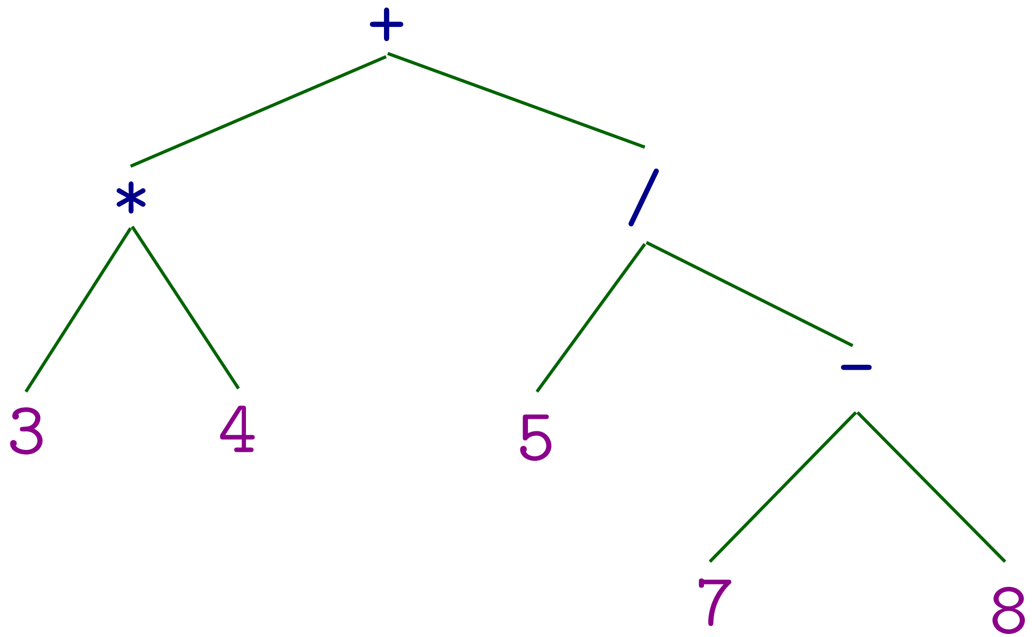
Compilation of Expressions

CS4212 – Lecture 3 (live)

Outline

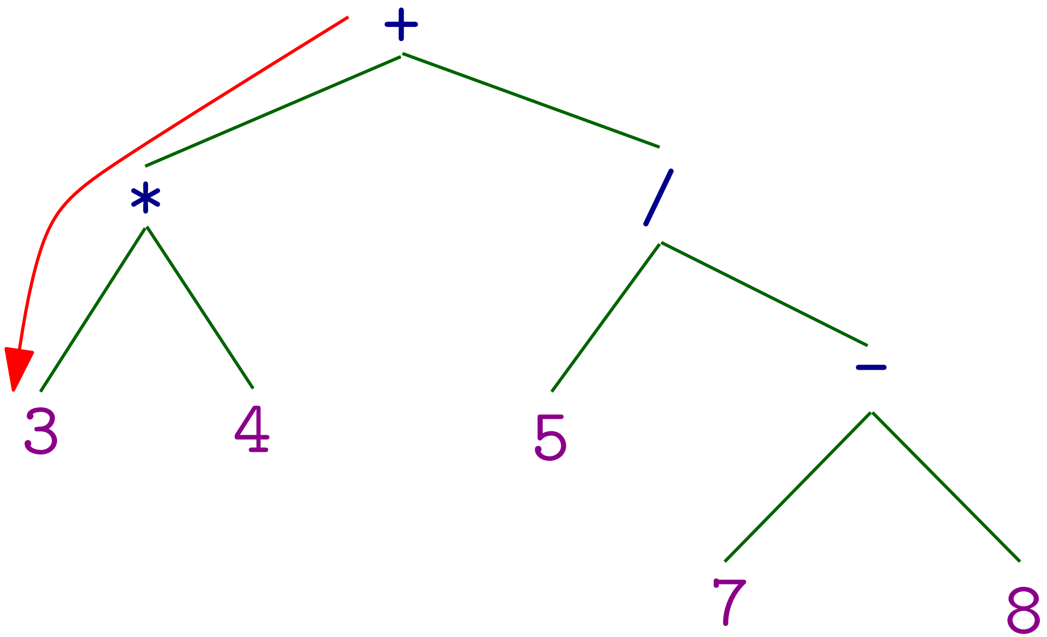
- Version 1: simple binary expressions
- Version 2: add conditional operators
- Version 3: better engineering
- Version 4: adding assignments

Compilation by Post-Order Traversal



```
pushl $3
pushl $4
popl %ebx
popl %eax
imull %ebx,%eax
pushl %eax
pushl $5
pushl $7
pushl $8
popl %ebx
popl %eax
sub %ebx,%eax
pushl %eax
popl %ebx
popl %eax
cdq
idivl %ebx
pushl %eax
popl %ebx
popl %eax
addl %ebx,%eax
pushl %eax
```

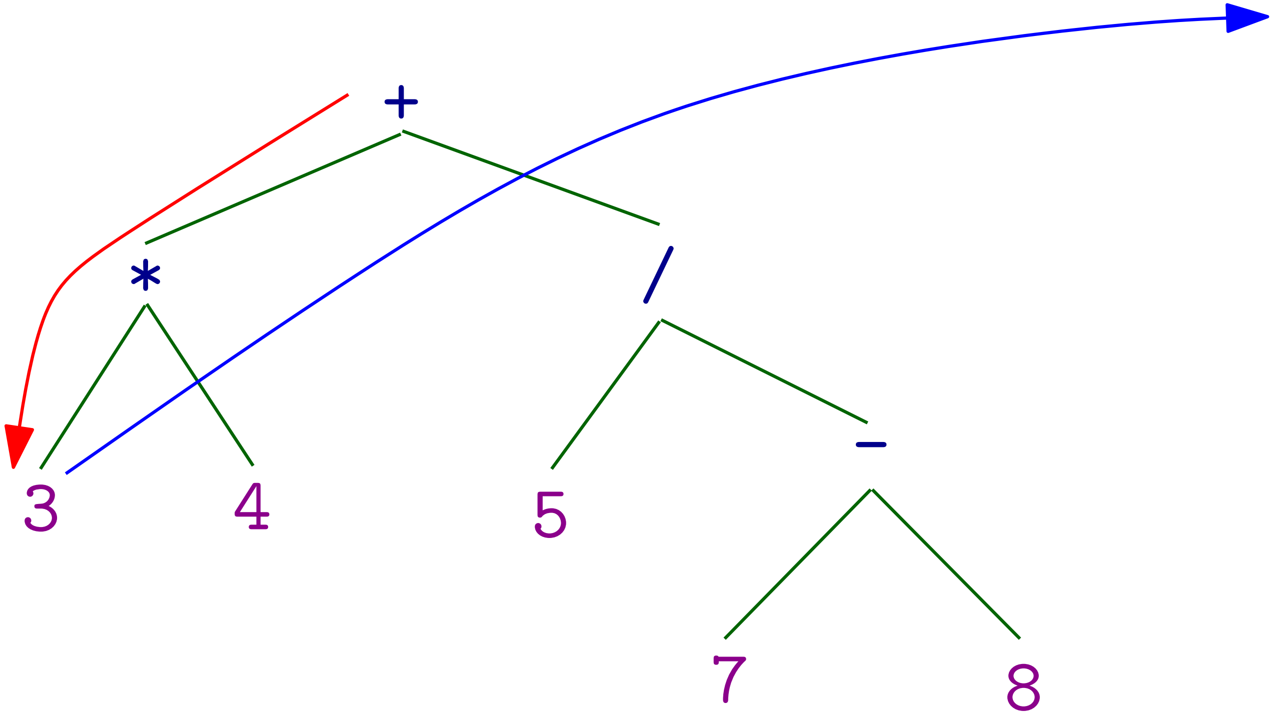
Compilation by Post-Order Traversal



```
pushl $3
pushl $4
popl %ebx
popl %eax
imull %ebx,%eax
pushl %eax
pushl $5
pushl $7
pushl $8
popl %ebx
popl %eax
sub %ebx,%eax
pushl %eax
popl %ebx
popl %eax
cdq
idivl %ebx
pushl %eax
popl %ebx
popl %eax
addl %ebx,%eax
pushl %eax
```

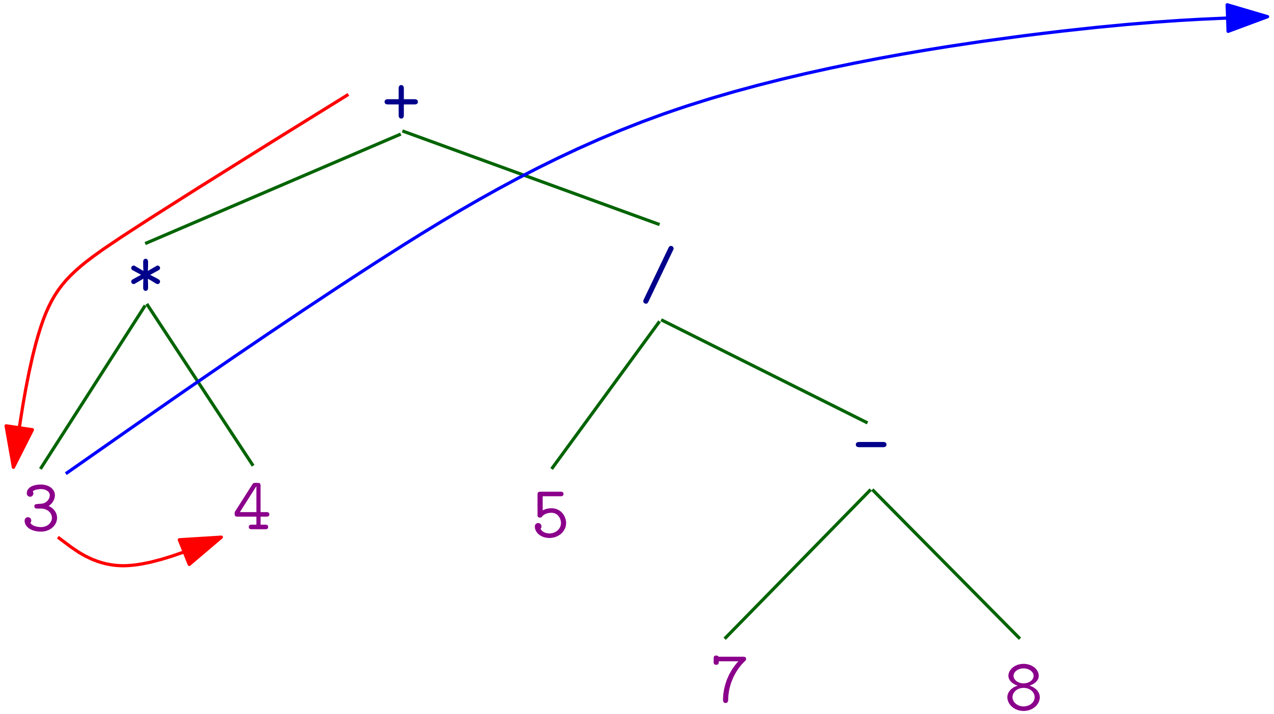
Compilation by Post-Order Traversal

[C]



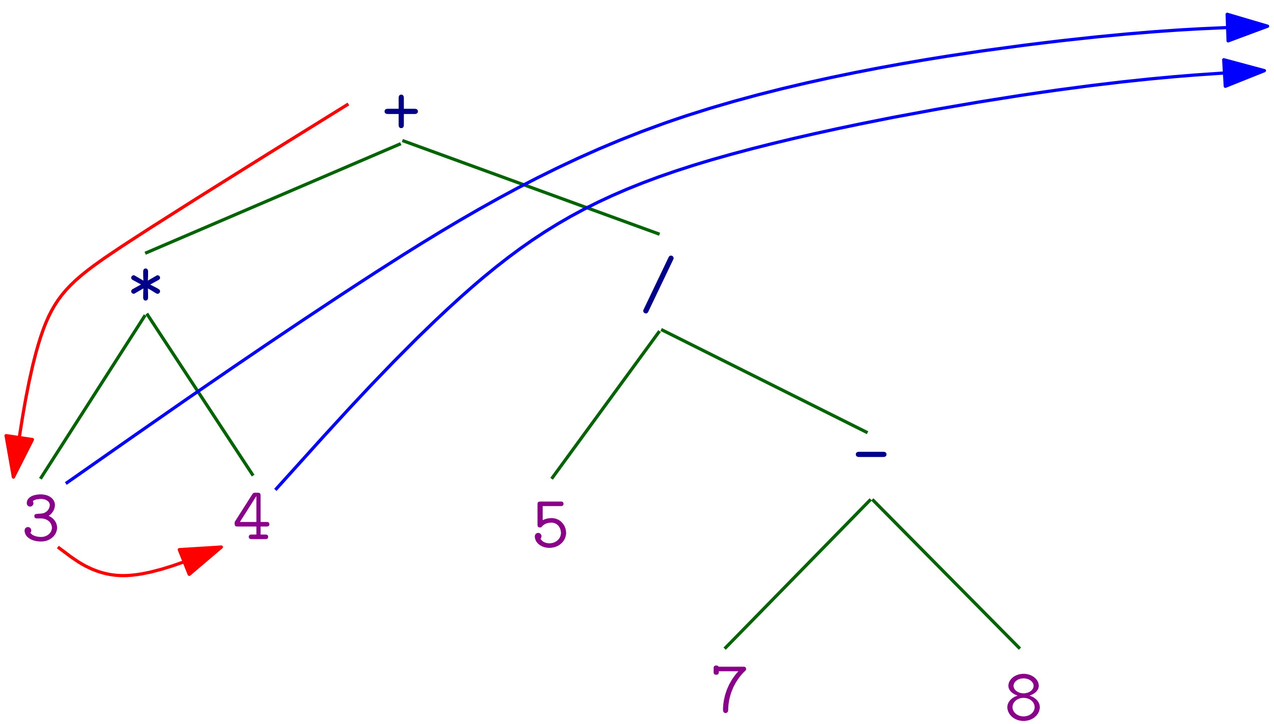
```
pushl $3
pushl $4
popl %ebx
popl %eax
imull %ebx,%eax
pushl %eax
pushl $5
pushl $7
pushl $8
popl %ebx
popl %eax
sub %ebx,%eax
pushl %eax
popl %ebx
popl %eax
cdq
idivl %ebx
pushl %eax
popl %ebx
popl %eax
addl %ebx,%eax
pushl %eax
```

Compilation by Post-Order Traversal



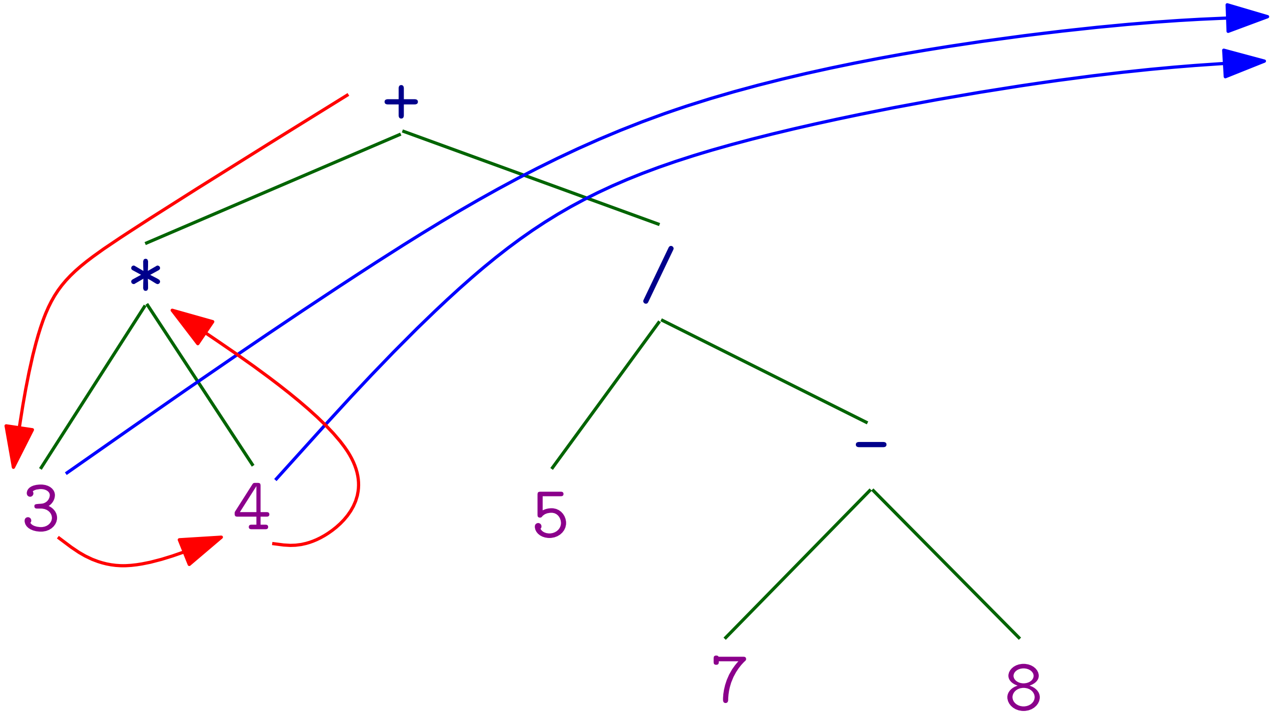
```
pushl $3
pushl $4
popl %ebx
popl %eax
imull %ebx,%eax
pushl %eax
pushl $5
pushl $7
pushl $8
popl %ebx
popl %eax
sub %ebx,%eax
pushl %eax
popl %ebx
popl %eax
cdq
idivl %ebx
pushl %eax
popl %ebx
popl %eax
addl %ebx,%eax
pushl %eax
```

Compilation by Post-Order Traversal



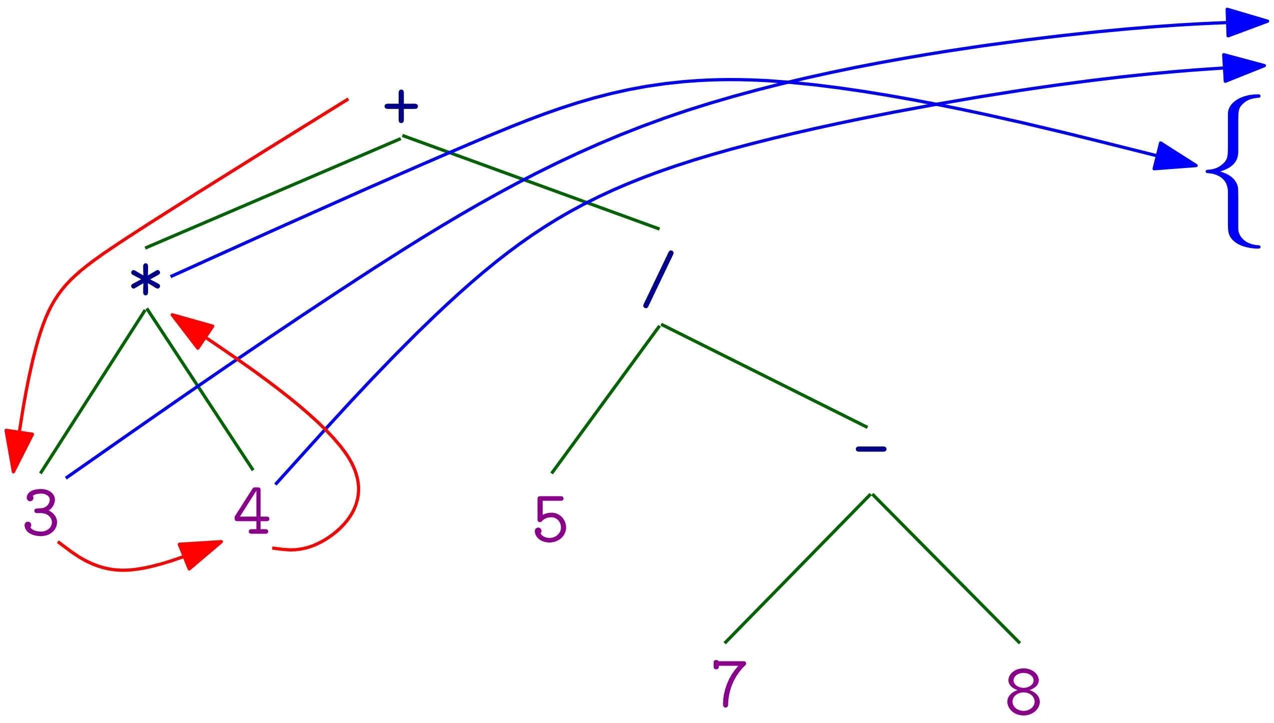
```
pushl $3
pushl $4
popl %ebx
popl %eax
imull %ebx,%eax
pushl %eax
pushl $5
pushl $7
pushl $8
popl %ebx
popl %eax
sub %ebx,%eax
pushl %eax
popl %ebx
popl %eax
cdq
idivl %ebx
pushl %eax
popl %ebx
popl %eax
addl %ebx,%eax
pushl %eax
```

Compilation by Post-Order Traversal



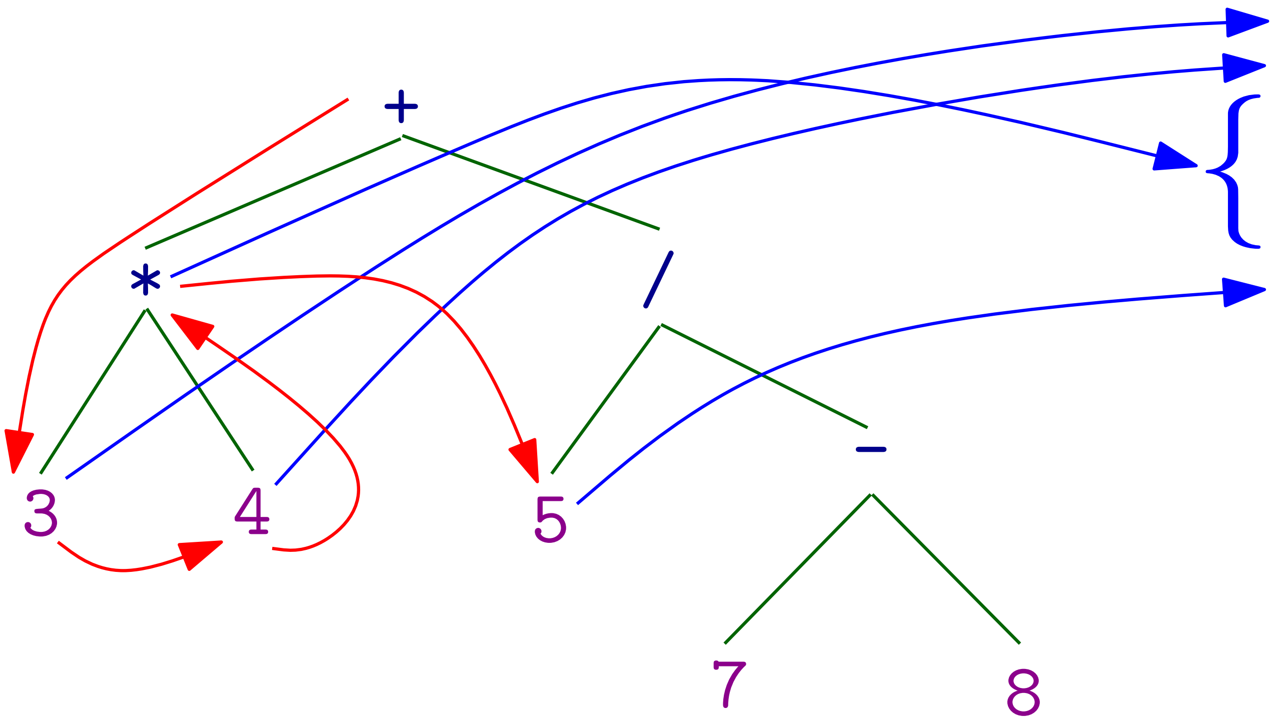
```
pushl $3
pushl $4
popl %ebx
popl %eax
imull %ebx,%eax
pushl %eax
pushl $5
pushl $7
pushl $8
popl %ebx
popl %eax
sub %ebx,%eax
pushl %eax
popl %ebx
popl %eax
cdq
idivl %ebx
pushl %eax
popl %ebx
popl %eax
addl %ebx,%eax
pushl %eax
```

Compilation by Post-Order Traversal



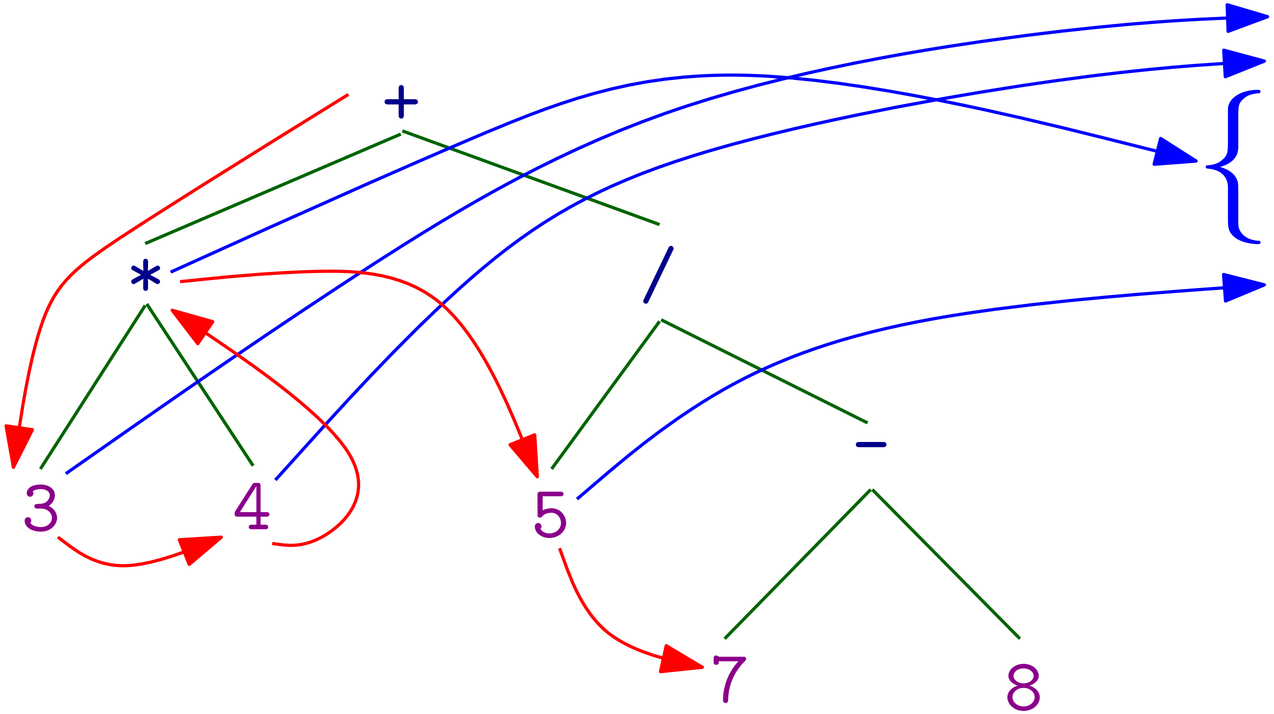
```
pushl $3
pushl $4
popl %ebx
popl %eax
imull %ebx,%eax
pushl %eax
pushl $5
pushl $7
pushl $8
popl %ebx
popl %eax
sub %ebx,%eax
pushl %eax
popl %ebx
popl %eax
cdq
idivl %ebx
pushl %eax
popl %ebx
popl %eax
addl %ebx,%eax
pushl %eax
```

Compilation by Post-Order Traversal



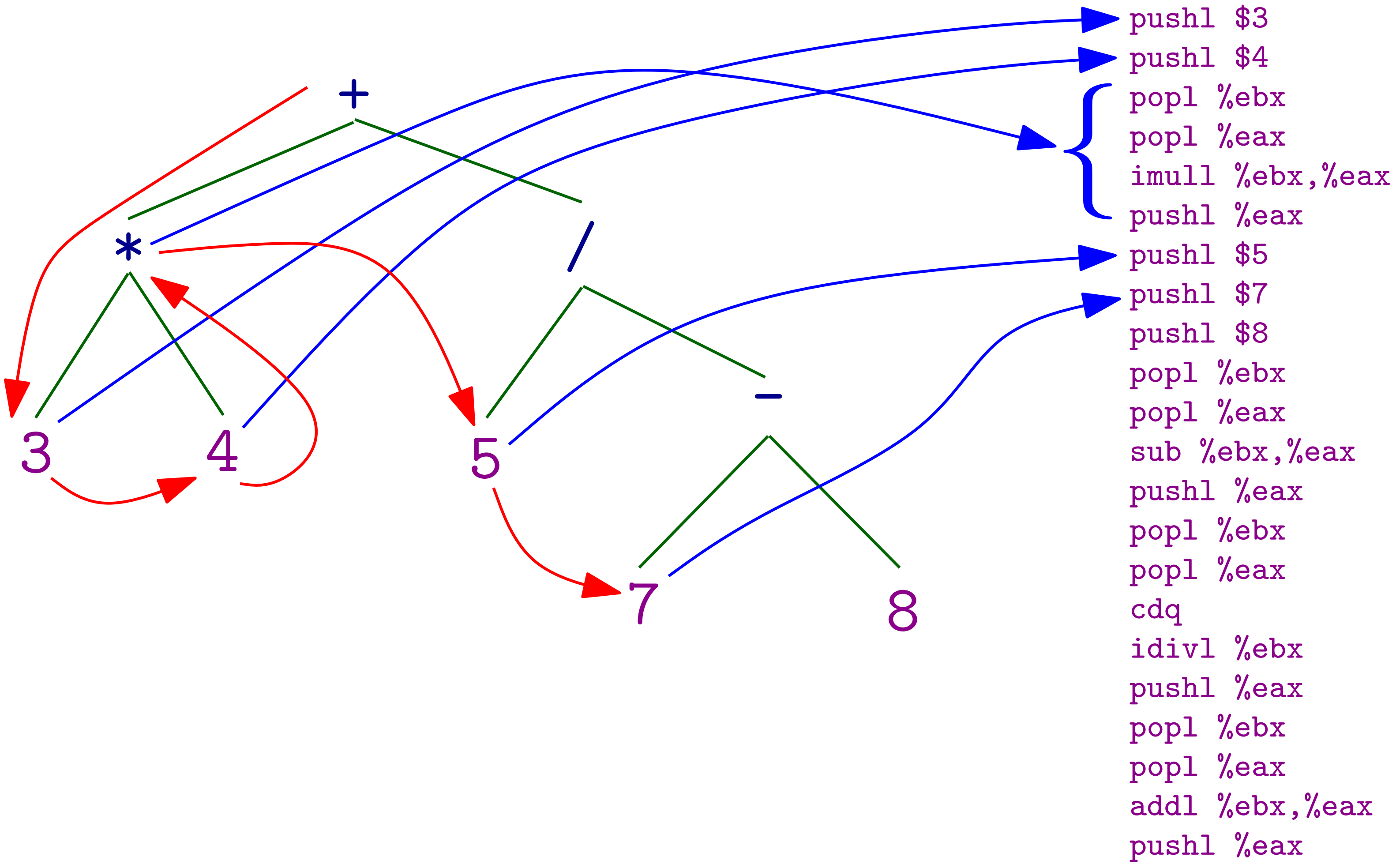
```
pushl $3
pushl $4
popl %ebx
popl %eax
imull %ebx,%eax
pushl %eax
pushl $5
pushl $7
pushl $8
popl %ebx
popl %eax
sub %ebx,%eax
pushl %eax
popl %ebx
popl %eax
cdq
idivl %ebx
pushl %eax
popl %ebx
popl %eax
addl %ebx,%eax
pushl %eax
```

Compilation by Post-Order Traversal



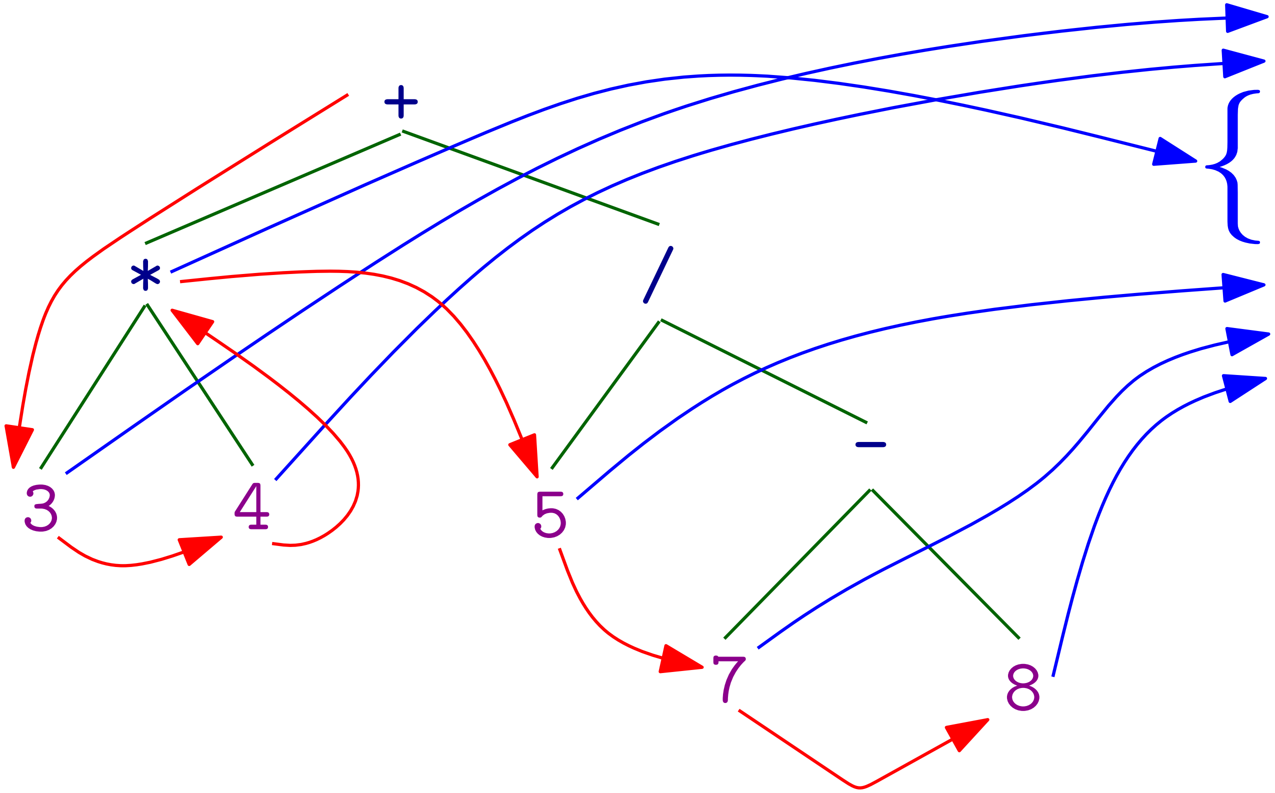
```
pushl $3
pushl $4
popl %ebx
popl %eax
imull %ebx,%eax
pushl %eax
pushl $5
pushl $7
pushl $8
popl %ebx
popl %eax
sub %ebx,%eax
pushl %eax
popl %ebx
popl %eax
cdq
idivl %ebx
pushl %eax
popl %ebx
popl %eax
addl %ebx,%eax
pushl %eax
```

Compilation by Post-Order Traversal



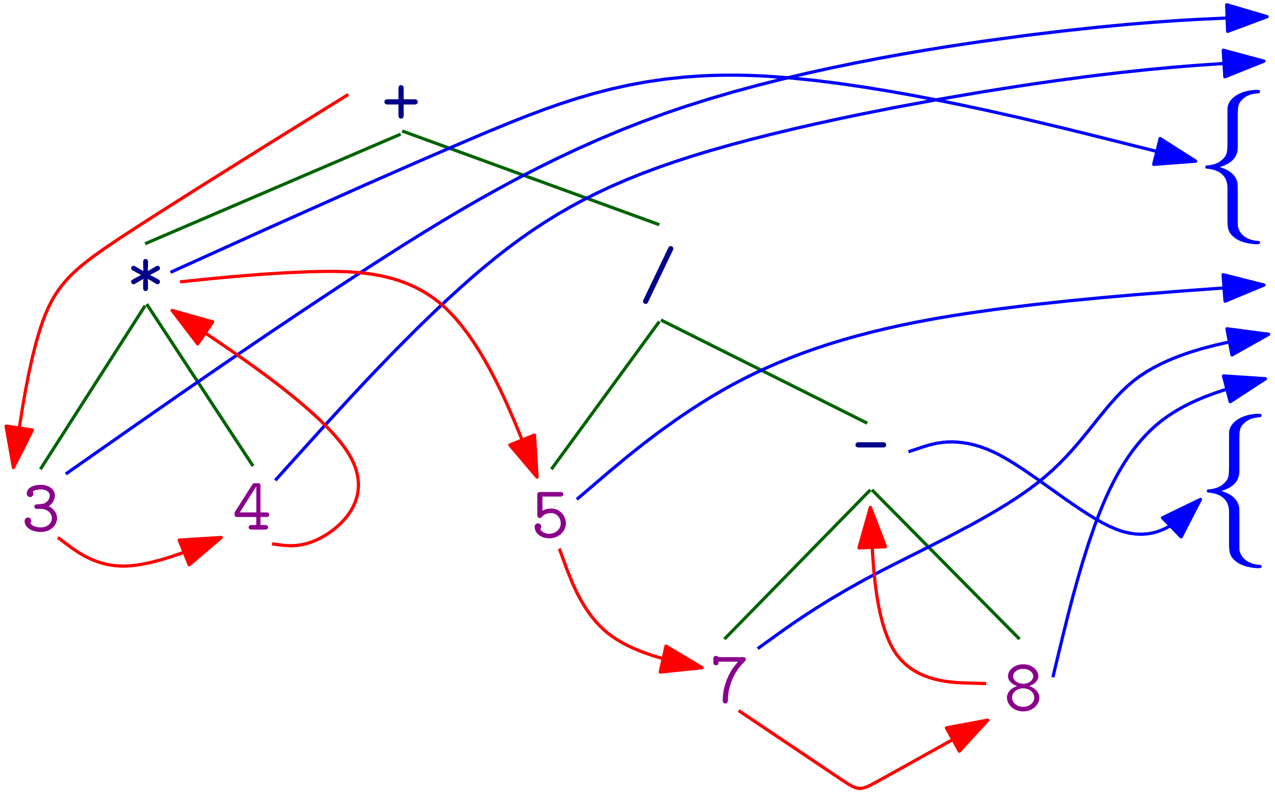
```
pushl $3
pushl $4
popl %ebx
popl %eax
imull %ebx,%eax
pushl %eax
pushl $5
pushl $7
pushl $8
popl %ebx
popl %eax
sub %ebx,%eax
pushl %eax
popl %ebx
popl %eax
cdq
idivl %ebx
pushl %eax
popl %ebx
popl %eax
addl %ebx,%eax
pushl %eax
```

Compilation by Post-Order Traversal



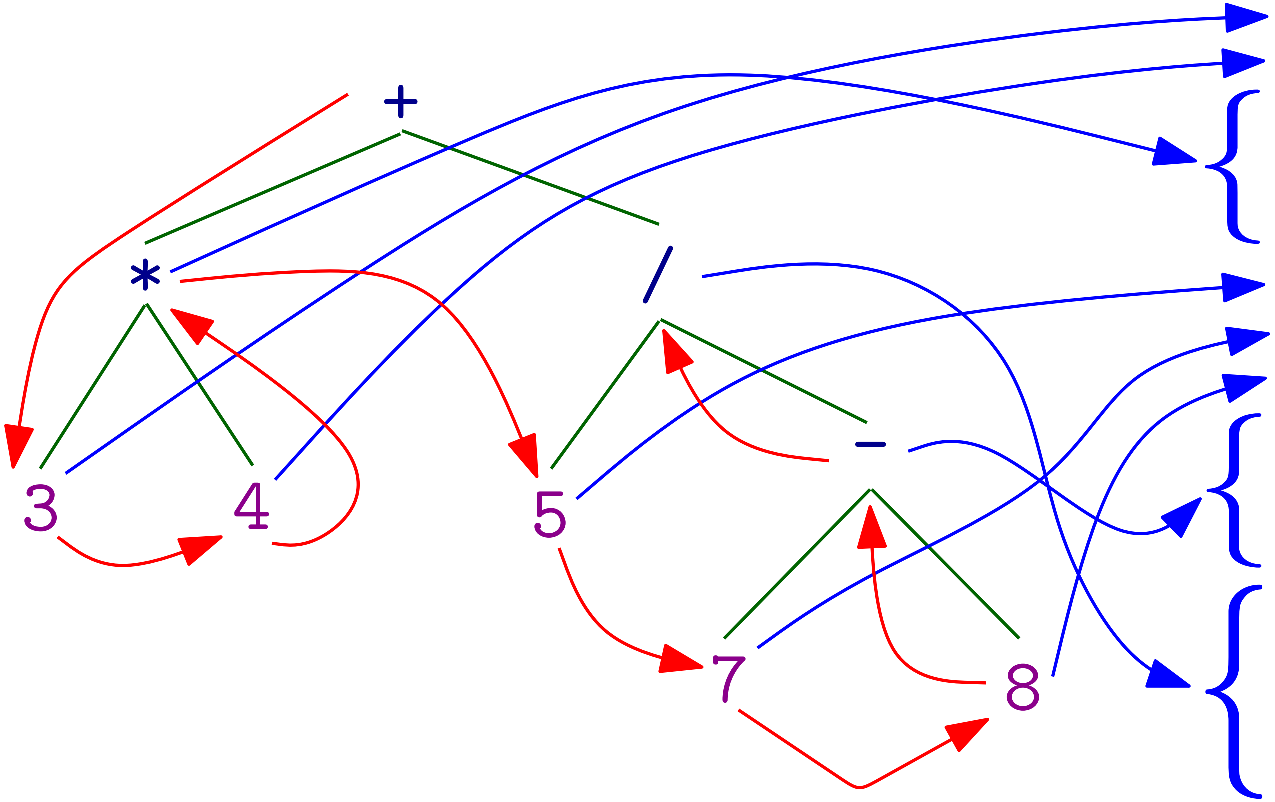
```
pushl $3
pushl $4
popl %ebx
popl %eax
imull %ebx,%eax
pushl %eax
pushl $5
pushl $7
pushl $8
popl %ebx
popl %eax
sub %ebx,%eax
pushl %eax
popl %ebx
popl %eax
cdq
idivl %ebx
pushl %eax
popl %ebx
popl %eax
addl %ebx,%eax
pushl %eax
```

Compilation by Post-Order Traversal



```
pushl $3
pushl $4
popl %ebx
popl %eax
imull %ebx,%eax
pushl %eax
pushl $5
pushl $7
pushl $8
popl %ebx
popl %eax
sub %ebx,%eax
pushl %eax
popl %ebx
popl %eax
cdq
idivl %ebx
pushl %eax
popl %ebx
popl %eax
addl %ebx,%eax
pushl %eax
```

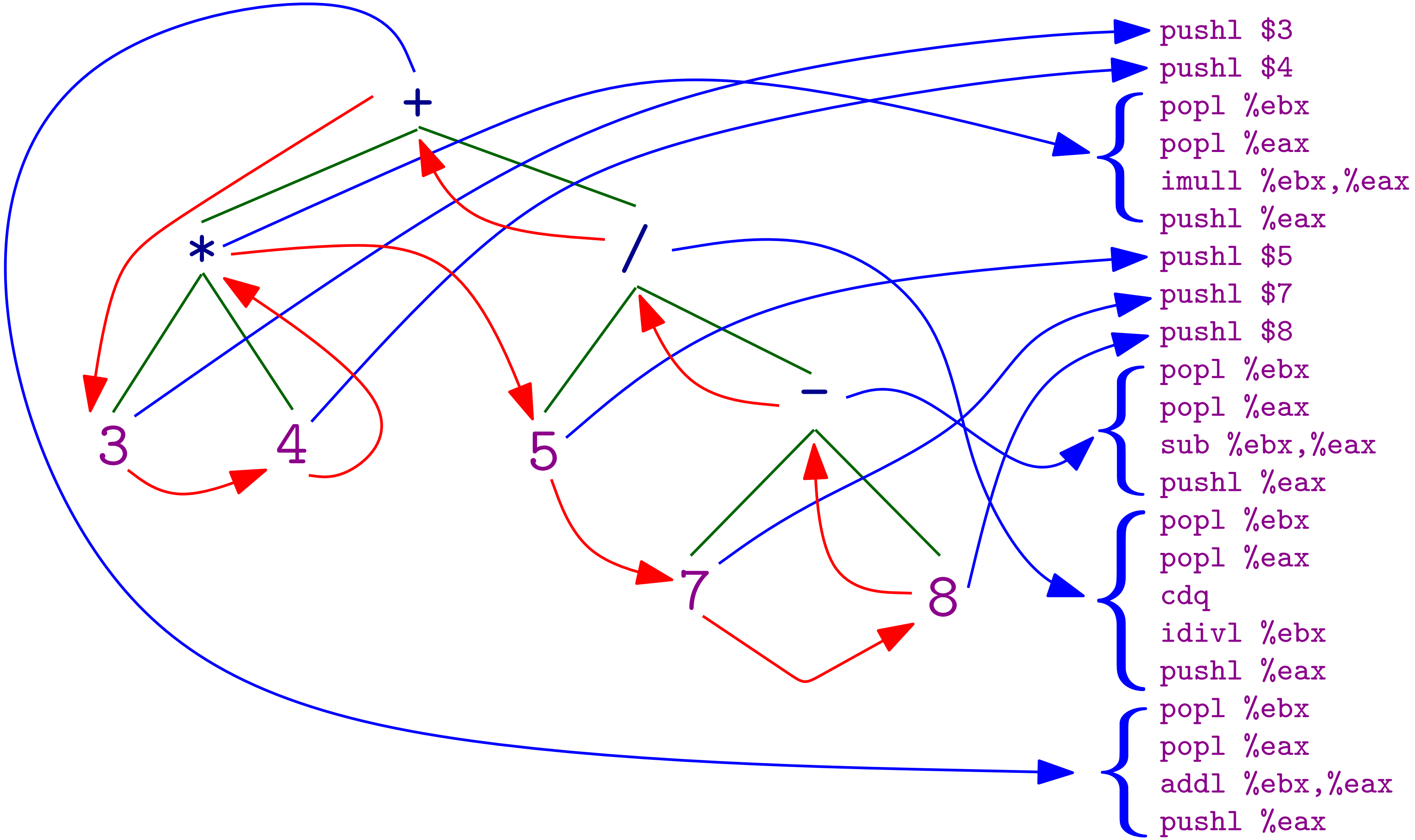
Compilation by Post-Order Traversal



```
pushl $3
pushl $4
popl %ebx
popl %eax
imull %ebx,%eax
pushl %eax
pushl $5
pushl $7
pushl $8
popl %ebx
popl %eax
sub %ebx,%eax
pushl %eax
popl %ebx
popl %eax
cdq
idivl %ebx
pushl %eax
popl %ebx
popl %eax
addl %ebx,%eax
pushl %eax
```

Compilation by Post-Order Traversal

[C]



```
pushl $3
pushl $4
popl %ebx
popl %eax
imull %ebx,%eax
pushl %eax
pushl $5
pushl $7
pushl $8
popl %ebx
popl %eax
sub %ebx,%eax
pushl %eax
popl %ebx
popl %eax
cdq
idivl %ebx
pushl %eax
popl %ebx
popl %eax
addl %ebx,%eax
pushl %eax
```

- For each constant, we push its value on the stack.
- Each operator pulls 2 operands from the stack, computes the result, and pushes it on the stack.
- Some instructions (IDIVL) have peculiar operands

Version 1

[C]

```
ce(C,[Instr]) :-
        integer(C),
        !,
        atomic_list_concat([’pushl $’,C],Instr).

ce(E,Code) :-
        E =.. [Op,E1,E2],
        member(Op,[+,-,*,/,rem]),
        !,
        ce(E1,C1),
        ce(E2,C2),
        cop(Op,Cop),
        append([C1,C2,Cop],Code).
```

Version 1

```
ce(C,[Instr]) :-
        integer(C),
        !,
        atomic_list_concat(['pushl $',C],Instr).

ce(E,Code) :-
        E =.. [Op,E1,E2],
        member(Op,[+,-,*,/,rem]),
        !,
        ce(E1,C1),
        ce(E2,C2),
        cop(Op,Cop),
        append([C1,C2,Cop],Code).
```

Post-order traversal

Code Issued for Operators

```
cop(+,[’popl %ebx’, ’popl %eax’, ’addl %ebx,%eax’, ’pushl %eax’]).

cop(-,[’popl %ebx’, ’popl %eax’, ’subl %ebx,%eax’, ’pushl %eax’]).

cop(*,[’popl %ebx’, ’popl %eax’, ’imull %ebx,%eax’, ’pushl %eax’]).

cop(/,[’popl %ebx’, ’popl %eax’, ’cdq’, ’idiv %ebx’, ’pushl %eax’]).

cop(rem,[’popl %ebx’, ’popl %eax’, ’cdq’, ’idiv %ebx’, ’pushl %edx’]).
```

Main Predicate

[C]

```
out(E) :-
        ce(E,Code),
        Pre = [ ’.section .text’,
                ’.globl _start’,
                ’_start:’,
                ’pushl %ebp’,
                ’movl %esp,%ebp’],
        Post = [’popl %eax’,
                ’movl %ebp,%esp’,
                ’popl %ebp’,
                ’ret’],
        append([Pre,Code,Post],All),
        atomic_list_concat([’’|All],’\n\t’,AllWritable),
        write(AllWritable).
```

Main Predicate

```
out(E) :-
        ce(E,Code),
        Pre = [ '.section .text',
                '.globl _start',
                '_start:',
                'pushl %ebp',
                'movl %esp,%ebp'],
        Post = ['popl %eax',
                'movl %ebp,%esp',
                'popl %ebp',
                'ret'],
        append([Pre,Code,Post],All),
        atomic_list_concat([''|All],'\n\t',AllWritable),
        write(AllWritable).
```

Preamble

Postamble

All code as single atom

Demo

```
?- out(1+2).

        .section .text
        .globl _start
        _start:
        pushl %ebp
        movl %esp,%ebp
        pushl $1
        pushl $2
        popl %ebx
        popl %eax
        addl %ebx,%eax
        pushl %eax
        popl %eax
        movl %ebp,%esp
        popl %ebp
        ret
true.
```

```c
#include <stdio.h>

int start() asm("_start") ;

int main() {
    printf("Result = %d\n",start()) ;
}
```

```
gcc -o test runtime.c test.s

./test

Result = 5
```

Demo

```
?- out(1+2).

        .section .text
        .globl _start
        _start:
        pushl %ebp
        movl %esp,%ebp
        pushl $1
        pushl $2
        popl %ebx
        popl %eax
        addl %ebx,%eax
        pushl %eax
        popl %eax
        movl %ebp,%esp
        popl %ebp
        ret
true.
```

```c
#include <stdio.h>

int start() asm("_start") ;

int main() {
    printf("Result = %d\n",start()) ;
}
```

```
gcc -o test runtime.c test.s

./test

Result = 5
```

Demo

```
?- out(2*3-4/5+6).

        .section .text
        .globl _start
        _start:
        pushl %ebp
        movl %esp,%ebp
        pushl $2
        pushl $3
        popl %ebx
        popl %eax
        imull %ebx,%eax
        pushl %eax
        pushl $4
        pushl $5
        popl %ebx
        popl %eax
        cdq
        idiv %ebx
        pushl %eax
```

```
        popl %ebx
        popl %eax
        subl %ebx,%eax
        pushl %eax
        pushl $6
        popl %ebx
        popl %eax
        addl %ebx,%eax
        pushl %eax
        popl %eax
        movl %ebp,%esp
        popl %ebp
        ret
true.
```

Code in `comp_expr_naive_1.pro`

Version 2

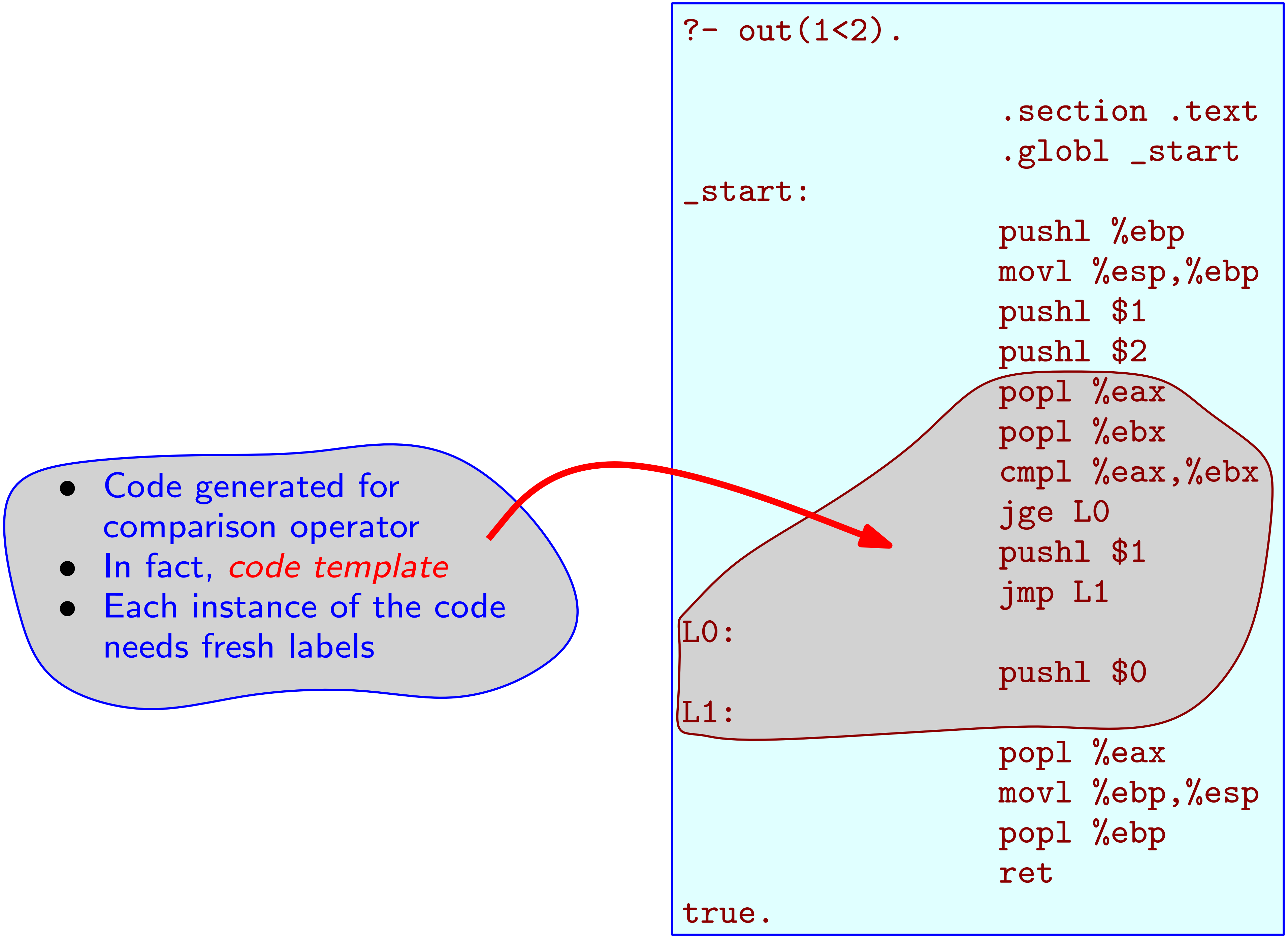
[C]

```
?- out(1<2).

                 .section .text
                 .globl _start
_start:
                 pushl %ebp
                 movl %esp,%ebp
                 pushl $1
                 pushl $2
                 popl %eax
                 popl %ebx
                 cmpl %eax,%ebx
                 jge L0
                 pushl $1
                 jmp L1
L0:
                 pushl $0
L1:
                 popl %eax
                 movl %ebp,%esp
                 popl %ebp
                 ret
true.
```

Version 2

[C]



- Code generated for comparison operator
- In fact, *code template*
- Each instance of the code needs fresh labels

```
?- out(1<2).

                 .section .text
                 .globl _start
_start:
                 pushl %ebp
                 movl %esp,%ebp
                 pushl $1
                 pushl $2
                 popl %eax
                 popl %ebx
                 cmpl %eax,%ebx
                 jge L0
                 pushl $1
                 jmp L1
L0:

                 pushl $0
L1:
                 popl %eax
                 movl %ebp,%esp
                 popl %ebp
                 ret
true.
```

Operator Code

```
cop(+,[],
    ['\n\t\t popl %ebx',
     '\n\t\t popl %eax',
     '\n\t\t addl %ebx,%eax',
     '\n\t\t pushl %eax']).


cop(<,[L1,L2],
    ['\n\t\t popl %eax',
     '\n\t\t popl %ebx',
     '\n\t\t cmpl %eax,%ebx',
     '\n\t\t jge ', L1,
     '\n\t\t pushl $1',
     '\n\t\t jmp ', L2,'\n',
L1,':',
     '\n\t\t pushl $0','\n',
L2,':'                  ]).
```

Operator Code

```
cop(+,[],
    ['\n\t\t popl %ebx',
     '\n\t\t popl %eax',
     '\n\t\t addl %ebx,%eax',
     '\n\t\t pushl %eax']).
```

Older operators generate the same code

```
cop(<,[L1,L2],
    ['\n\t\t popl %eax',
     '\n\t\t popl %ebx',
     '\n\t\t cmpl %eax,%ebx',
     '\n\t\t jge ', L1,
     '\n\t\t pushl $1',
     '\n\t\t jmp ', L2,'\n',
L1,':',
     '\n\t\t pushl $0','\n',
L2,':'                  ]).
```

Comparison generates a template, with variables L1 and L2 acting as placeholders for fresh variables

Post-Order Traversal

```
ce(C,[Instr],LabelSuffix,LabelSuffix) :-
        integer(C),!,
        atomic_list_concat([’\n\t\t pushl $’,C],Instr).

ce(E,Code,LabelSuffixIn,LabelSuffixOut) :-
        E =.. [Op,E1,E2],
        member(Op,[+,-,*,/,rem,<]),!,
        cop(Op,LPlaceholders,Cop),
        generateLabels(LPlaceholders,LabelSuffixIn,LabelSuffixAux1),
        ce(E1,C1,LabelSuffixAux1,LabelSuffixAux2),
        ce(E2,C2,LabelSuffixAux2,LabelSuffixOut),
        append([C1,C2,Cop],Code).
```

Post-Order Traversal

New arguments help generate new labels whenever they are needed. Since assignment is not available, we always need input and output versions.

```
ce(C,[Instr],LabelSuffix,LabelSuffix) :-
        integer(C),!,
        atomic_list_concat(['\n\t\t pushl $',C],Instr).

ce(E,Code,LabelSuffixIn,LabelSuffixOut) :-
        E =.. [Op,E1,E2],
        member(Op,[+,-,*,/,rem,<]),!,
        cop(Op,LPlaceholders,Cop),
        generateLabels(LPlaceholders,LabelSuffixIn,LabelSuffixAux1),
        ce(E1,C1,LabelSuffixAux1,LabelSuffixAux2),
        ce(E2,C2,LabelSuffixAux2,LabelSuffixOut),
        append([C1,C2,Cop],Code).
```

Predicate that bounds placeholders to fresh variables

Label Generator

```
generateLabels([],LabelSuffix,LabelSuffix).

generateLabels([H|T],LabelSuffixIn,LabelSuffixOut) :-
        atomic_list_concat([’L’,LabelSuffixIn],H),
        LabelSuffixAux #= LabelSuffixIn + 1,
        generateLabels(T,LabelSuffixAux,LabelSuffixOut).
```

Main Predicate for Version 2

```
out(E) :-
        ce(E,Code,0,_),
        Pre = [ ’\n\t\t .section .text’,
                ’\n\t\t .globl _start’,
                ’\n_start:’,
                ’\n\t\t pushl %ebp’,
                ’\n\t\t movl %esp,%ebp’],
        Post = [’\n\t\t popl %eax’,
                ’\n\t\t movl %ebp,%esp’,
                ’\n\t\t popl %ebp’,
                ’\n\t\t ret’],
        append([Pre,Code,Post],All),
        atomic_list_concat(All,AllWritable),
        write(AllWritable).
```

Demo

[C]

```
?- out((1<2)*((3<4)+2*4/5)).

                 .section .text
                 .globl _start
_start:
                 pushl %ebp
                 movl %esp,%ebp
                 pushl $1
                 pushl $2
                 popl %eax
                 popl %ebx
                 cmpl %eax,%ebx
                 jge L0
                 pushl $1
                 jmp L1
L0:
                 pushl $0
L1:
                 pushl $3
                 pushl $4
                 popl %eax
                 popl %ebx
                 cmpl %eax,%ebx
                 jge L2
                 pushl $1
                 jmp L3
L2:
                 pushl $0
L3:
```

```
                 pushl $2
                 pushl $4
                 popl %ebx
                 popl %eax
                 imull %ebx,%eax
                 pushl %eax
                 pushl $5
                 popl %ebx
                 popl %eax
                 cdq
                 idiv %ebx
                 pushl %eax
                 popl %ebx
                 popl %eax
                 addl %ebx,%eax
                 pushl %eax
                 popl %ebx
                 popl %eax
                 imull %ebx,%eax
                 pushl %eax
                 popl %eax
                 movl %ebp,%esp
                 popl %ebp
                 ret
true.
```

Code in `comp_expr_naive_2.pro`

Version 3

- We had to add 2 new arguments to the traversal predicate to implement label generation.
  - They are called *attributes*
- In general, as we add more features to the language, more attributes need to be added.
- From software engineering perspective, not a good idea to implement attributes as arguments to the predicate
- Better solution: add a dictionary (input+output versions) to list of arguments; put all attributes in the dictionary as (key,value) pairs

The New Post-Order Traversal Predicate

[C]

```
ce(C,[Instr],A,A) :-
        integer(C),!,
        atomic_list_concat([’\n\t\t pushl $’,C],Instr).

ce(E,Code,AIn,AOut) :-
        E =.. [Op,E1,E2],
        member(Op,[+,-,*,/,rem,<]),!,
        cop(Op,LPlaceholders,Cop),
        get_assoc(labelsuffix,AIn,LabelSuffixIn,Aaux1,LabelSuffixAux1),
        generateLabels(LPlaceholders,LabelSuffixIn,LabelSuffixAux1),
        ce(E1,C1,Aaux1,Aaux2),
        ce(E2,C2,Aaux2,AOut),
        append([C1,C2,Cop],Code).
```

The New Main Predicate

```
out(E) :-
        empty_assoc(Empty),
        put_assoc(labelsuffix,Empty,0,A),
        ce(E,Code,A,_),
        Pre = [ ’\n\t\t .section .text’,
                ’\n\t\t .globl _start’,
                ’\n_start:’,
                ’\n\t\t pushl %ebp’,
                ’\n\t\t movl %esp,%ebp’],
        Post = [’\n\t\t popl %eax’,
                ’\n\t\t movl %ebp,%esp’,
                ’\n\t\t popl %ebp’,
                ’\n\t\t ret’],
        append([Pre,Code,Post],All),
        atomic_list_concat(All,AllWritable),
        write(AllWritable).
```

Version 4: Demo

[C]

```
?- out(x=1;y=2;x+y,'test10.s').
true.
```

Code in comp_expr_naive_3.pro

```
                 .section .text
                 .globl _start
_start:
                 pushl %ebp
                 movl %esp,%ebp
                 pushl $1
                 popl %eax
                 movl %eax,x
                 pushl %eax
                 popl %eax
                 pushl $2
                 popl %eax
                 movl %eax,y
                 pushl %eax
```

```
    popl %eax
    pushl x
    pushl y
    popl %ebx
    popl %eax
    addl %ebx,%eax
    pushl %eax
    popl %eax
    movl %ebp,%esp
    popl %ebp
    ret

    .comm x,4,4
    .comm y,4,4
```

Version 4: Demo

```
?- out(x=1;y=2;x+y,'test10.s').
true.
```

Code in `comp_expr_naive_3.pro`

```
                   .section .text
                   .globl _start
_start:
                   pushl %ebp
                   movl %esp,%ebp
                   pushl $1
              =    popl %eax
                   movl %eax,x
                   pushl %eax
              ;    popl %eax
                   pushl $2
              =    popl %eax
                   movl %eax,y
                   pushl %eax
```

```
   popl %eax
   pushl x
   pushl y
   popl %ebx
   popl %eax
   addl %ebx,%eax
   pushl %eax
   popl %eax
   movl %ebp,%esp
   popl %ebp
   ret

   .comm x,4,4
   .comm y,4,4
```

List of variables must be collected so as to reserve space for them ⟹ another attribute

Post-Order Traversal [C]

```
ce(C,[Instr],A,A) :-
        (   integer(C), P = '$' ; atom(C),P='' ),!,
        atomic_list_concat(['\n\t\t pushl ',P,C],Instr).
ce(E,Code,AIn,AOut) :-
        E =.. [Op,E1,E2],
        member(Op,[+,-,*,/,rem,<,=]),!,
        cop(Op,LPlaceholders,Cop),
        (   Op = (=)
        ->  atom(E1),
            get_assoc(vars,AIn,OldVars,Aaux,NewVars),
            union(OldVars,[E1],NewVars),
            ce(E2,C2,Aaux,AOut),
            LPlaceholders = [E1],
            append([C2,Cop],Code)
        ;   get_assoc(labelsuffix,AIn,LabelSuffixIn,Aaux1,LabelSuffixAux1),
            generateLabels(LPlaceholders,LabelSuffixIn,LabelSuffixAux1),
            ce(E1,C1,Aaux1,Aaux2),
            ce(E2,C2,Aaux2,AOut),
            append([C1,C2,Cop],Code) ).
ce((S1;S2),Code,Ain,Aout) :-
        ce(S1,C1,Ain,Aaux),
        ce(S2,C2,Aaux,Aout),
        append([C1,['\n\t\t popl %eax'],C2], Code).
```

The Operator Code

```
cop(=,[V],
    [’\n\t\t popl %eax’,
     ’\n\t\t movl %eax,’,V,
     ’\n\t\t pushl %eax’ ]).
```

The Main Predicate

```
out(E,File) :-
        tell(File),
        empty_assoc(Empty),
        put_assoc(labelsuffix,Empty,0,A1),
        put_assoc(vars,A1,[],A2),
        ce(E,Code,A2,A3),
        Pre = [ ’\n\t\t .section .text’,
                ’\n\t\t .globl _start’,
                ’\n_start:’,
                ’\n\t\t pushl %ebp’,
                ’\n\t\t movl %esp,%ebp’],
        Post = [’\n\t\t popl %eax’,
                ’\n\t\t movl %ebp,%esp’,
                ’\n\t\t popl %ebp’,
                ’\n\t\t ret’],
        append([Pre,Code,Post],All),
        atomic_list_concat(All,AllWritable),
        writeln(AllWritable),
        get_assoc(vars,A3,VarList),
        allocvars(VarList,VarCode),
        atomic_list_concat(VarCode,WritableVars),
        write(WritableVars),
        told.
```

Variable Allocator

```
allocvars([],[]).
allocvars([V|VT],[D|DT]) :-
        atomic_list_concat([’\n\t\t .comm ’,V,’,4,4’],D),
        allocvars(VT,DT).
```

Demo

Code in comp_expr_naive_4.pro

[C]

```
?- out(x=1;y=4/2+(0<1);x+2*y,’test0.s’).
true.
```

```
        .section .text
        .globl _start
_start:
        pushl %ebp
        movl %esp,%ebp
        pushl $1
        popl %eax
        movl %eax,x
        pushl %eax
        popl %eax
        pushl $4
        pushl $2
        popl %ebx
        popl %eax
        cdq
        idiv %ebx
        pushl %eax
        pushl $0
        pushl $1
        popl %eax
        popl %ebx
        cmpl %eax,%ebx
        jge L0
        pushl $1
        jmp L1
L0:
        pushl $0
L1:
```

```
        popl %ebx
        popl %eax
        addl %ebx,%eax
        pushl %eax
        popl %eax
        movl %eax,y
        pushl %eax
        popl %eax
        pushl x
        pushl $2
        pushl y
        popl %ebx
        popl %eax
        imull %ebx,%eax
        pushl %eax
        popl %ebx
        popl %eax
        addl %ebx,%eax
        pushl %eax
        popl %eax
        movl %ebp,%esp
        popl %ebp
        ret

        .comm x,4,4
        .comm y,4,4
```

Conclusion

- Syntax-based processing is achieved by post-order traversal of the AST
  - May require multiple traversals with more complicated language constructs
  - The state of the translation process is recorded in *attributes* (computed or inherited)
  - New features in the language usually require new attributes in the traversal predicate

- The generated code is very inefficient
  - The stack discipline is very simple, but under-utilizes the registers
  - We devise each code template to work independently of the siblings of the current node.
  - Instructions at node boundary may become redundant
  - Hard to optimize in the traversal process; optimization performed later, on the whole generated code.
  - We shall see better utilization of the registers in next recitation.