Code Generation

Introduction

Phases of typical compiler and position of code generation.

<fig: 9.1 - page 513>

Since code generation is an "undecidable problem (mathematically speaking), we must be content with heuristic technique that generate "good" code (not necessarily optimal code).

Code generation must do following things:

- Produce correct code
- make use of machine architecture.
- run efficiently.

Issues in the Design of Code generator

Code generator concern with:

1. Memory management.
2. Instruction Selection.
3. Register Utilization (Allocation).
4. Evaluation order.

1. Memory Management
Mapping names in the source program to address of data object is cooperating done in pass 1 (Front end) and pass 2 (code generator).

Quadruples → address Instruction.

Local variables (local to functions or procedures) are stack-allocated in the activation record while global variables are in a static area.

2. Instruction Selection
The nature of instruction set of the target machine determines selection.

-"Easy" if instruction set is regular that is uniform and complete.
Uniform: all triple addresses
all stack single addresses.

Complete: use all register for any operation.

If we don't care about efficiency of target program, instruction selection is straight forward.

For example, the address code is:

a := b + c
d := a + e

Inefficient assembly code is:

1. MOV b, R_0 $R_0 \leftarrow b$
2. ADD c, R_0 $R_0 \leftarrow c + R_0$
3. MOV R_0, a a $\leftarrow R_0$
4. MOV a, R_0 $R_0 \leftarrow a$
5. ADD e, R_0 $R_0 \leftarrow e + R_0$
6. 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.

3. Register Allocation

Register can be accessed faster than memory words. Frequently accessed variables should reside in registers (register allocation). Register assignment is picking a specific register for each such variable.

Formally, there are two steps in register allocation:

1. Register allocation (what register?)
   This is a register selection process in which we select the set of variables that will reside in register.
2. Register assignment (what variable?)
   Here we pick the register that contain variable. Note that this is a NP-Complete problem.

Some of the issues that complicate register allocation (problem).

1. Special use of hardware for example, some instructions require specific register.

2. Convention for Software:

   For example

- Register R6 (say) always return address.
- Register R5 (say) for stack pointer.
- Similarly, we assigned registers for branch and link, frames, heaps, etc.,

3. Choice of Evaluation order

Changing the order of evaluation may produce more efficient code.
This is NP-complete problem but we can bypass this hindrance by generating code for quadruples in the order in which they have been produced by intermediate code generator.

ADD x, Y, T1
ADD a, b, T2

is legal because X, Y and a, b are different (not dependent).

The Target Machine

Familiarity with the target machine and its instruction set is a prerequisite for designing a good code generator.

Typical Architecture

Target machine is:

1. Byte addressable (factor of 4).
2. 4 byte per word.
3. 16 to 32 (or n) general purpose register.
4. Two addressable instruction of form:
   Op source, destination.
   e.g., move A, B
   add A, D

Typical Architecture:

1. Target machine is :
2. Bit addressing (factor of 1).
3. Word purpose registers.
4. Three address instruction of forms:
   Op source 1, source 2, destination
   e.g.,
   ADD A, B, C

- Byte-addressable memory with 4 bytes per word and n general-purpose registers, R_0, R_1, . . . , R_{n-1}. Each integer requires 2 bytes (16-bits).

- Two address instruction of the form
  mnemonic source, destination

MODE	FORM	ADDRESS	EXAMPLE	ADDED-COST
Absolute	M	M	ADD R_0, R_1	1
register	R	R	ADD temp, R_1	0
Index	c (R)	c + contents (R)	ADD 100(R_2), R_1	1
Indirect register	*R	contents (R)	ADD * R_2, R_1	0
Indirect Index	*c (R)	contents (c + contents (R)	ADD * 100(R_2), R_1	1
Literal	# c	constant c	ADD # 3, R_1	1

Instruction costs:

Each instruction has a cost of 1 plus added costs for the source and destination.

=> cost of instruction = 1 + cost associated the source and destination address mode.

This cost corresponds to the length (in words) of instruction.

Examples

1. Move register to memory $R_0 \leftarrow M$.
   MOV R_0, M cost = 1+1 = 2.
2. Indirect indexed mode:
   MOV * 4 (R_0), M
   cost = 1 plus indirect index plus instruction word
   = 1 + 1 + 1 = 3
3. Indexed mode:
   MOV 4(R_0), M
   cost = 1 + 1 + 1 = 3
4. Litetral mode:
   MOV #1, R_0
   cost = 1 + 1 = 2
5. Move memory to memory
   MOV m, m cost = 1 + 1 + 1 = 3