COMPILATION TECHNIQUES

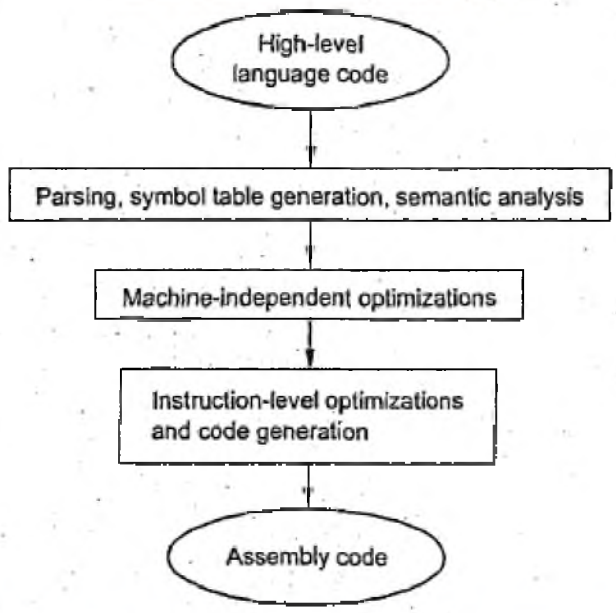
The Compilation Process

Definition:

The compilation is a process of converting the source code into object code. It is done with the help of the compiler and an assembler. The compiler checks the source code for the syntactical or structural errors, and if the source code is error-free, then it generates an assembly code. This assembly code is then converted into object code by using an assembler.

Compilation = -Translation + Optimization

The compilation process is outlined . Compilation begins with high level language code such as C or C++ and generally produces assembly code.



The high-level language program is parsed to break it into statements and expressions. In addition, a symbol table is generated, which includes all the named objects in the program.

Some compilers may then perform higher-level optimizations that can be viewed as modifying the high-level language program input without reference to an instructions.

Basic Compilation Methods

(1) Procedures

Procedures are known as functions in C that requires a specialized code. We generate the code to handle the procedure call and return. At each call of the procedure, we set up the procedure parameters and then makes the call.

The information for a call to a procedure is known as a frame. The frames are stored on a stack to keep track of the order in which the procedures have been called.

Procedure stacks are typically built to grow down from high addresses. A Stack Pointer (SP) defines the end of the current frame, while a Frame Pointer (FP) defines the end of the last frame.

The procedure can refer to an element in the frame by addressing relative to SP. When a new procedure is called, the SP and FP are modified to push another frame onto the stack.

The ARM Procedure Call Standard (APCS) is the recommended procedure linkage for ARM processors. R0- r3 are used to pass the first four parameters into the procedure and r0 is. also used to hold the return value. -

(2) Data Structures

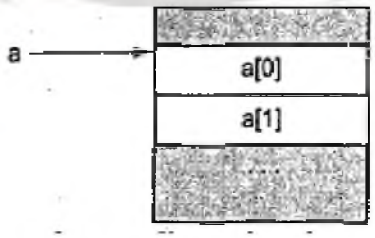
The compiler must also translate references to data structures into references to raw memories.

Address of an array element must be computed at run time, because the array index may change.

Let us consider a one dimensional array:

a[i]

The layout of the array in memory is shown in Fig 6.19. The zeroth element is stored as the first element of the array, the first element directly below, and so on.



For our convenience, we can use the pointer aptr for the reading of a[i] as.

(aptr + i)

If the a[ ] array with the size of M x N, then the two-dimensional arm access can be expressed as,

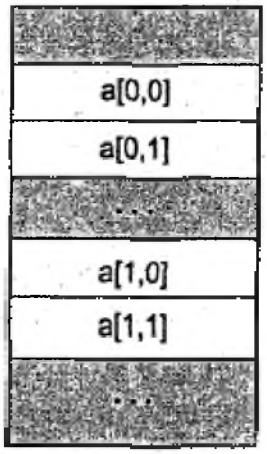
a[i> j]

Two-dimensional array structure which is implemented as

a contiguous block of memory. Fields in this structure can be accessed using constant offsets to the base address of the structure.

For example,, if fieldl is 4 bytes long, then field2 can be accessed as

(aptr + 4)



Compiler Optimizations

The basic compilation techniques can generate inefficient code. Then compilers use a wide range of algorithms to optimize the code which it generates.

(1)Loop Transformations:

Loop Optimization Techniques .

Loop optimization is the process of increasing an execution speed and reducing the overheads associated with loops. It plays an important role in improving cache performance and making effective use of parallel processing capabilities.

Most execution time of a scientific program is spent on loops.

(i) Loop Unrolling:

Loop unrolling is a loop transformation technique that helps to optimize the execution time of a program. It increases the program's speed by eliminating loop control instruction and loop test instructions.

(ii) Loop Fusion (or) Loop Jamming

Loop fusion is combining two or more loops in a single loop which reduces the loop overhead and also reduces the time taken to compile the many loops i.e., improve the run-time performance.

```
Before optimization:
for(int i=0; i<5; i++)
        a = i + 5;
for(int i=0; i<5; i++)
        b = i + 10;
After optimization:
for(int i=0; i<5; i++)
{
    a = i + 5;
    b = i + 10;
}
```

(iii) Loop Distribution

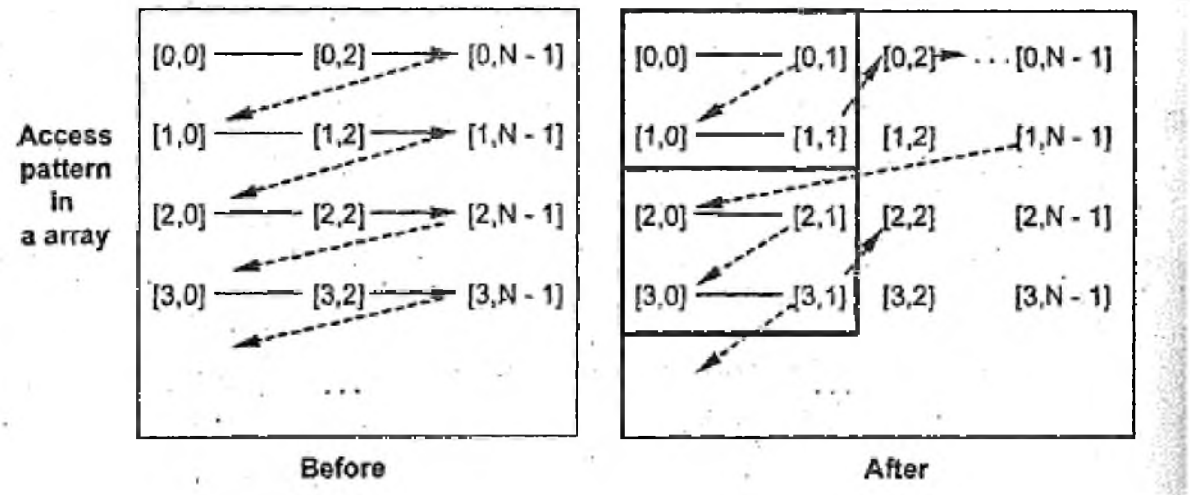
Loop distribution (or) Loop fission is a compiler optimization technique in which a loop is broken into multiple loops over the same index range with each taking only a part of the original loop's body.

This optimization is most efficient in multi-core processors that can split a task into multiple tasks for each processor.

(d) Loop Tiling

Loop tiling breaks up a loop into a set of nested loops and each inner loop performing the operations on a subset of the data.

An example is here, each loop is broken up into tiles of size two that is, each loop is split into two loops. For example, the inner 'ii' loop iterates within the tile and the outer ' i' loop iterates across the tiles.



(2) Dead Code Elimination

Dead Code:

Dead code is the code that can be never executed. It can be generated by the programmers, either inadvertently or purposefully and also by the compilers.

Dead code can be identified by reach ability analysis that is, finding the other statements or instructions from which it can be reached. Dead code elimination analyzes the code for reach ability and removes it.

(3) Register Allocation

Register allocation is an important method in the final phase of the compiler. Registers are faster to access than cache memory and registers are available in small size up to few hundred kB .Thus, it is necessary to use a minimum number of registers for variable allocation.

(4) Scheduling:

Instruction Scheduling:

Instruction scheduling is a process of mapping a series of instructions into execution of resources. It decides when and where an instruction is executed.

We can keep track of CPU resources during instruction scheduling by using a reservation table which illustrated in Rows in the table represent instruction execution time slots and columns represent resources that must be scheduled.

(5)Instruction Selection

Selecting the instructions, to implement each operation is a difficult task. There may be a several different instructions that can be used to accompli.^ the same goal, but they may have different execution times.

Using one instruction for one part of the program may affect the instructions that can be used in adjacent code. One useful technique for generating code is template matching.