 0 or Current page
CLA = Clear Accumulator
CLL = Clear Link
CMA = CoMplement Accumulator
CML = CoMplement Link
RAR = Rotate Accumultator Right
RAL = Rotate Accumulator Left
BSW = Byte SWap

IAC = Increment ACcumulator
SMA = Skip on Minus Accumulator
SZA = Skip on Zero Accumulator
SNL = Skip on Nonzero Link
RSS = Reverse Skip Sense
OSR = Or with Switch Register
HLT = HaLT
MQA = Multiplier Quotient into Accumulator
MQL = Multiplier Quotient Load

Figure 11.5 PDP-8 Instruction Formats

PDP-10 Instruction Format

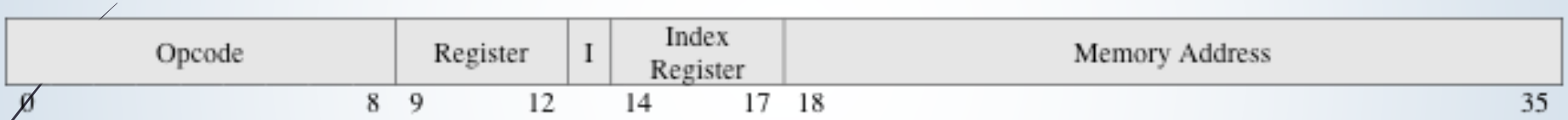


Figure 11.6 PDP-10 Instruction Format

Orthogonality: Orthogonality is a principle by which two variables are independent of each other. In the context of an instruction set, the term indicates that other elements of an instruction are independent of (not determined by) the opcode.

Variable-Length Instructions

- Variations can be provided efficiently and compactly
- Increases the complexity of the processor
- Does not remove the desirability of making all of the instruction lengths integrally related to word length
  - Because the processor does not know the length of the next instruction to be fetched a typical strategy is to fetch a number of bytes or words equal to at least the longest possible instruction
  - Sometimes multiple instructions are fetched

x86 Instruction Format

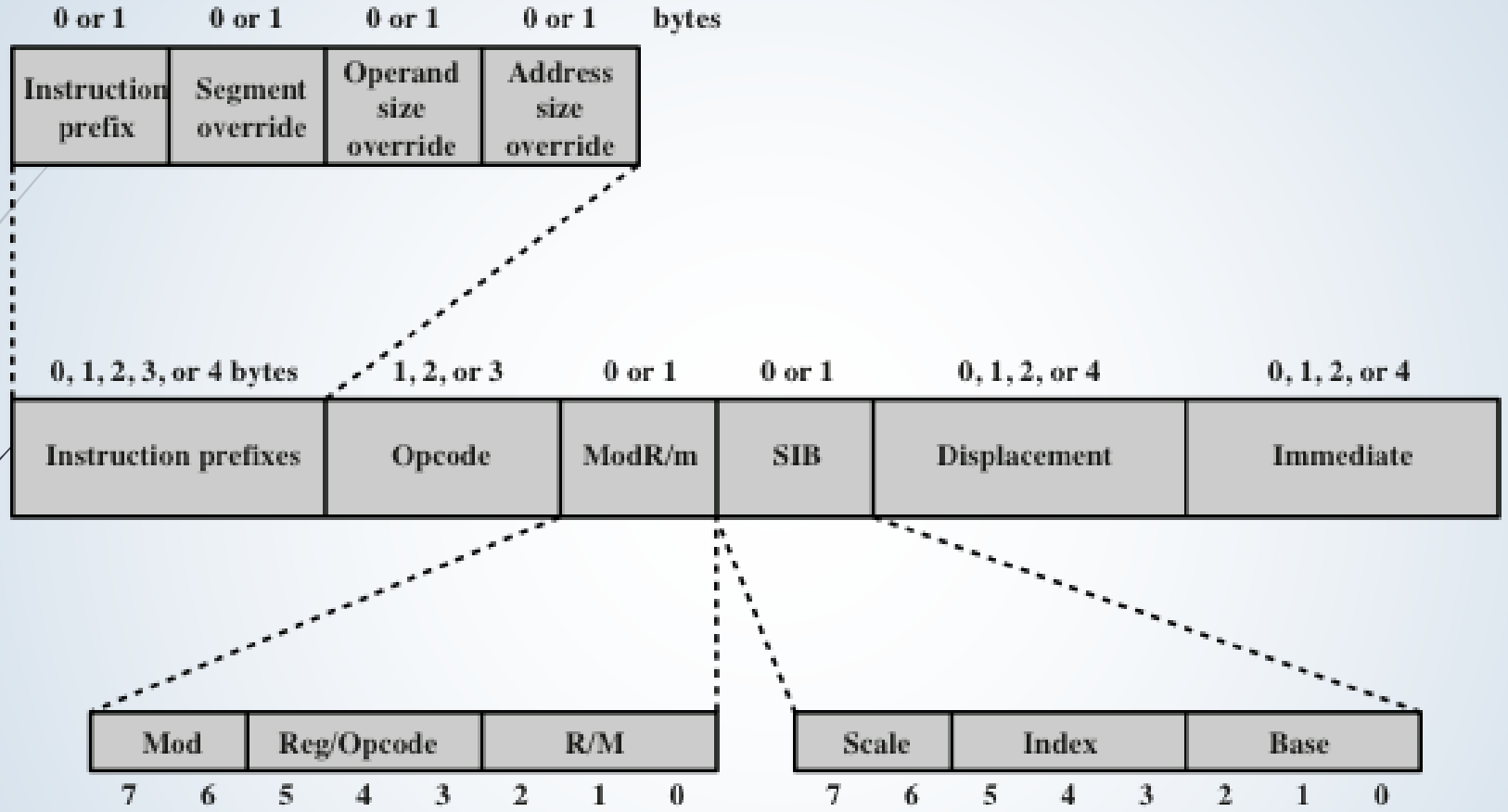


Figure 13.9 x86 Instruction Format

Thumb Instruction Set

The Thumb instruction set is a re-encoded subset of the ARM instruction set. Thumb is designed to increase the performance of ARM implementations that use a 16-bit or narrower memory data bus

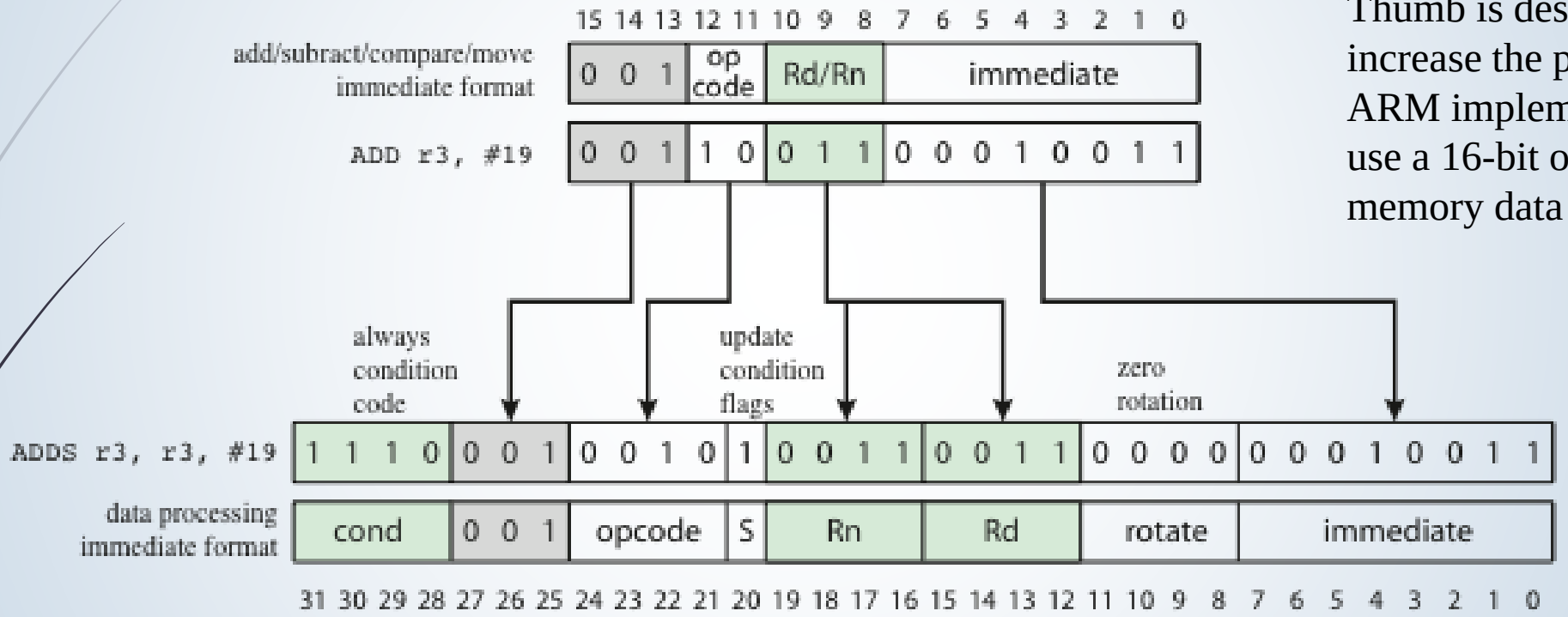


Figure 13.12 Expanding a Thumb ADD Instruction into its ARM Equivalent

Assembler

Address	Contents			
101	0010	0010	101	2201
102	0001	0010	102	1202
103	0001	0010	103	1203
104	0011	0010	104	3204
201	0000	0000	201	0002
202	0000	0000	202	0003
203	0000	0000	203	0004
204	0000	0000	204	0000

(a) Binary program

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

(b) Hexadecimal 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

(c) Symbolic 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

Figure 11.13 Computation of the Formula N = I + J+ K

Summary

- Chapter 13
  - Addressing modes
    - Immediate addressing
    - Direct addressing
    - Indirect addressing
    - Register addressing
    - Register indirect addressing
    - Displacement addressing
    - Stack addressing
- Instruction Sets: Addressing Modes and Formats
  - x86 addressing modes
  - ARM addressing modes
  - Instruction formats
    - Instruction length
    - Allocation of bits
    - Variable-length instructions
  - X86 instruction formats
  - ARM instruction formats