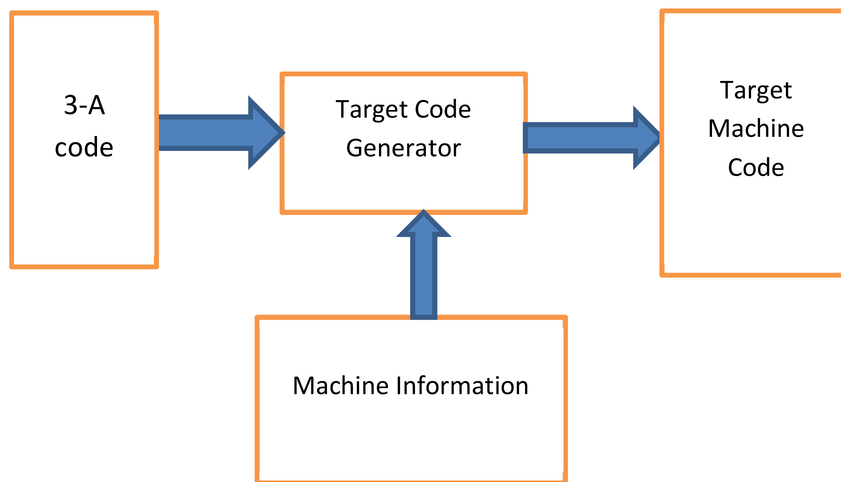
TARGET CODE GENERATION -Date12/11/13

By: Sourav Sarkar 11CS30037

Summary: In this lecture we have studied about how to create machine code from a given 3-Address (3-A) code.

As m/c code is machine dependent, to convert 3-A code to machine code we need to know following about the specific target machine-

1. It's Instruction Set. I.e. format of different basic instructions like
2. Machine's Register Set. Its takes a very crucial role in code optimization.
3. Type of the machine i.e. CISC or RISC.
4. Address modes the machine supports.



Following are the basic most information we must know about the machine to generate the target code,

1> **Load-Store instructions:** Load dst,src [eg: Load R0,x] , Store dst,src [eg: Store x,R0] etc.

2> **Arithmetic and Logical operations:** Like ADD R0,R1,R2; ADD R0,R1,m; etc.

3> **Control instructions:** Unconditional Branch (BRN L), Conditional Branch (BGTZ x L) etc.

4> Addressing Modes:

a. Register
b. Indirect.
c. Index
d. Immediate, etc

TARGET CODE GENERATION -Date12/11/13

Challenges of Code Generation:

1> Instruction Selection:

Each and every instruction must be mapped to some machine instructions maintaining proper consistency.

Eg:

Suppose we have the 3-A instruction: "x=y+z"

One solution is as follows:

Load R1,y
ADD R1,R2,z
Store x,R1.

But suppose we have the following piece of code:

"a=b+c
f=x+a"

Using previous method:

Load R1,b
Add R1,R1,c
Store R1,a
Load R1,a
Add R1,R1,x
Store f,R1

But this Store Load combination is redundant and also it slows program execution as its accessing memory which take much more time than CPU. Instead we could directly use R1

So it seems that concerning only single instructions is highly non-optimized. So the Challenge is to make as efficient instruction selection as possible.

2>Allocation of Registers:

a) Register allocation: The task is to get a set a registers to hold a set of variables.
b) Register Assignment: After that we need to assign which register will hold which variable.

Measure of Optimality :

So to examine which strategy is best for code optimization, we first need criteria for optimization. Here we measure optimality as follows:

a) Number of memory access.
b) Length of the instruction.

Rule for cost assignment:

TARGET CODE GENERATION -Date12/11/13

a> Add +1 for each instruction.
b> Add +1 for each instruction involving memory access.
So for this scheme ADD R0,R1,R2 will have cost 1

But ADD R0,R1,m will have cost 2 !

Using this measure we can say the less a code access memory and the less instruction it takes the more is it optimized i.e. the code with the minimum cost is the optimized one.

So to optimize the machine we need to analyze the 3-A code first.

Block Based Analysis:

Split the intermediate code in basic blocks. Sequence of instructions in a block execute together. We define a “**Block**” as

Set of consecutive statements such that

A) Control will enter the block by only executing its first statement.
B) There should not be any jump from inside the block.
C) Control will exit only by executing the last instruction.

So To identify the Blocks we define the **Leader** of the block as the very first instruction of the block. If for a given 3-A code we identify all the Leaders then equivalently all the Blocks are also identified.

Rule to identify the Leaders:

A) The first instruction of the 3-A code is a Leader.
B) All the Target statements i.e. Labels are Leaders too.
C) Statements that follows goto.

So all the instructions following a Leader (including it) before the very next leader is a Block.

We illustrate the Strategy by an Example:

High Level Code:

```
begin:
loc =-1
i=0
while(I<100)
do
   begin :
   if(A[i] == x) then loc = I
   i=i+1
   end
```

TARGET CODE GENERATION -Date12/11/13

end

……………………

………………………

<u>Corresponding 3-A code:</u>

1. loc = -1
2. i=0
3. if i<100 goto L2
4. goto L3
5. L1: t1=2*i
6. t2=A[t1]
7. If t2=x goto L3
8. goto L4
9. L3: loc =i
10. L4: t3=i+1
11. i=t3
12. goto L1
13. L5

Using the algorithm the get the Leaders:

1,3,5,9,10,4,8,13

And so we also have the blocks and now we can represent the the code as a **Flow Graph** by the following rules:

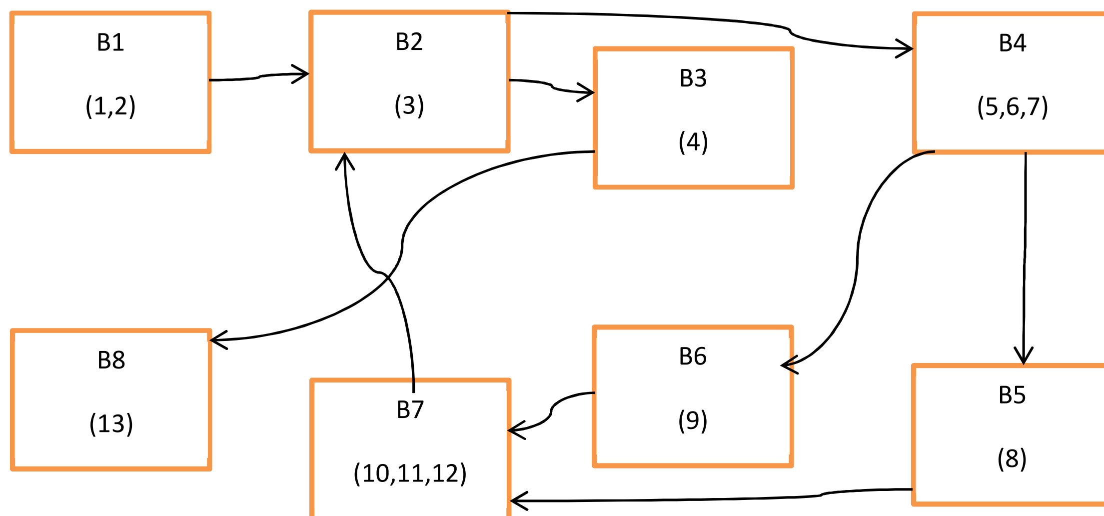
1> Nodes are the Blocks

2> Now there is a flow from Bi to Bj if either

a) The last statement of Bi is a conditional or unconditional jump to Bj

b) If the last statement of Bi is not an unconditional jump to some other block and j=i+1

So the corresponding Flow Graph for the above 3-A code is as follows –



Flow Graph and the Block Division of the 3-A Code