SNS COLLEGE OF TECHNOLOGY

(Autonomous)

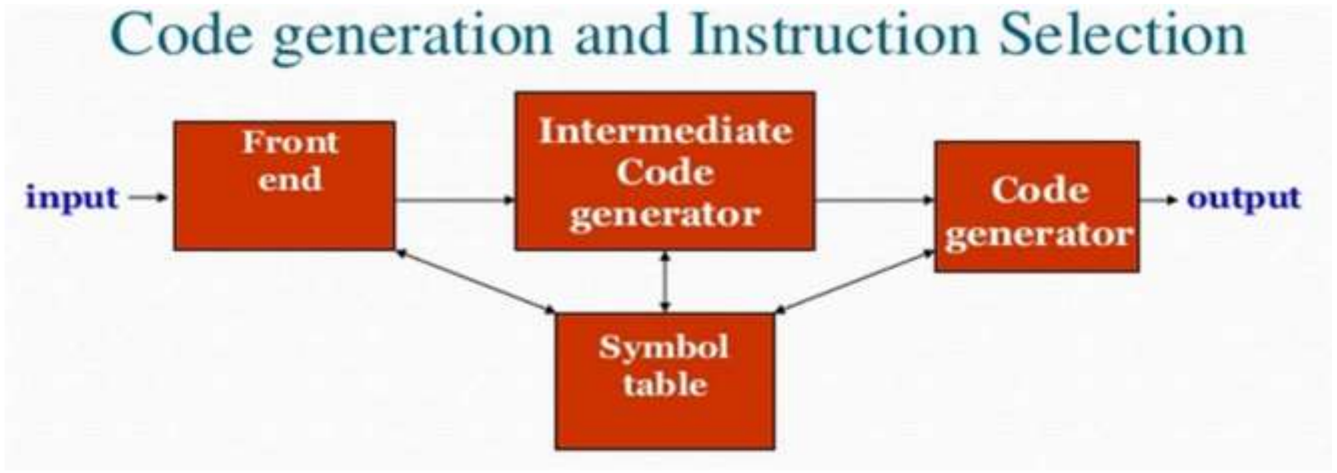
COIMBATORE-35

Code Generation

Issues in the Design of Code Generator

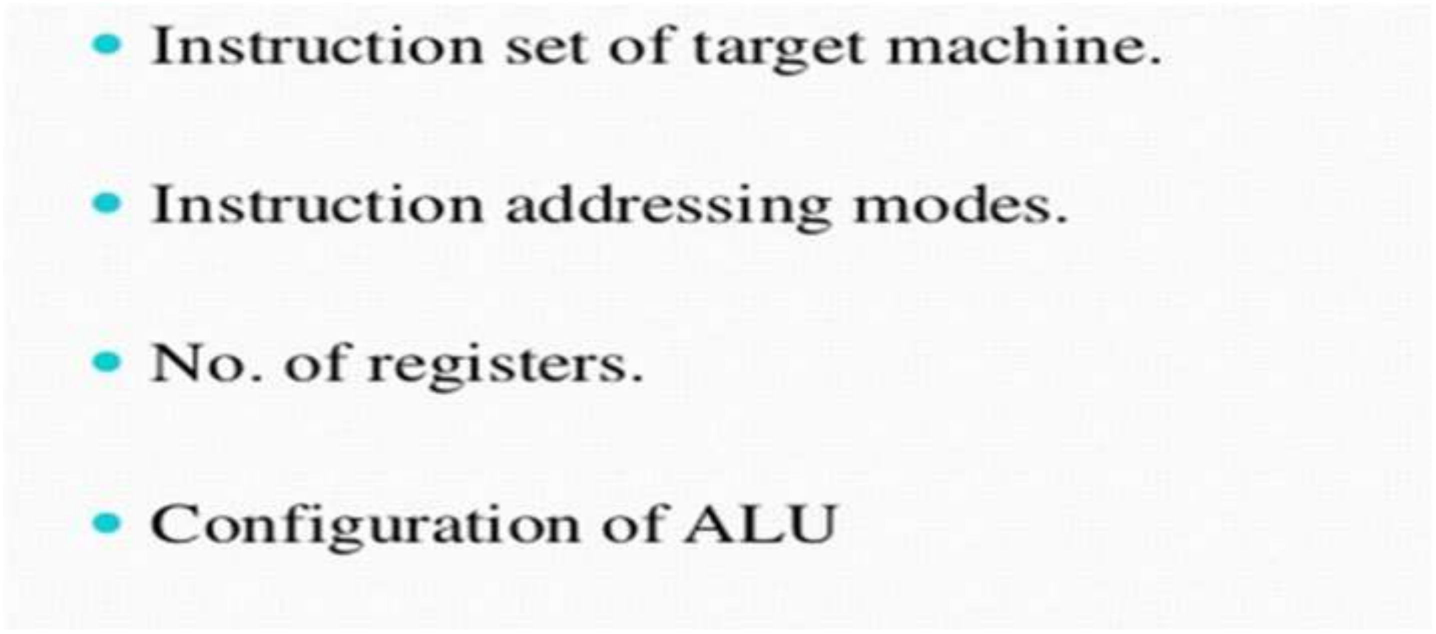
Code Generation

- ➢ Final phase of Compiler Design
- ➢ Optimized intermediate code is provided as input
- ➢ It generates target code
- ➢ Output code must be correct
- ➢ Output code must be high quality
- ➢ Code generator should run efficiently


Code generation and Instruction Selection
input
Front end
Intermediate Code generator
Code generator
output
Symbol table

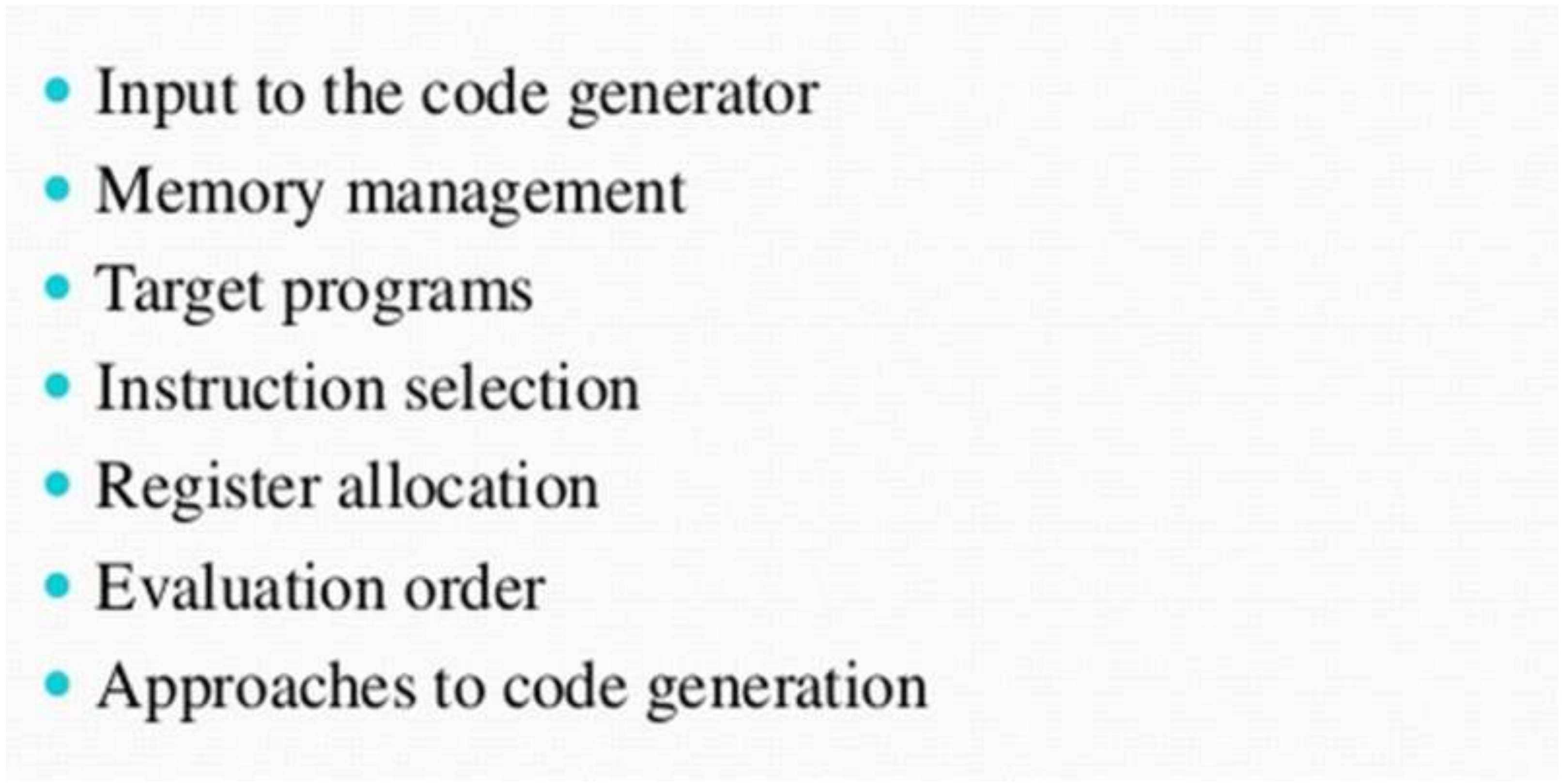
Code Generation

Prerequisites



- Instruction set of target machine.
- Instruction addressing modes.
- No. of registers.
- Configuration of ALU

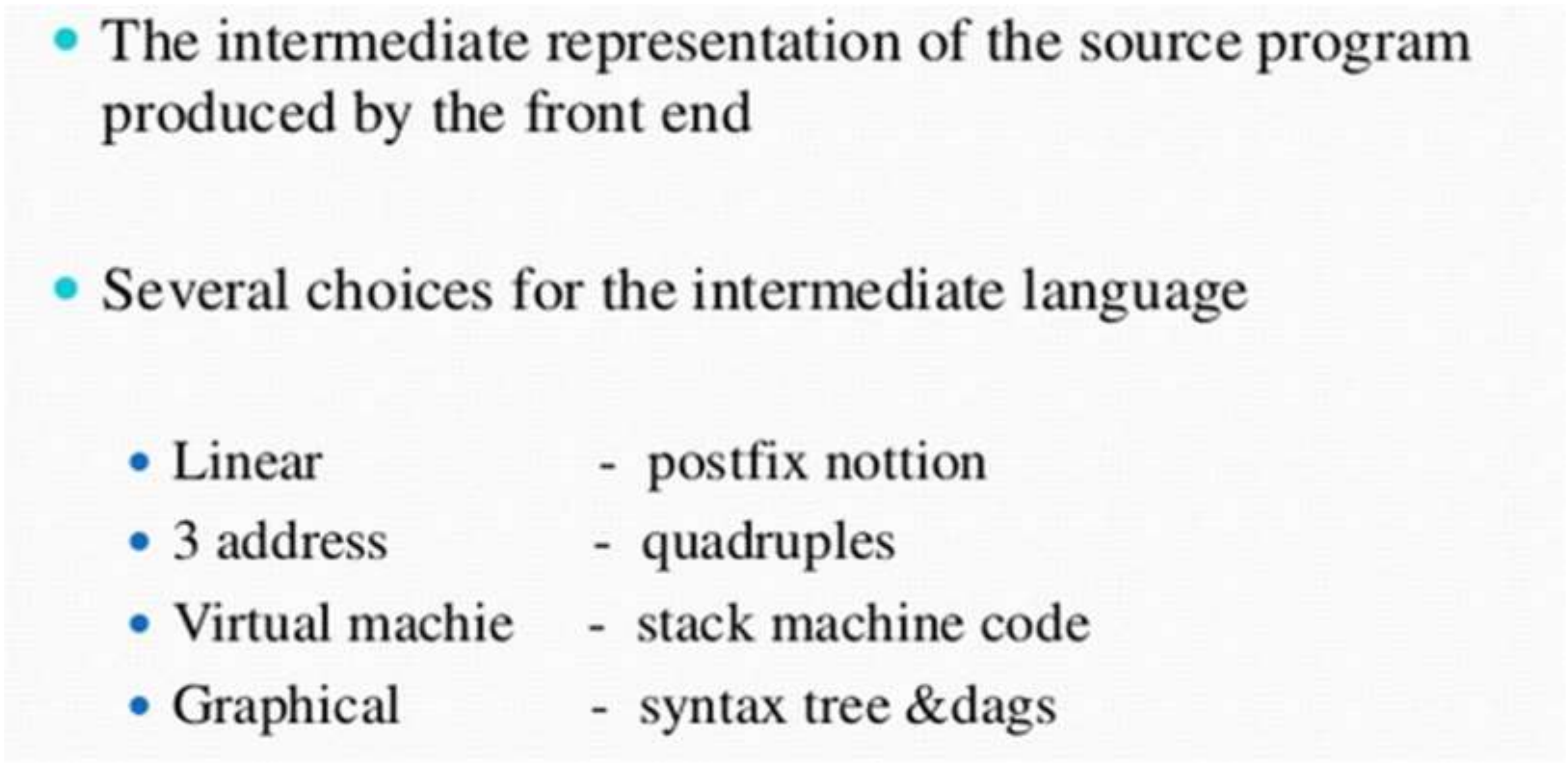
Issues in the Design of Code Generator



- Input to the code generator
- Memory management
- Target programs
- Instruction selection
- Register allocation
- Evaluation order
- Approaches to code generation

Issues in the Design of Code Generator

Input to the code Generator



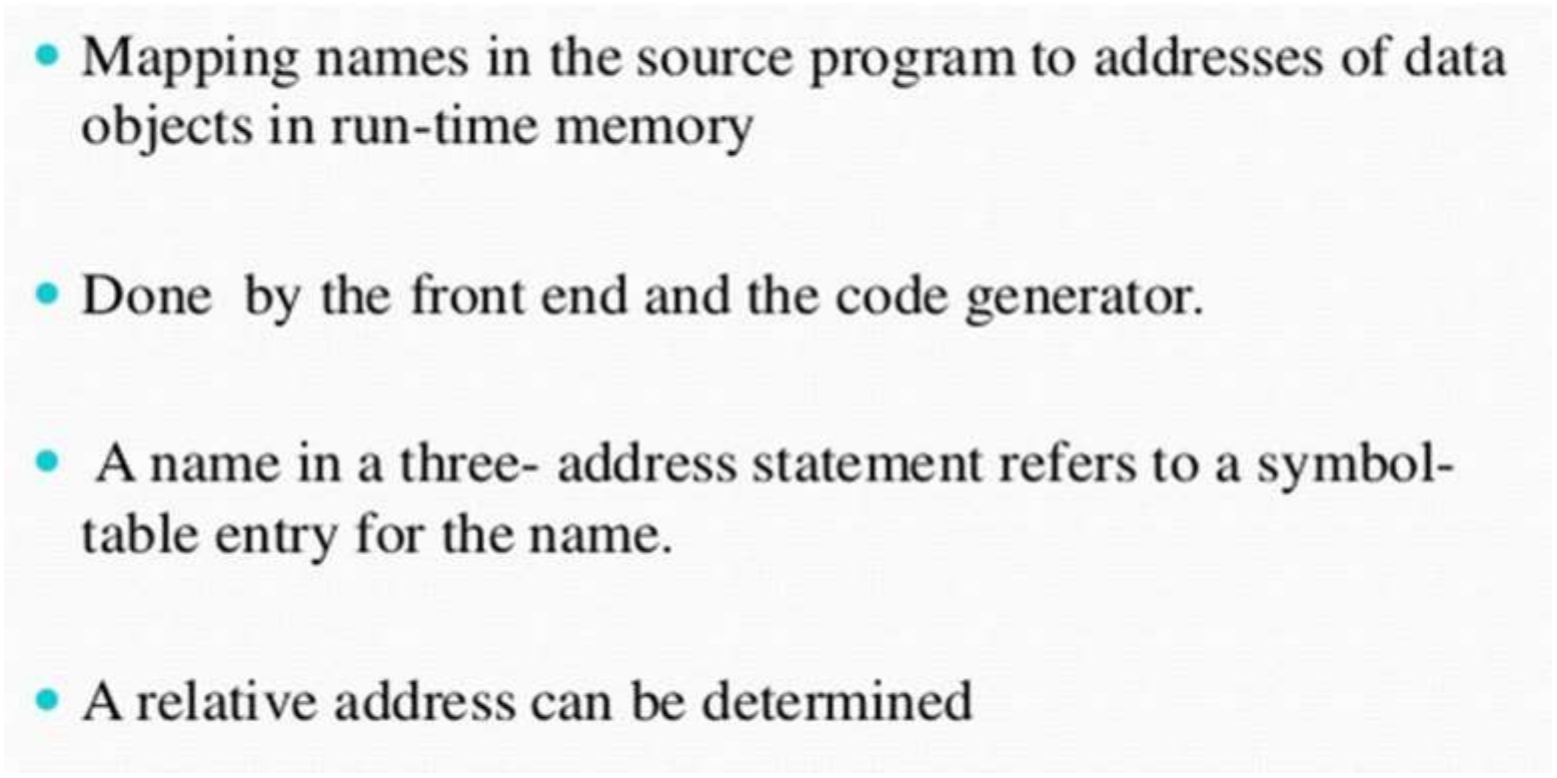
- The intermediate representation of the source program produced by the front end

- Several choices for the intermediate language

    - Linear - postfix nottion
    - 3 address - quadruples
    - Virtual machie - stack machine code
    - Graphical - syntax tree &dags

Issues in the Design of Code Generator

Memory Management



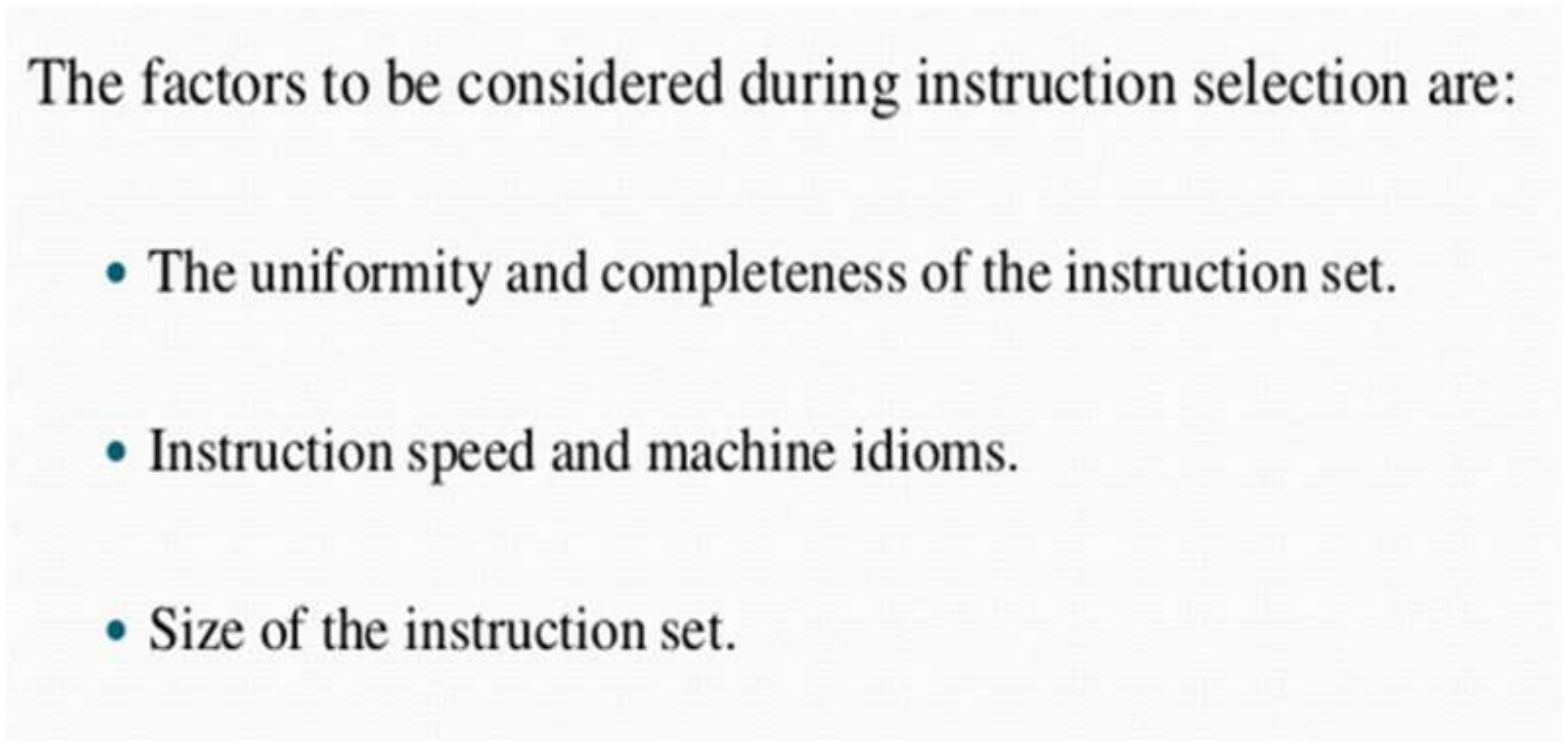
- Mapping names in the source program to addresses of data objects in run-time memory
- Done by the front end and the code generator.
- A name in a three- address statement refers to a symbol-table entry for the name.
- A relative address can be determined

Issues in the Design of Code Generator

- **Target program:**
- • The output of the code generator is the target program. The output may be : a. Absolute machine language
- - It can be placed in a fixed memory location and can be executed immediately. b. Relocatable machine language
- - It allows subprograms to be compiled separately.
- c. Assembly language
- - Code generation is made easier.

Issues in the Design of Code Generator

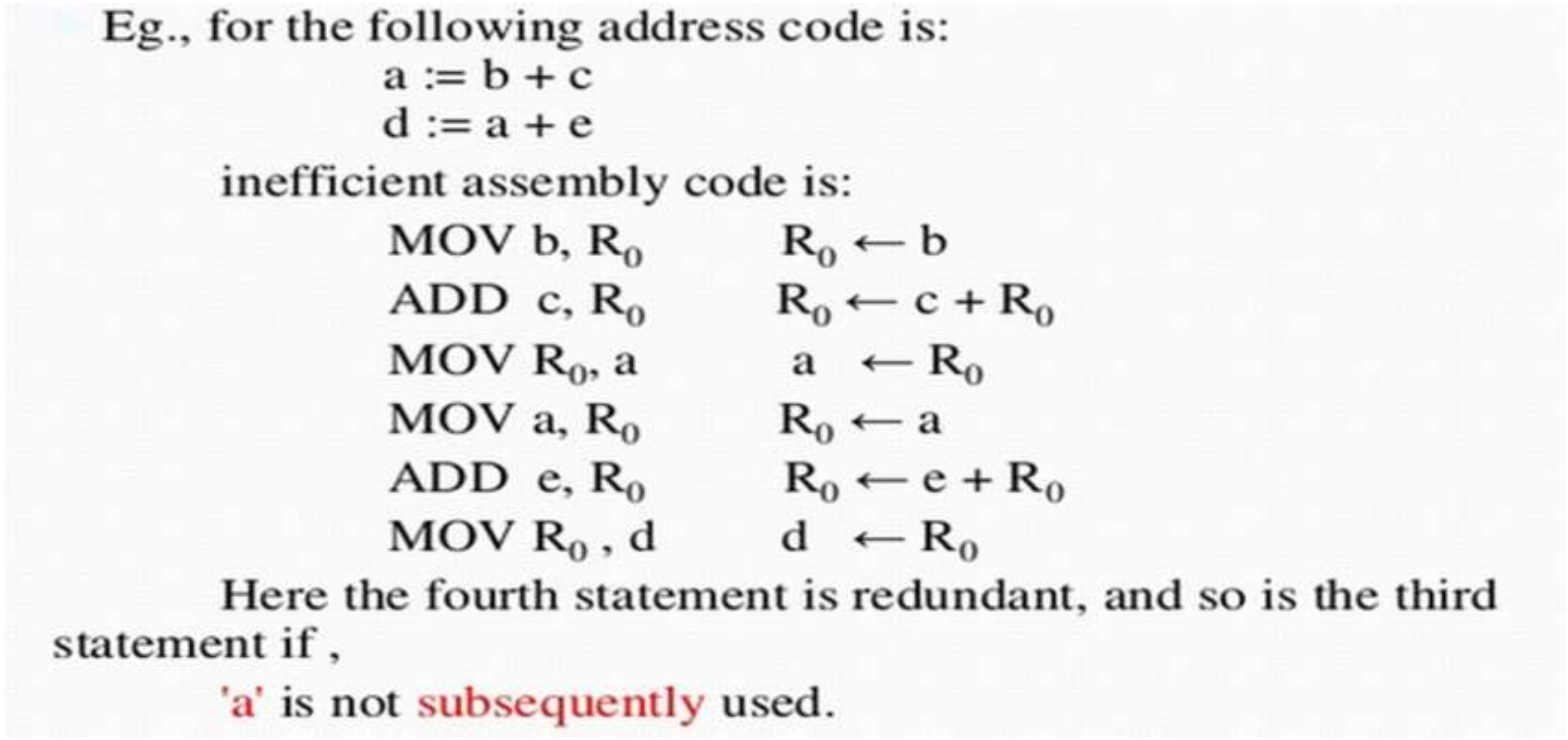
Instruction Selection


The factors to be considered during instruction selection are:

- The uniformity and completeness of the instruction set.
- Instruction speed and machine idioms.
- Size of the instruction set.

Issues in the Design of Code Generator

Instruction Selection



Eg., for the following address code is:

$a := b + c$

$d := a + e$

inefficient assembly code is:

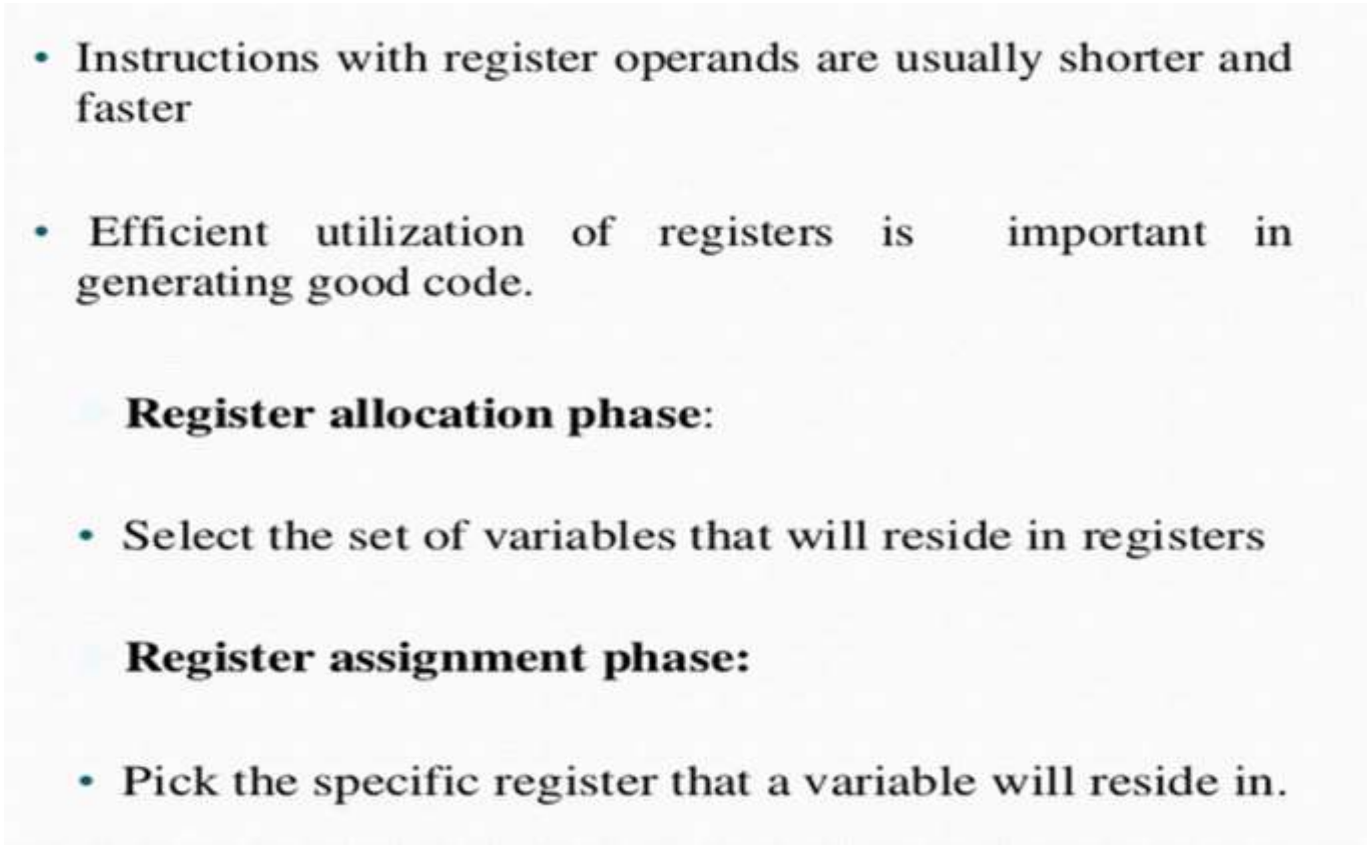
MOV b, R_0	$R_0 \leftarrow b$
ADD c, R_0	$R_0 \leftarrow c + R_0$
MOV R_0, a	$a \leftarrow R_0$
MOV a, R_0	$R_0 \leftarrow a$
ADD e, R_0	$R_0 \leftarrow e + R_0$
MOV R_0 , d	$d \leftarrow R_0$

Here the fourth statement is redundant, and so is the third statement if ,

'a' is not subsequently used.

Issues in the Design of Code Generator

Register Allocation



- Instructions with register operands are usually shorter and faster
- Efficient utilization of registers is important in generating good code.

Register allocation phase:

- Select the set of variables that will reside in registers

Register assignment phase:

- Pick the specific register that a variable will reside in.

Issues in the Design of Code Generator

- **Evaluation order**
- • The order in which the computations are performed can affect the efficiency of the target code.
-
- Some computation orders require fewer registers to hold intermediate results than others.

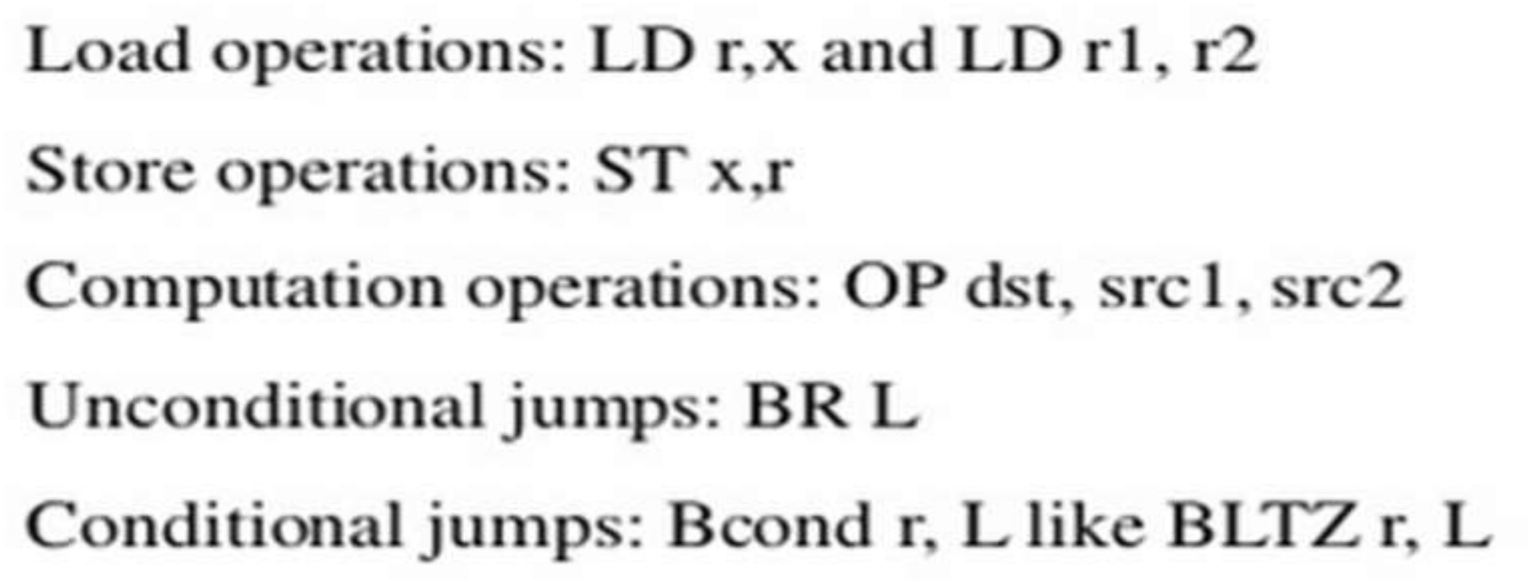
Target Machine

Target Machine

- The target computer is a type of byte-addressable machine. It has 4 bytes to a word.
- The target machine has n general purpose registers, R0, R1,...., Rn-1. It also has two-address instructions of the form:
- **op source, destination**
- Where, op is used as an op-code and source and destination are used as a data field.
- It has the following op-codes:
  ADD (add source to destination)
  SUB (subtract source from destination)
  MOV (move source to destination)

Target Machine

Simple Target Machine Model



Load operations: LD r,x and LD r1, r2

Store operations: ST x,r

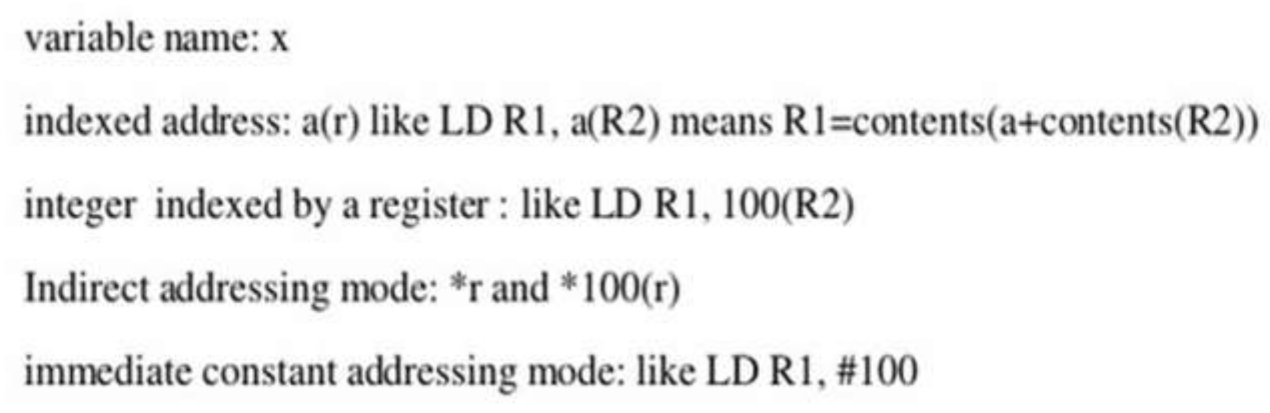
Computation operations: OP dst, src1, src2

Unconditional jumps: BR L

Conditional jumps: Bcond r, L like BLTZ r, L

Target Machine

Addressing Modes



variable name: x

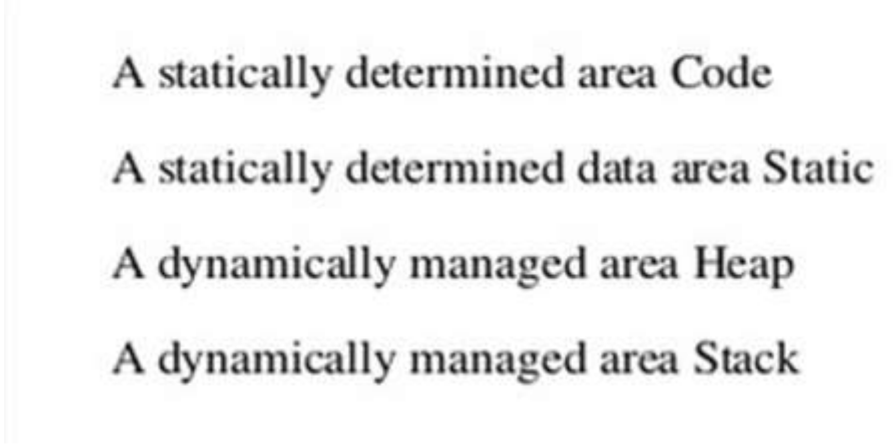
indexed address: a(r) like LD R1, a(R2) means R1=contents(a+contents(R2))

integer indexed by a register : like LD R1, 100(R2)

Indirect addressing mode: *r and *100(r)

immediate constant addressing mode: like LD R1, #100

Addresses in Target Code



A statically determined area Code

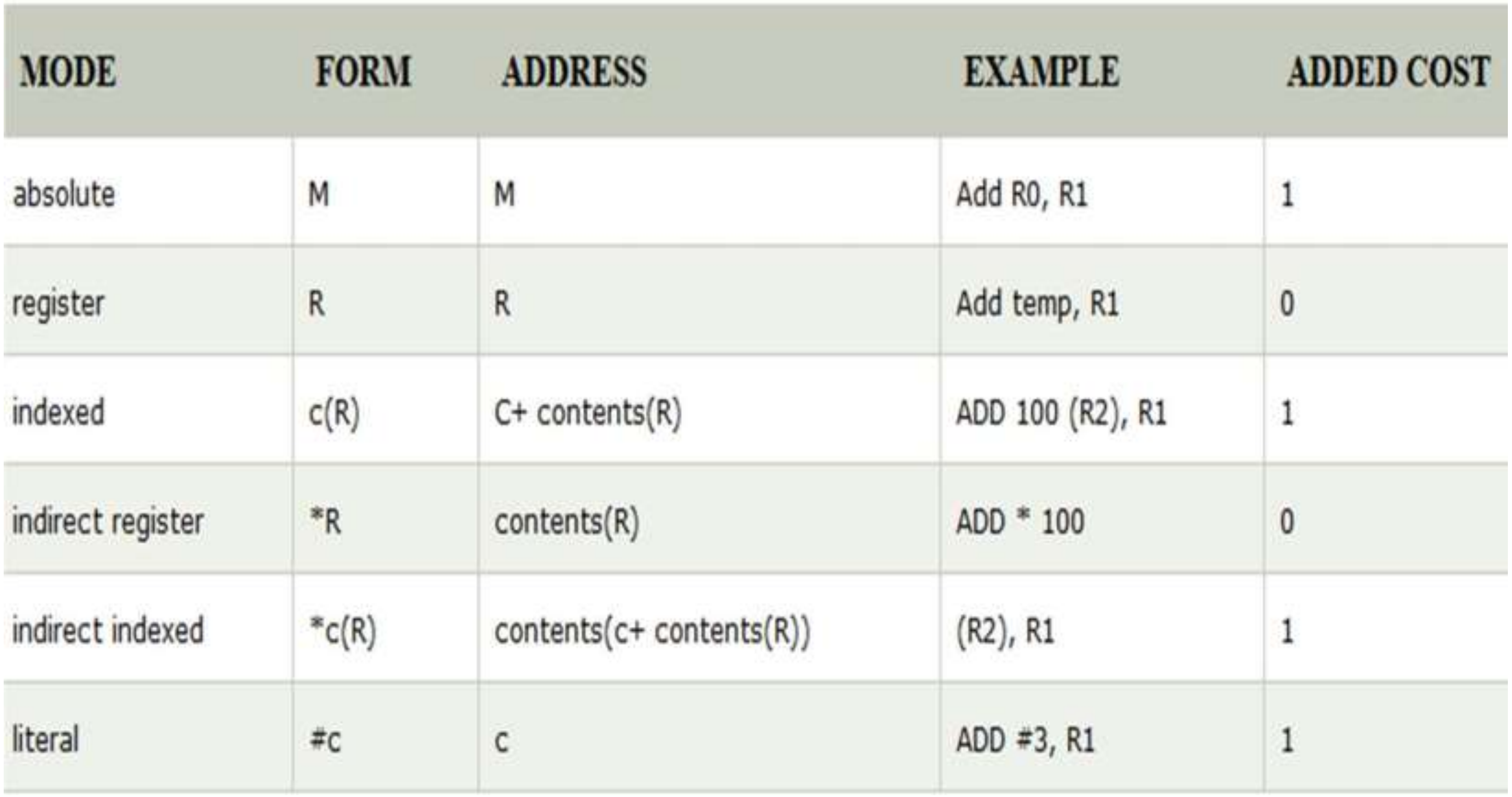
A statically determined data area Static

A dynamically managed area Heap

A dynamically managed area Stack

Target Machine

- The source and destination of an instruction can be specified by the combination of registers and memory location with address modes



MODE	FORM	ADDRESS	EXAMPLE	ADDED COST
absolute	M	M	Add R0, R1	1
register	R	R	Add temp, R1	0
indexed	c(R)	C+ contents(R)	ADD 100 (R2), R1	1
indirect register	*R	contents(R)	ADD * 100	0
indirect indexed	*c(R)	contents(c+ contents(R))	(R2), R1	1
literal	#c	c	ADD #3, R1	1

Target Machine

- Here, cost 1 means that it occupies only one word of memory.
- Each instruction has a cost of 1 plus added costs for the source and destination.
- Instruction cost = 1 + cost is used for source and destination mode.
- Example:
- 1. Move register to memory R0 → M
- MOV R0, M cost = 1+1+1   (since address of memory locatio n M is in word following the instruction)

Target Machine

2. Indirect indexed mode:

MOV * 4(R0), M

    cost = 1+1+1   (since one word for memory location M, one w ord for result of *4(R0) and one for instruction)

3. Literal Mode:

MOV #1, R0

cost = 1+1+1 = 3   (one word for constant 1 and one for instructio n)

Summarization

Summarization