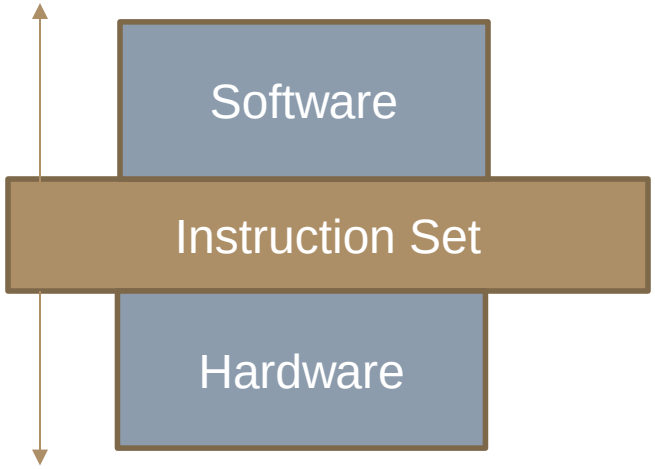
Fundamentals of Computer Architecture

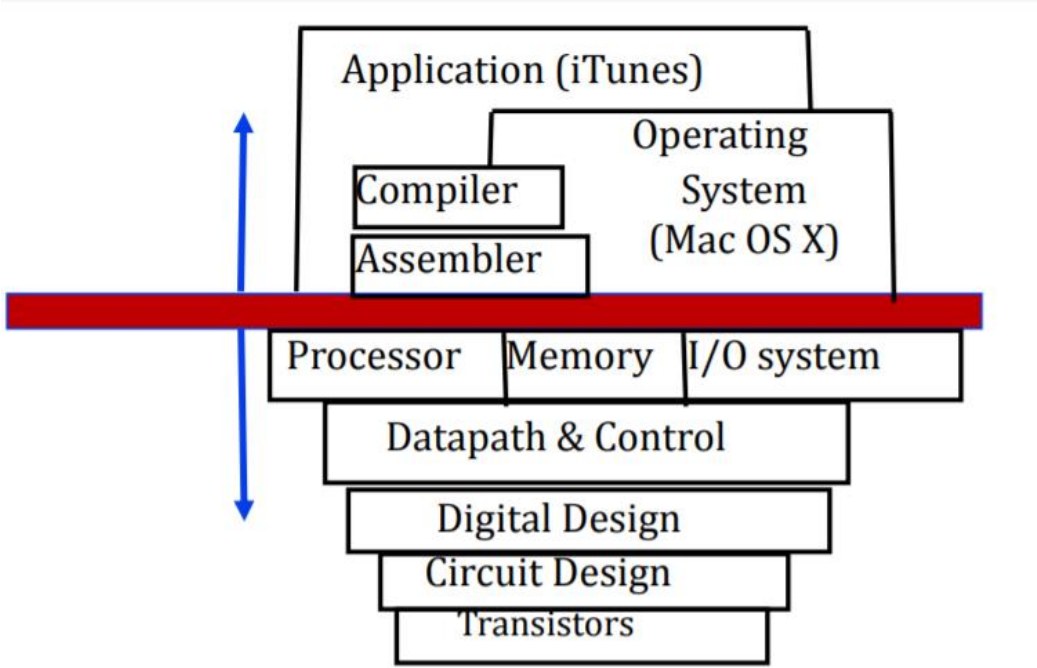
- ISA
- Instruction Formats
- Instruction types
- Addressing Modes
- Assembly language programming

Instruction Set Architecture



- Interface between hardware and software
- Interface between high level language and machine level language

ISA…


Application (iTunes)
Operating System (Mac OS X)
Compiler
Assembler
Processor
Memory
I/O system
Datapath & Control
Digital Design
Circuit Design
Transistors

Elements of an ISA…

- Instructions
  - Instruction Formats
  - Addressing Modes
  - Condition Codes
  - Instruction sets

Characteristics of good ISA

- Must be Clear
- Less usage of complex instructions
- Ease of compilation
- Ease of implementation

ADDRESSING MODES

- In general, a program operates on data that reside in the computer's memory.
- These data can be organized in a variety of ways. If we want to keep track of students' names, we can write them in a list.
- If we want to associate information with each name, for example to record telephone numbers or marks in various courses, we may organize this information in the form of a table. Programmers use organizations called *data structures* to represent the data used in computations.
- These include lists, linked lists, arrays, queues, and so on.

Addressing mode ?

- The different ways in which the location of an operand is specified in an instruction are referred to as *addressing modes*.

Generating Memory Addresses

- How to specify the address of branch target?
- Can we give the memory operand address directly in a single Add instruction in the loop?
- Use a register to hold the address of NUM1; then increment by 4 on each pass through the loop.

Direct methods

The address of the operand is specified directly in the instruction

- Register
- Absolute
- Immediate

Direct methods

Register mode

- The operand is the contents of a processor register; the name (address) of the register is given in the instruction. It is used to access the local variables in the program.

Absolute mode

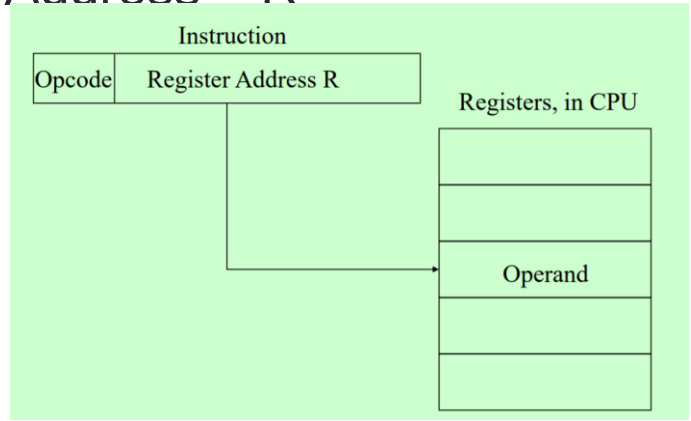
- The operand is in a memory location; the address of this location is given explicitly in the instruction. It is also called as *Direct mode.* It also used to access the global variables in the program.

Example instruction for register and absolute mode

- Move LOC, R2

Register Mode

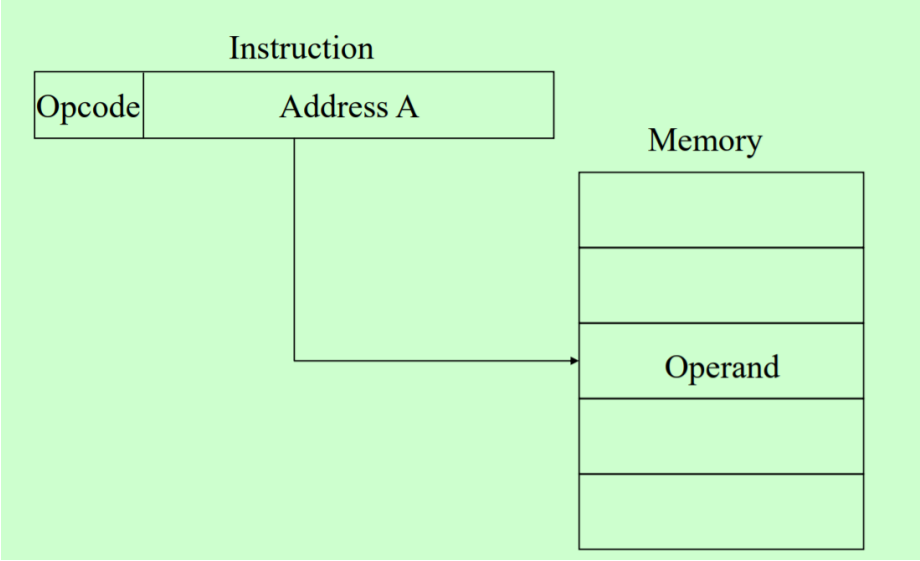
- Register
  - Indicate which register holds the operand
  - Operand is held in register named in address field
  - Effective Address = R



Absolute Addressing Mode

- Direct addressing mode

Use the given address to access a memory location



Immediate mode

Address and data constants can be represented in assembly language using the Immediate mode. The operand is given explicitly in the instruction.

For example, the instruction

- Move 200*immediate*, R0
- Move #200, R0

Example

A = B + 6

- contains the constant 6. Assuming that A and B have been declared earlier as variables and may be accessed using the Absolute mode, this statement may be compiled as follows:
- Move B, R1
- Add #6, R1
- Move R1, A

Indirect Methods

- The instruction does not give the operand's address explicitly
- The address of the operands are specified indirectly and referred to as Effective Address (EA)
  - Indirection & pointers
  - Indexing and arrays
  - Relative mode
  - Auto Increment and Auto Decrement mode

Indirection and Pointers

- The effective address of the operand is the contents of a register or memory location whose address appears in the instruction
- Indirection is denoted by enclosing the name of register or memory location given in the instruction with a parenthesis

Ex:

A = *B, where B is a pointer variable

The statement gets compiled into

Move B,R1

Move (R1),A

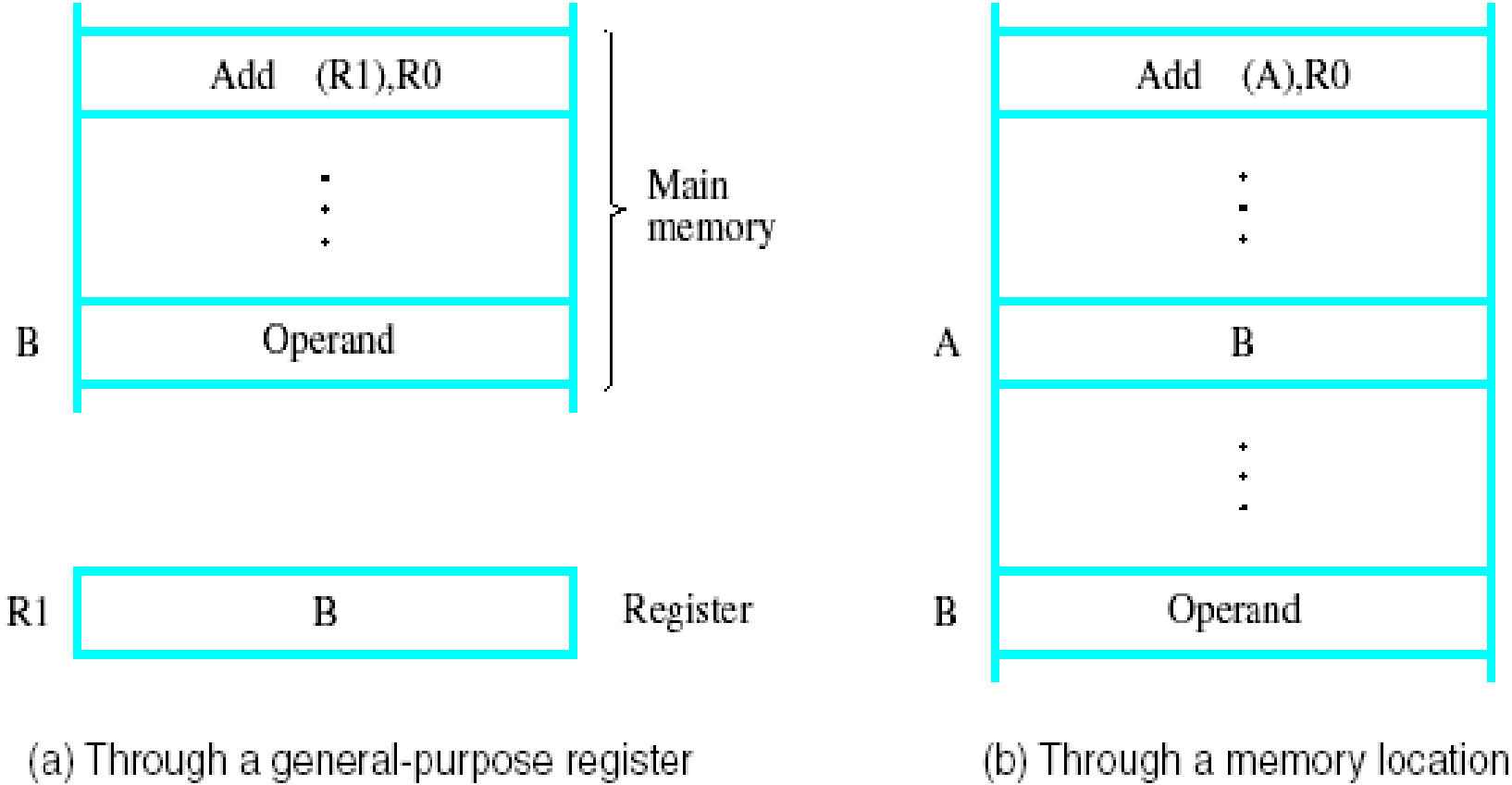
100 5

R1 =100

(R1) = 5

A= 5

Indirect Mode of Adressing



(a) Through a general-purpose register

(b) Through a memory location

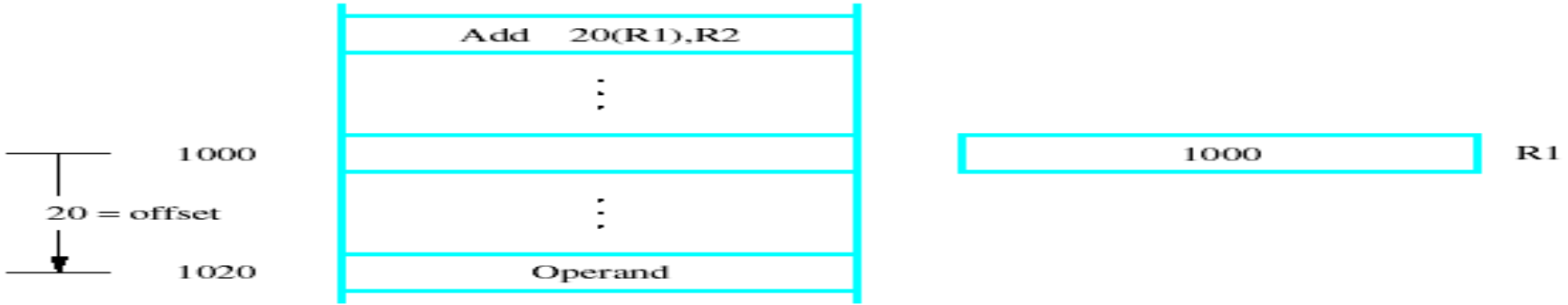
Indexing and Arrays

- Index mode – the effective address (EA) of the operand is generated by adding a constant value to the contents of a register.
- Index register
- $X(R_i)$: $EA = X + [R_i]$ where X is the offset or displacement
- The constant X may be given either as an explicit number or as a symbolic name representing a numerical value.
- If X is shorter than a word, sign-extension is needed.

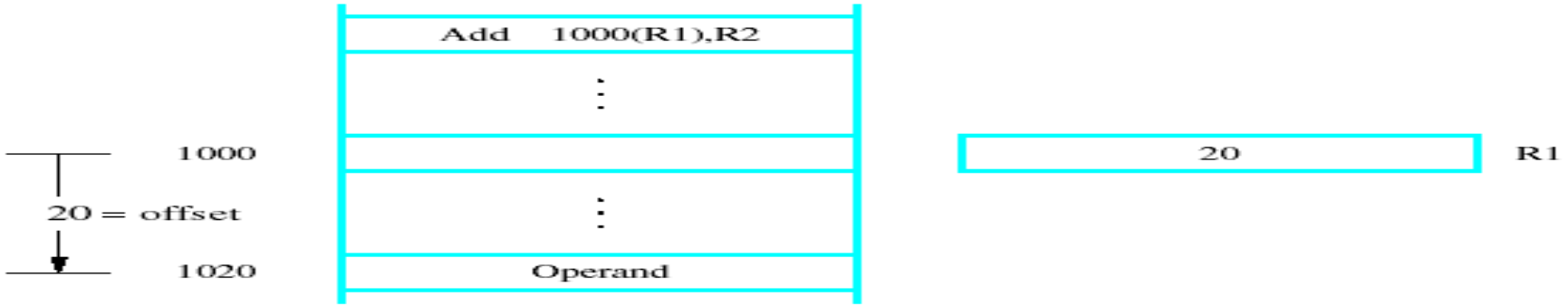
Indexing and Arrays

- In general, the Index mode facilitates access to an operand whose location is defined relative to a reference point within the data structure in which the operand appears.
- Several variations:
  (R_i, R_j): $EA = [R_i] + [R_j]$
  $X(R_i, R_j)$: $EA = X + [R_i] + [R_j]$

Index add Mode



(a) Offset is given as a constant



(b) Offset is in the index register

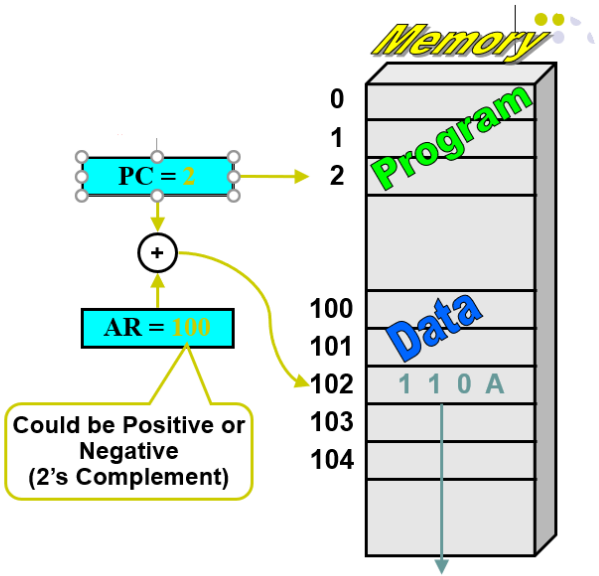
$EA = X + [R_i]$

Relative Addressing

- Relative mode – the effective address is determined by the Index mode using the program counter in place of the general-purpose register.
- X(PC) – note that X is a signed number
- Branch>0 LOOP
- This location is computed by specifying it as an offset from the current value of PC.
- Branch target may be either before or after the branch instruction, the offset is given as a singed num.

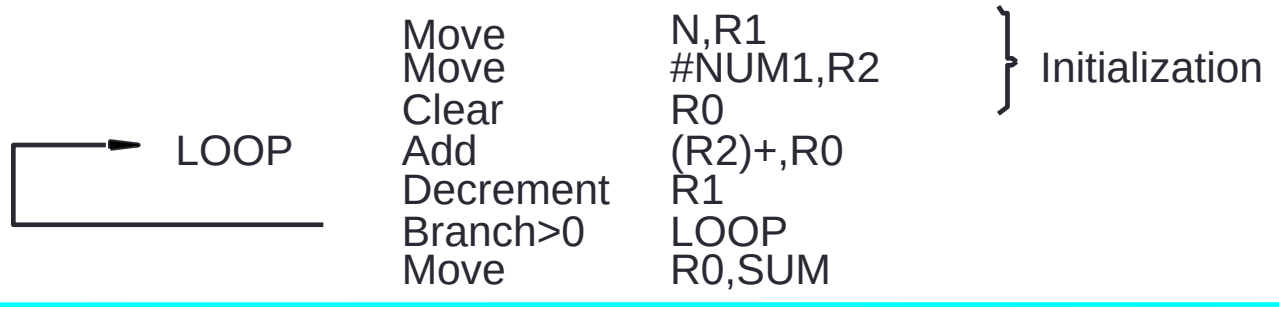
Relative Mode

- Relative Address
  - *EA* = PC + Relative Addr



Additional Modes

- Autoincrement mode – the effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically incremented to point to the next item in a list.
- $(R_i)+$. The increment is 1 for byte-sized operands, 2 for 16-bit operands, and 4 for 32-bit operands.
- Autodecrement mode: $-(R_i)$ – decrement first



The Autoincrement addressing mode.

Addressing Modes- Summary

- The different ways in which the location of an operand is specified in an instruction are referred to as addressing modes.

Name	Assembler syntax	Addressing function
Immediate	#Value	Operand = Value
Register	R*i*	EA = R*i*
Absolute (Direct)	LOC	EA = LOC
Indirect	(R*i*) (LOC)	EA = [R*i*] EA = [LOC]
Index	X(R*i*)	EA = [R*i*] + X
Base with index	(R*i*,R*j*)	EA = [R*i*] + [R*j*]
Base with index and offset	X(R*i*,R*j*)	EA = [R*i*] + [R*j*] + X
Relative	X(PC)	EA = [PC] + X
Autoincrement	(R*i*)+	EA = [R*i*] ; Increment R*i*
Autodecrement	–(R*i*)	Decrement R*i* ; EA = [R*i*]

Assembly Language Notation

Assembly Language

- It is the format to represent the machine instructions and programs.
- MOVE LOC, R1 ;it transfers from memory location LOC to R1
- ADD R1, R2, R3 ;Add two numbers placed in R1 and R2;

Sum is placed in R3

INSTRUCTION TYPES

Instruction Types

- Data Transfer Instructions

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Instruction Types…

- Data manipulation instructions
  - Arithmetic instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate	NEG

Logical & Bit Manipulation

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

Shift instructions

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

Instruction Types…

- Program Control Instructions

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (Subtract)	CMP
Test (AND)	TST

Instruction Types…

- Conditional Branch Instructions

Mnemonic	Branch Condition	Tested Condition
BZ	Branch if zero	Z = 1
BNZ	Branch if not zero	Z = 0
BC	Branch if carry	C = 1
BNC	Branch if no carry	C = 0
BP	Branch if plus	S = 0
BM	Branch if minus	S = 1
BV	Branch if overflow	V = 1
BNV	Branch if no overflow	V = 0