into registers R1 and R2 and the value t produced in register R2 Notice that we can use R2 for t because the value b previously in R2 is not needed within the block Since b is presumably live on exit from the block had it not been in its own memory location as indicated by its address descriptor we would have had to store R2 into b rst The decision to do so had we needed R2 would be taken by *getReg*

The second instruction u a c does not require a load of a since it is already in register R1 Further we 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 if a is needed outside the block Note that we 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 Further we 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 a register We show register R2 s descriptor holding both a and d The addition of a to the register descriptor is the result of our processing the copy statement and is not the result of any machine instruction

The fth instruction d v u uses two values that are in registers Since u is a temporary whose value is no longer needed we have chosen to reuse its register R1 for the new value of d Notice that d is now in only R1 and is not in its own memory location The same holds for a which is in R2 and not in the memory location called a As a result we need a coda to the machine code for the basic block that stores the live on exit variables a and d into their memory locations We show these as the last two instructions □

8 6 3 Design of the Function *getReg*

Lastly let us consider how to implement *getReg* I for a three address in struction I There are many options although there are also some absolute prohibitions against choices that lead to incorrect code due to the loss of the value of one or more live variables We begin our examination with the case of an operation step for which we again use x y z as the generic example First we must pick a register for y and a register for z The issues are the same so we shall concentrate on picking register R_y for y The rules are as follows

1 If y is currently in a register pick a register already containing y as R_y Do not issue a machine instruction to load this register as none is needed

2 If y is not in a register but there is a register that is currently empty pick one such register as R_y

3 The di cult case occurs when y is not in a register and there is no register that is currently empty We need to pick one of the allowable registers anyway and we need to make it safe to reuse Let R be a candidate

register and suppose v is one of the variables that the register descriptor for R says is in R We need to make sure that v s value either is not really needed or that there is somewhere else we can go to get the value of v The possibilities are

a If the address descriptor for v says that v is somewhere besides R then we are OK

b If v is x the variable being computed by instruction I and x is not also one of the other operands of instruction I z in this example then we are OK The reason is that in this case we know this value of x is never again going to be used so we are free to ignore it

c Otherwise if v is not used later that is after the instruction I there are no further uses of v and if v is live on exit from the block then v is recomputed within the block then we are OK

d If we are not OK by one of the rst three cases then we need to generate the store instruction `ST` v R to place a copy of v in its own memory location This operation is called a *spill*

Since R may hold several variables at the moment we repeat the above steps for each such variable v At the end R s score is the number of store instructions we needed to generate Pick one of the registers with the lowest score

Now consider the selection of the register R_x The issues and options are almost as for y so we shall only mention the di erences

1 Since a new value of x is being computed a register that holds only x is always an acceptable choice for R_x This statement holds even if x is one of y and z since our machine instructions allows two registers to be the same in one instruction

2 If y is not used after instruction I in the sense described for variable v in item 3c and R_y holds only y after being loaded if necessary then R_y can also be used as R_x A similar option holds regarding z and R_z

The last matter to consider specially is the case when I is a copy instruction x y We pick the register R_y as above Then we always choose R_x R_y

8 6 4 Exercises for Section 8 6

Exercise 8 6 1 For each of the following C assignment statements

a `x a b c`

b `x a b c d e f`

c `x a i 1`