DESIGN OF A SIMPLE CODE GENERATOR

A code generator generates target code for a sequence of three- address statements and effectively uses registers to store operands of the statements. One of the primary issues during code generation is deciding how to use registers to best advantage. There are four principal uses of registers:

1. In most machine architectures, some or all of the operands of an operation must be in registers in order to perform the operation.
2. Registers make good temporaries - places to hold the result of a sub expression while a larger expression is being evaluated, or more generally, a place to hold a variable that is used only within a single basic block.
3. Registers are used to hold (global values that are computed in one basic block and used in other blocks, for example, a loop index that is incremented going around the loop and is used several times within the loop.
4. Registers are often used to help with run-time storage management, for example, to manage the run-time stack, including the maintenance of stack pointers and possibly the top elements of the stack itself.

Register and Address Descriptors

- The code-generation algorithm considers each three-address instruction and decides what loads are necessary to get the needed operands into registers. After generating the loads, it generates the operation itself. Then, if there is a need to store the result into a memory location, it also generates that store.
- A data structure is needed to keep track of the program variables currently having their value in a register, and which register or registers. Also need to know whether the memory location for a given variable currently has the proper value for that variable, since a new value for the variable may have been computed in a register and not yet stored.

The desired data structure has the following descriptors:

1. Register Descriptor: For each available register, a register descriptor keeps track of the variable names whose current value is in that register. Since only those registers that are available for local use is available, assume that initially, all register descriptors are empty. As the code generation progresses, each register will hold the value of zero or more names.
2. Address Descriptor: For each program variable, an address descriptor keeps track of the location or locations where the current value of that variable can be found. The location might be a register, a memory address, a stack location. The information can be stored in the symbol table entry for that variable name.

The Code-Generation Algorithm

- The code-generation algorithm contains a function getReg(I), which selects registers for each memory location associated with the three-address instruction I.

- Function getReg has access to the register and address descriptors for all the variables of the basic block, and to certain useful data-flow information such as the variables that are live on exit from the block.
- Assume that there are enough registers to accomplish any three-address operation.
- In a three-address instruction such as x = y + z, + is treated as a generic operator and ADD as the equivalent machine instruction.

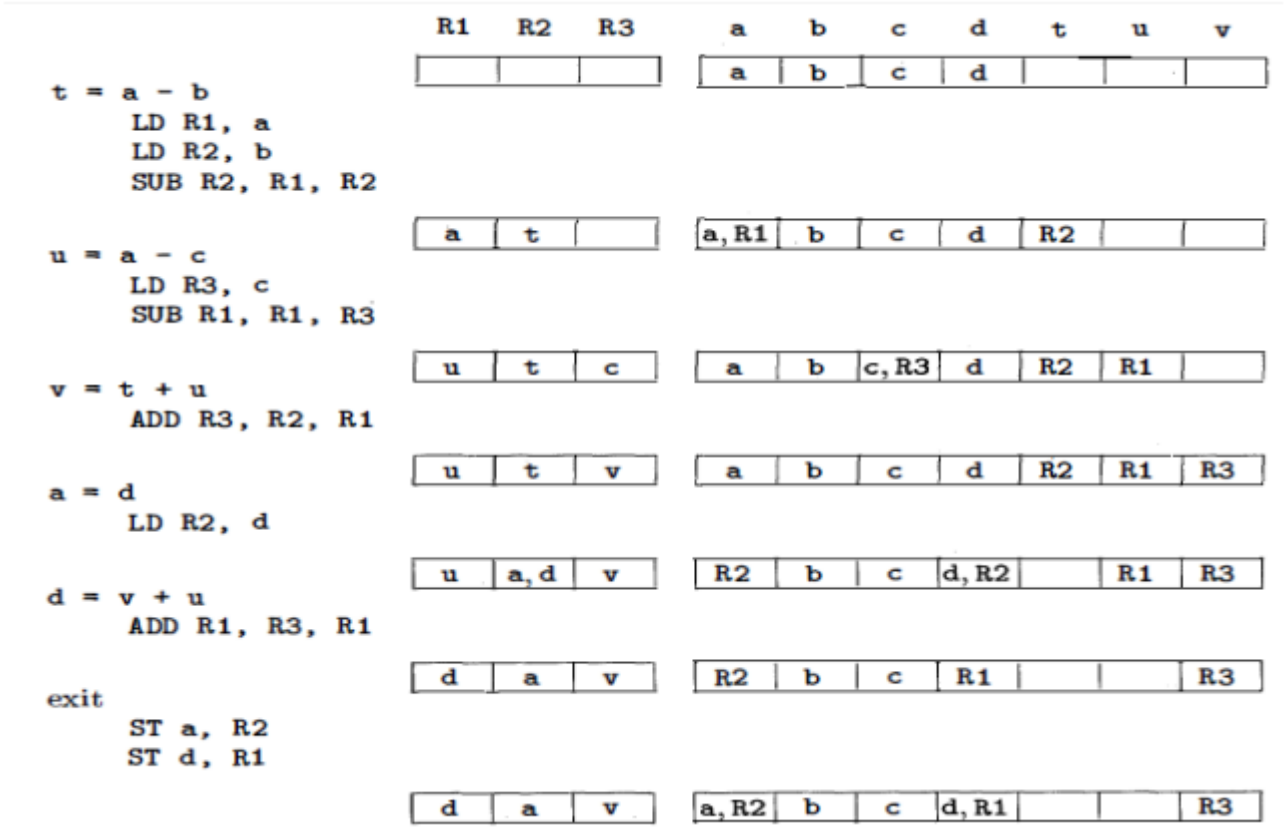
Ending the Basic Block:

- Variables used by the block may wind up with their only location being a register.
- If the variable is a temporary used only within the block, then when the block ends, the value of the temporary is forgot and assume its register is empty.
- If the variable is live on exit from the block, or if we don't know which variables are live on exit, then assume that the value of the variable is needed later. In that case, if the value for variable x is not in memory location of x, then generate the instruction ST x, R, where R is a register holding the x's value.

Example: Translate the basic block consisting of the three-address statements

```
t = a - b
u = a - c
v = t + u
a = d
d = v + u
```

- Assume that t , u, and v are temporaries, local to the block, while a, b, c , and d are variables that are live on exit from the block.
- Assume that there are as many registers as needed and when a register's value is no longer needed then reuse its register.
- It shows the register and address descriptors before and after the translation of each three-address instruction.



Three Address Code	Assembly Code	Register Descriptor	Address Descriptor	
Initial Setup of Register and Address Descriptor		R1 = EMPTY R2 = EMPTY R3 = EMPTY	a= a b= b c= c d= d	t= u= v=

t=a-b	LD R1, a
	LD R2, b
	SUB R2, R1, R2
		R1 = a
		R2 = t
		R3 = EMPTY
			a= a
			b= b
			c= c
			d= d
				t=R2
				u=
				v=

u = a-c	LD R3, c SUB R1, R1, R3	R1 = u R2 = t R3 = c	a= a b= b c= c, R3 d= d	t=R2 u=R1 v=
v = t + u	ADD R3, R2, R1	R1 = u R2 = t R3 = v	a= a b= b c= c d= d	t=R2 u=R1 v=R3
a = d	LD R2, d	R1 = u R2 = a,d R3 = v	a= R2 b= b c= c d= d, R2	t= u= R1 v= R3
d = v+u	ADD R1, R3, R1	R1 = d R2 = a R3 = v	a= R2 b= b c= c d= R1	t= u= v= R3
Exit	ST a, R2 ST d, R1	R1 = d R2 = a R3 = v	a= a,R2 b= b c= c d= d, R1	t= u= v= R3

- For the first three-address instruction, t = a - b we need to issue three instructions, since nothing is in a register initially. Thus, a and b are loaded into registers R1 and R2, and the value t produced in register R2. Can use R2 for t because the value b previously in R2 is not needed within the block.
- The second instruction, u = a - c, does not require a load of a, since it is already in register R1. Can reuse R1 for the result, u, since the value of a, previously in that register, is no longer needed within the block, and its value is in its own memory location. Change the address descriptor for a to indicate that it is no longer in R1, but is in the memory location called a.
- The third instruction, v = t + u, requires only the addition. Can use R3 for the result, v, since the value of c in that register is no longer needed within the block, and c has its value in its own memory location.
- The copy instruction, a = d, requires a load of d, since it is not in memory. Register R2's descriptor holding both a and d. The addition of a to the register descriptor is the result of processing the copy statement.
- The fifth instruction, d = v + u, uses two values that are in registers. Since u is a temporary whose value is no longer needed, reuse its register R1 for the new value of d. Variable d and a is now in only R1, and is not in its own memory location. The machine code for the basic block that stores the live-on-exit variables a and d into their memory locations is needed.