Unit - V

Object Code Generation| Error Recovery

Object Code Generation

- Object code forms
- Machine dependent code optimization (Peephole Optimization)
- Register allocation and assignment
- Generic code generation algorithms

Error Recovery

- Various errors in phases and recovery of errors in compilation
- Introduction to tools of compiler

Dr. R. Madana Mohana
Professor, Artificial Intelligence & Data Science | I/c-Head, Artificial Intelligence & Machine Learning
CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY
Hyderabad - 500 075, Telangana, INDIA
www.cbit.ac.in

Register Allocation and Assignment

- Global Register Allocation
- Usage Counts
- Register Assignment for Outer Loops
- Register Allocation by Graph Coloring

Register Allocation and Assignment

- Efficient utilization of the limited set of registers is important to generate good code
- Registers are assigned by
  - *Register allocation* to select the set of variables that will reside in registers at a point in the code
  - *Register assignment* to pick the specific register that a variable will reside in
- Finding an optimal register assignment in general is NP-complete

Example

```
t:=a*b
t:=t+a
t:=t/d
```

{ R1=t }

```
MOV  a,R1
MUL  b,R1
ADD  a,R1
DIV  d,R1
MOV  R1,t
```

```
t:=a*b
t:=t+a
t:=t/d
```

{ R0=a, R1=t }

```
MOV  a,R0
MOV  R0,R1
MUL  b,R1
ADD  R0,R1
DIV  d,R1
MOV  R1,t
```

Register Allocation and Assignment

- Instructions involving only register operands are shorter and faster than those involving memory operands. Therefore, efficient utilization of registers is important in generating good code.
- **Register Allocation:** What values in program should reside in registers.
- **Register Assignment:** Which register each value should reside.
- One approach to **register allocation** and **assignment** is to assign specific values in an object program to certain registers.

Register Allocation and Assignment

For example, a decision can be made to assign base addresses to one group of registers, arithmetic computations to another, the top of the run-time stack to a fixed register, and so on.

Advantages of this Approach: It simplifies the design of a compiler.

Disadvantages: Applied to strictly, it uses registers inefficiently; certain registers may go unused over substantial portions of code, while unnecessary loads and stores are generated.

Register Allocation and Assignment

Global Register Allocation

- The code-generation algorithm used registers to hold values for the duration of a single basic block. However, all live variables were stored at the end of each block. To save some of these stores and corresponding loads, we might arrange to assign registers to frequently used variables and keep these registers consistent across block boundaries (globally).
- Since programs spend most of their time in inner loops, a natural approach to global register assignment is to try to keep a frequently used value in a fixed register throughout a loop.

Register Allocation and Assignment

Global Register Allocation

- One strategy for global register allocation is to assign some fixed number of registers to hold the most active values in each inner loop. The selected values may be different in different loops. Registers not already allocated may be used to hold values local to one block.
  - ✓ This approach has the drawback that the fixed number of registers is not always the right number to make available for global register allocation.

Register Allocation and Assignment

Usage Counts

- We shall assume that the savings to be realized by keeping a variable `x` in a register for the duration of a loop `L` is one unit of cost for each reference to `x` if `x` is already in a register.
- However, if we use the approach discussed earlier to generate code for a block, there is a good chance that after `x` has been computed in a block it will remain in a register if there are subsequent uses of `x` in that block.
- Thus we count a savings of one for each use of `x` in loop `L` that is not preceded by an assignment to `x` in the same block.
- We also save two units if we can avoid a store of `x` at the end of a block.
- Thus, if `x` is allocated a register, we count a savings of two for each block in loop `L` for which `x` is live on exit and in which `x` is assigned a value.

Register Allocation and Assignment

Usage Counts

- On the debit side, if `x` is live on entry to the loop header, we must load `x` into its register just before entering loop `L`. This load costs two units. Similarly, for each exit block `B` of loop `L` at which `x` is live on entry to some successor of `B` outside of `L`, we must store `x` at a cost of two. However, on the assumption that the loop is iterated many times, we may neglect these debits since they occur only once each time we enter the loop.

Register Allocation and Assignment

Usage Counts

- Thus, an approximate formula for the benefit to be realized from allocating a register `x` within loop `L` is

$$\sum_{\text{blocks } B \text{ in } L} use(x, B) + 2 * live(x, B) \qquad \text{Eq (1)}$$

where `use(x, B)` is the number of times `x` is used in `B` prior to any definition of `x`; `live(x, B)` is `1` if `x` is live on exit from `B` and is assigned a value in `B`, and `live(x, B)` is `0` otherwise.

Register Allocation and Assignment

Usage Counts: Example

Consider the basic blocks in the inner loop depicted in below Fig., where jump and conditional jump statements have been omitted. Assume registers R0, R0, and R2 are allocated to hold values throughout the loop. Variables live on entry into and on exit from each block are shown in below Fig., for convenience, immediately above and below each block, respectively. For example, notice that both e and f are live at the end of B1, but of these, only e is live on entry to B2 and only f on entry to B3. In general, the variables live at the end of a block are the union of those live at the beginning of each of its successor blocks.

Register Allocation and Assignment

Usage Counts: Example

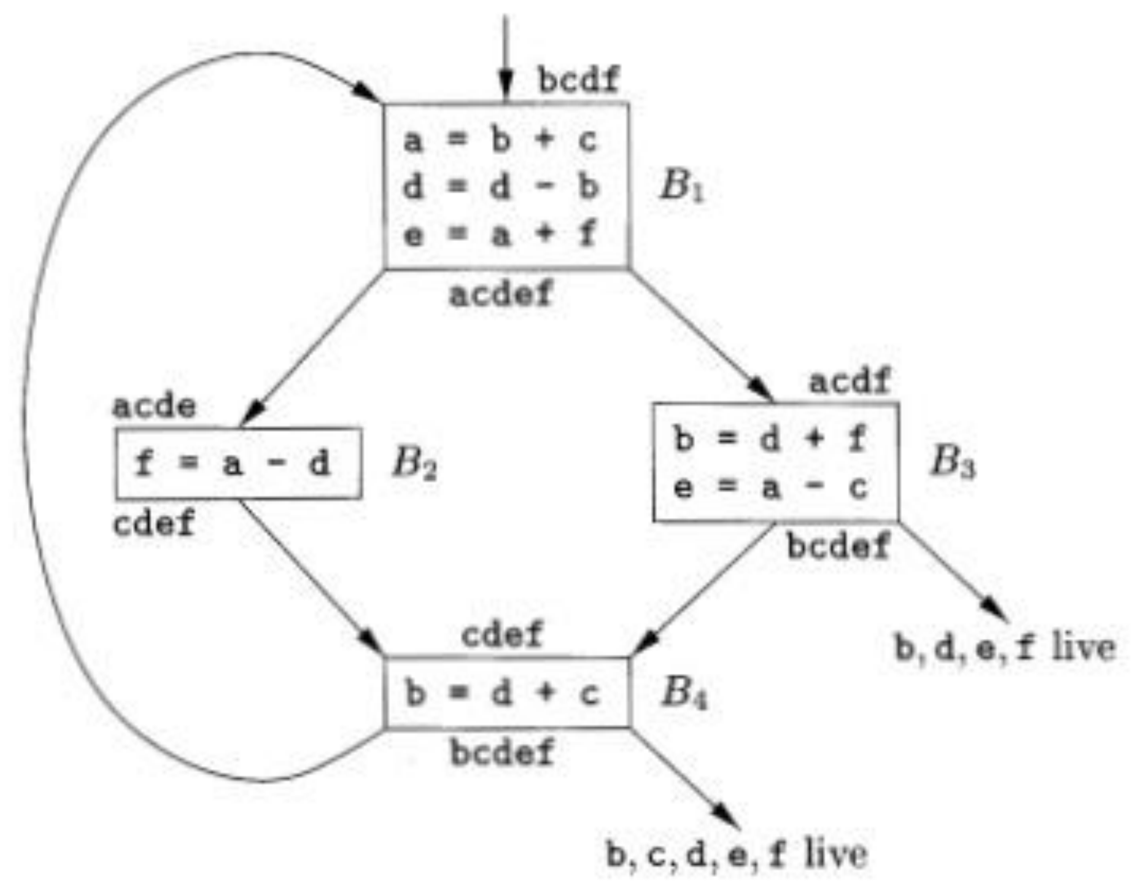

Fig: Flow graph of an inner loop

Register Allocation and Assignment

Usage Counts: Example

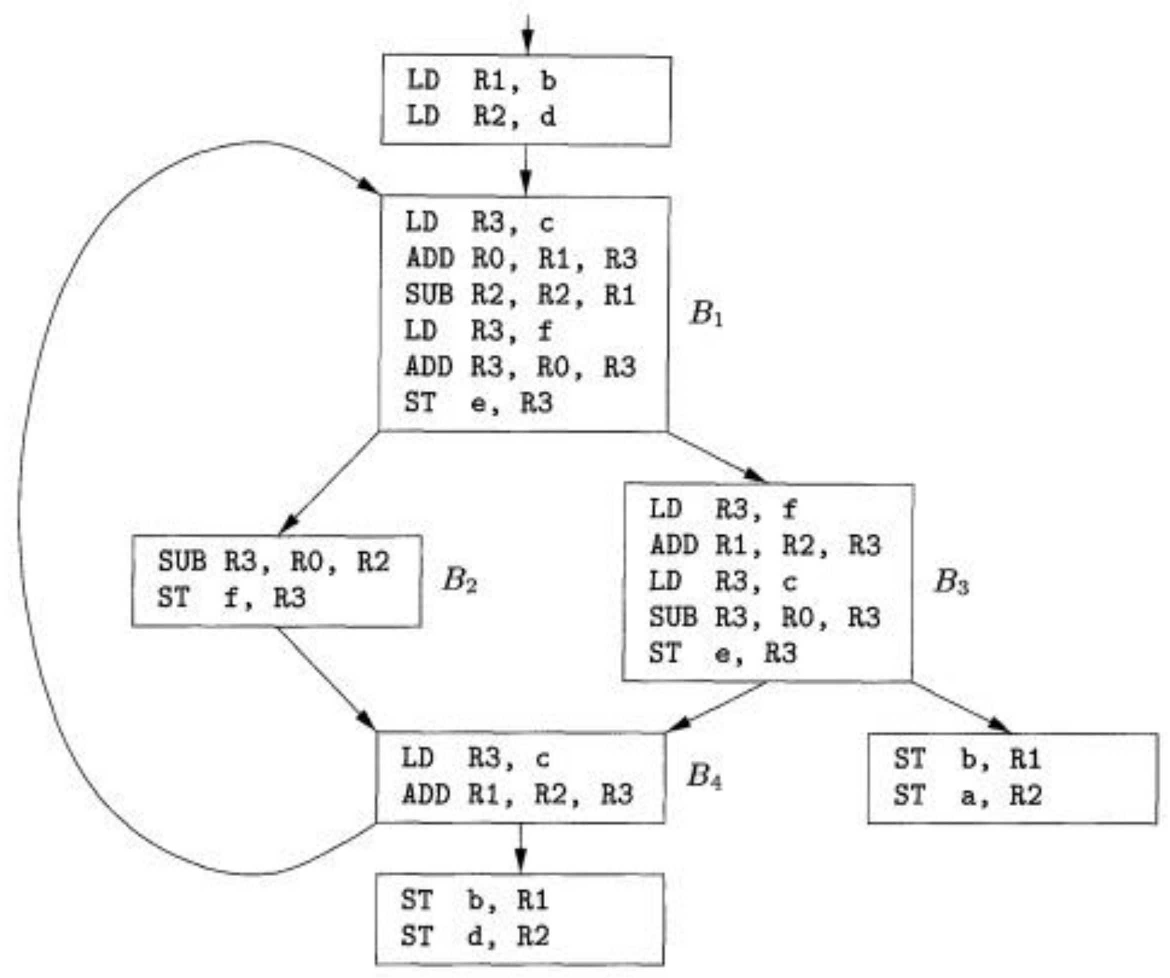

Fig: **Code sequence using global register assignment**

Register Allocation and Assignment

Register Assignment for Outer Loops:

- Having assigned registers and generated code for inner loops, we may apply the same idea to progressively larger Loops.
- If an outer loop L_1 contains an inner loop L_2, the names allocated registers in L_2 need not be allocated registers in $L_1 - L_2$.
- However, if name x is allocated a register in loop L_1 but not L_1, we must load x on entrance to L_2 and store x on exit from L_2.

Register Allocation and Assignment

Register Allocation by Graph Coloring:

- When a register is needed for a computation but all available registers are in use, the contents of one of the used registers must be stored (spilled) into a memory location in order to free up a register.
- Graph coloring is a simple, systematic technique for allocating registers and managing register spills.
- In this method, two passes are used.

First pass:

- In the *first pass*, target machine instructions are selected as though there were an infinite number of symbolic registers; in effect, names used in the intermediate code become names of registers and the three-address instructions become machine-language instructions.

Register Allocation and Assignment

Register Allocation by Graph Coloring:

First pass: cont'd

- If access to variables requires instructions that use stack pointers, display pointers, base registers, or other quantities that assist access, then we assume that these quantities are held in registers reserved for each purpose.
- Once the instructions have been selected, a second pass assigns physical registers to symbolic ones. The goal is to find an assignment that minimizes the cost of the spills.

Register Allocation and Assignment

Register Allocation by Graph Coloring:

Second pass:

- In the *second pass*, for each procedure a register-interference graph is constructed in which the nodes are symbolic registers and an edge connects two nodes if one is live at a point where the other is defined.

Register Allocation and Assignment

Register Allocation by Graph Coloring:

Example:

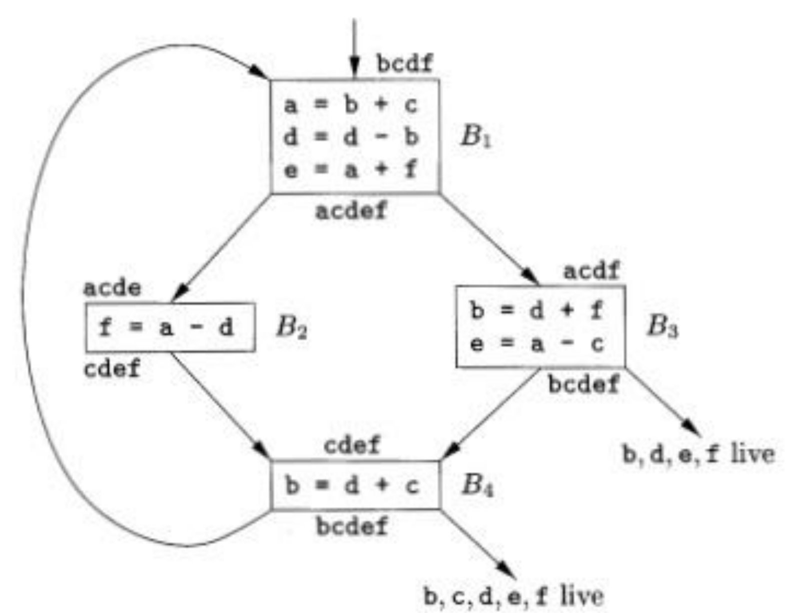

Fig1: **Flow graph of an inner loop**

- For example, a register-interference graph for Fig.1 would have nodes for names a and d. In block B1, a is live at the second statement, which defines d; therefore, in the graph there would be an edge between the nodes for a and d.

Register Allocation and Assignment

Register Allocation by Graph Coloring:

Example:

- An attempt is made to color the register-interference graph using `k` colors, where `k` is the number of assignable registers.
- A graph is said to be colored if each node has been assigned a color in such a way that no two adjacent nodes have the same color.
- A color represents a register, and the coloring makes sure that no two symbolic registers that can interfere with each other are assigned the same physical register.