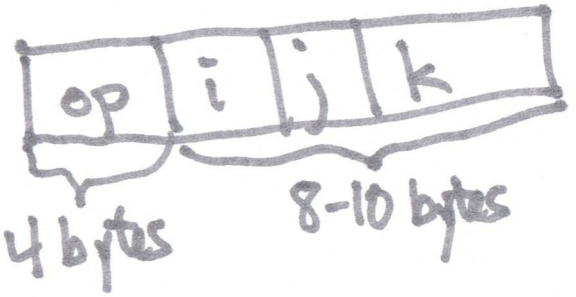
THREE-ADDRESS CODE (3AC)

$i = j \text{ op } k$ (most operations)



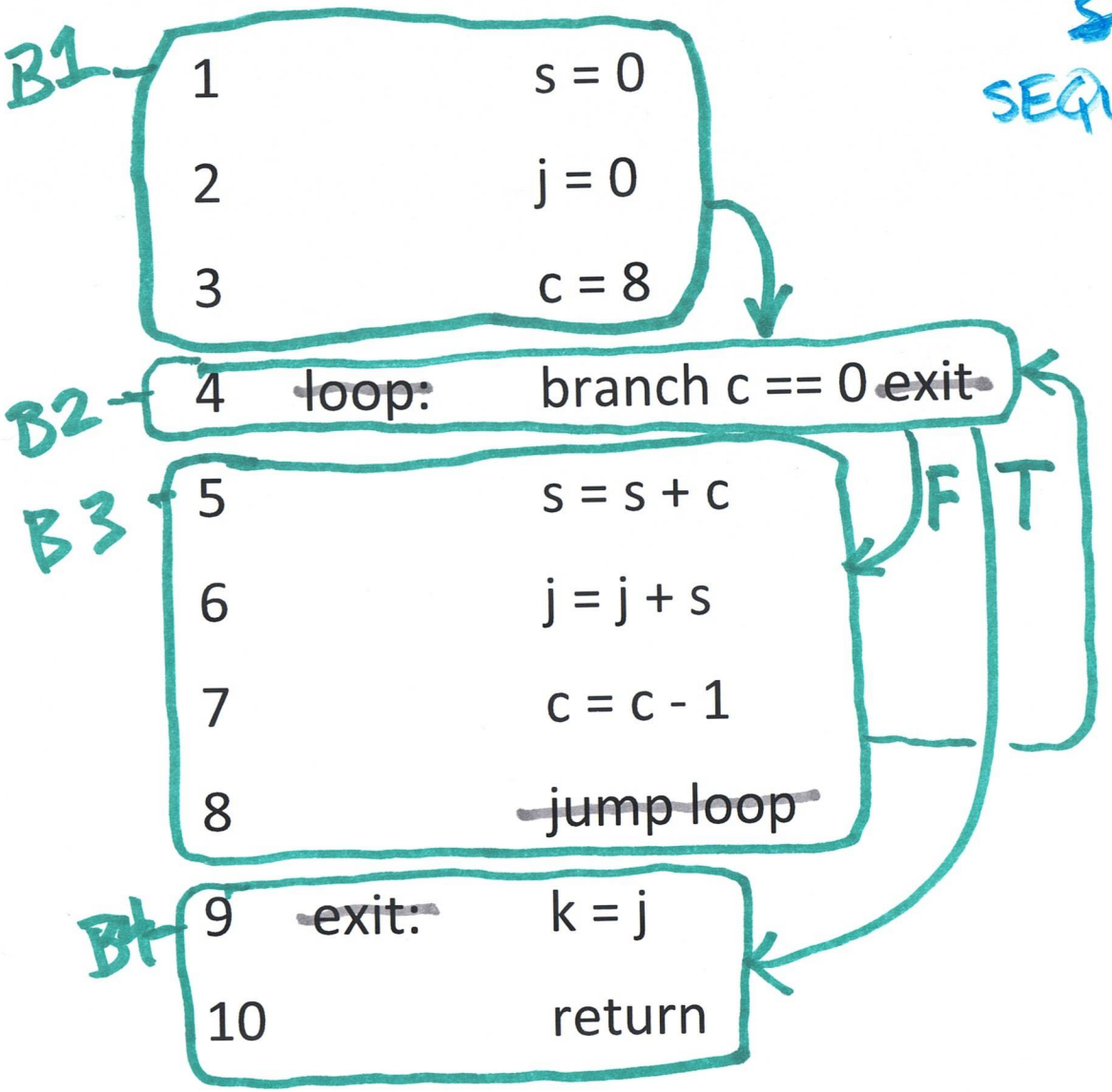
≈14 bytes/instruction.

QUADRUPLE

common
- array
- list of arrays
- list of pointers to quadruples.
- linked lists

BASIC BLOCK:

~~SET~~ OF INSTRUCTIONS SEQUENCE THAT MUST BE EXECUTED TOGETHER



leader: start of subroutine or labelled instruction, or statement after branch.

CONTROL FLOW GRAPH (CFG)

A[i][j] = B[i][j] + C[i][j]

r_1 = load B
r_2 = i – 1
r_3 = r_2 * 4
r_4 = r_1 + r_3
r_5 = load r_4
r_6 = j – 1
r_7 = r_6 * 4
r_8 = r_5 + r_7
r_9 = load r_8
r_{10} = load C
r_{11} = i – 1
r_{12} = r_{11} * 4
r_{13} = r_{10} + r_{12}
r_{14} = load r_{13}
r_{15} = j – 1
r_{16} = r_{15} * 4
r_{17} = r_{14} + r_{16}
r_{18} = load r_{17}
r_{19} = r_9 + r_{18}
r_{20} = load A
r_{21} = i – 1
r_{22} = r_{21} * 4
r_{23} = r_{20} + r_{22}
r_{24} = load r_{23}
r_{25} = j – 1
r_{26} = r_{25} * 4
r_{27} = r_{24} + r_{26}
store r_{27}, r_{19}

LONG BASIC BLOCKS ARE GOOD.

PLENTY OF OPTIMIZATION OPPORTUNITIES

OPTIMIZATIONS

- Optimizations done on a single basic block are called **LOCAL** optimizations.
- Optimizations done on a single procedure are called **GLOBAL** optimizations.
- Optimizations on a larger scale are called **INTERPROCEDURAL** optimizations.

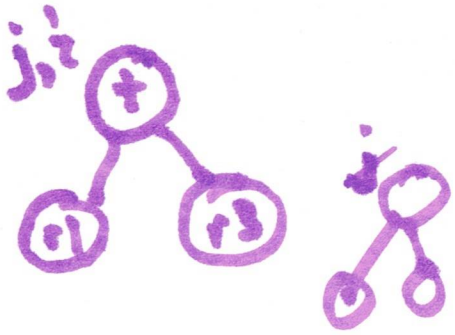
→ Optimizations done over more than one file are sometimes called **INTERMODULAR**.

Expression DAGs For Local Optimization

For each instruction:

1. create a node for ~~instruction~~ argument if it does not already exist.
2. create a node for operation if a node with same arguments and operation does not already exist connect this node to arguments
3. If a new node was created in (2), then call it by the result name. If not, add the result name to existing node. If another node with same name exists, deprecate the name there.

```
i = r1 + r3
j = r1 + r3
```



$r_1 = \text{load } B$

$r_2 = i - 1$

$r_2 = r_2 * 4$

$r_4 = r_1 + r_2$

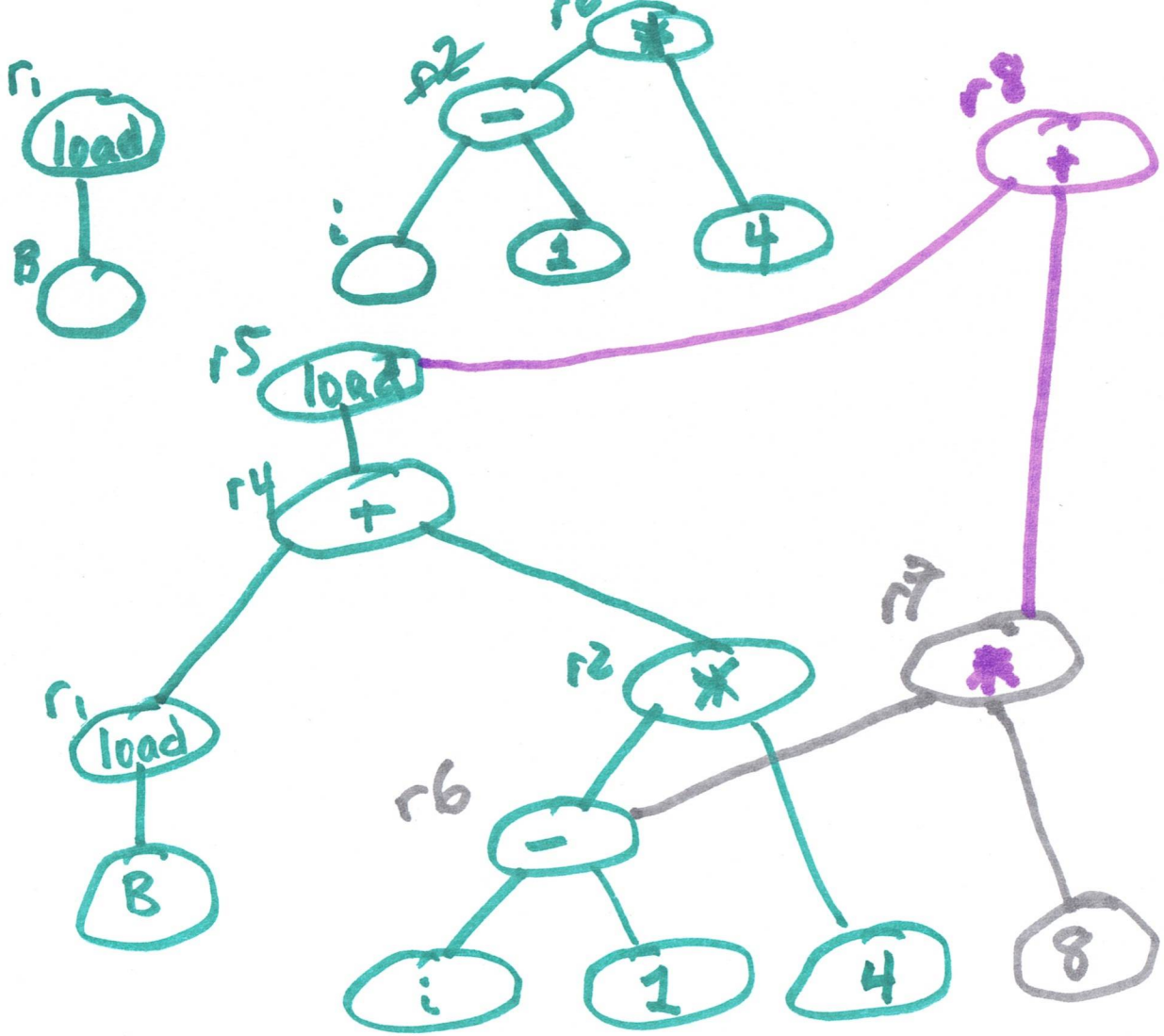
$r_5 = \text{load } r_4$

$r_6 = i - 1$

$r_7 = r_6 * 8$

$r_8 = r_5 + r_7$

8 instructions



$r_1 = \text{load } B$

$r_6 = i - 1$

$r_2 = r_6 * 4$

$r_4 = r_1 + r_2$

$r_5 = \text{load } r_4$

$r_7 = r_6 * 8$

$r_8 = r_5 + r_7$

7 instructions

STATIC SINGLE ASSIGNMENT FORM (SSA)

```
x=0
y=0
while (x<100)
    x=x+1
    y=y+x
```

SSA

```
x0=0
y0=0
loop: x1 = φ(x0,x2)      } concurrent execution.
      y1 = φ(y0,y2)
      if x1 ≥ 100 goto next
      x2=x1+1
      y2=y1+x2
      goto loop
next:
```

φ( ) is not just binary- it's n-ary.

→ φ-functions don't fit naturally into 3AC