How Compilers Work

by Walter Bright

Digital Mars

Compilers I've Built

- D programming language
- C++
- C
- Javascript
- Java
- A.B.E.L

Compiler Compilers

- Regex
- Lex
- Yacc
- Spirit

- Do only the easiest part
- Not very customizable
- Make compiler source less portable
- Not worth the bother

Source Code

```
extern int printf(const char *, ...);

int main(int argc, char **argv)
{
 // Print out each argument
 for (int i = 0; i < argc; i++)
 {
  printf("arg[%d] = '%s'\n",i,argv[i]);
 }
 return 0;
}
```

Compiler Output

```
000: 80  a  0  8 74 65 73 74 2e 63 70 70  0 88  6  0   ....test.cpp....
010:  0 9d 37 6e 4f  0 88  6  0  0 a1  1 43 56  0 96   ..7nO.......CV..
020: 2c  0  0  4 46 4c 41 54  5 5f 54 45 58 54  4 43   ,...FLAT._TEXT.C
030: 4f 44 45  5 5f 44 41 54 41  4 44 41 54 41  5 43   ODE._DATA.DATA.C
040: 4f 4e 53 54  4 5f 42 53 53  3 42 53 53  0 99  9   ONST._BSS.BSS...
050:  0 a9 2d  0  0  0  3  4  1  0 99  9  0 a9 10  0   ..-.............
060:  0  0  5  6  1  0 99  9  0 a9  0  0  0  0  7  7   ................
070:  1  0 99  9  0 a9  0  0  0  0  8  9  1  0 9a  2   ................
080:  0  2  0 88  6  0  0 9f 53 4e 4e  0 8c 1a  0  e   ........SNN.....
090: 5f 5f 61 63 72 74 75 73 65 64 5f 63 6f 6e  0  7   __acrtused_con..
0a0: 5f 70 72 69 6e 74 66  0  0 91  e  0  0  1  5 5f   _printf........_
0b0: 6d 61 69 6e  0  0  0  0  0  0 a1 33  0  1  0  0   main.......3....
0c0:  0  0 53 31 db 56 8b 74 24  c 85 f6 57 8b 7c 24   ..S1.V.t$...W.|$
0d0: 14 7e 16 ff 34 9f 53 68  0  0  0  0 e8  0  0  0   .~..4.Sh........
0e0:  0 43 83 c4  c 39 f3 7c ea 5f 31 c0 5e 5b c3  0   .C...9.|._1.^[..
0f0: 9d  b  0 a4 1b 16  1  2 e4 16 14  1  2  0 a1 16   ................
100:  0  2  0  0  0  0 61 72 67 5b 25 64 5d 20 3d 20   ......arg[%d] = 
110: 27 25 73 27  a  0  0 8a  2  0  0  0               '%s'........
```

Compiler Passes

١. Lexing
٢. Parsing
٣. Semantic Analysis
٤. Intermediate Code Generation
٥. Optimization
٦. Code Generation
٧. Object File Generation

Lexing

- turns character stream into tokens
- eliminates whitespace
- eliminates comments
- distinguishes keywords from identifiers
- strings, numbers turn into single tokens
- input is dramatically simplified

Tokenized Result

Token Name	Optional Data
extern	
int	
identifier	printf
lparen	
const	
char	
star	
identifier	format
comma	
ellipsis	
rparen	
semicolon	
int	
identifier	main
lparen	
int	
identifier	argc
comma	
char	
star	
star	
identifier	argv
rparen	
lbrace	
for	
lparen	
int	
identifier	i
equals	
number	0
semicolon	
identifier	i
lessthan	
identifier	argc
semicolon	
identifier	i
plusplus	
rparen	
lbrace	
identifier	printf
lparen	
string	“arg[%d] = '%s'\n”
comma	
identifier	argv
lbracket	
identifier	i
rbracket	
rparen	
semicolon	
rbrace	
return	
number	0
semicolon	
rbrace	

249 characters of source becomes 54 'characters' after lexing

Parsing

Grammar looks like:

```
ForStatement:
 for(Declaration; Expression; Expression)
   Statement
```

Data structure looks like:

```
struct ForStatement : Statement
{
    Declaration* decl;
    Expression*  cond;
    Expression*  inc;
    Statement*   body;
};
```

Note close correspondence

Semantic Analysis

1. Declaring symbols
2. Resolving symbols
3. Type determination
4. Type checking
5. Language rules checking
6. Overload resolution
7. Template expansion
8. Inlining functions

Results of Semantic Analysis

Compiled program is an array of symbols.

Symbols point to the parsed data structures.

Parsed data structures are 'decorated' with types, attributes, storage classes, and other information needed to generate intermediate code.

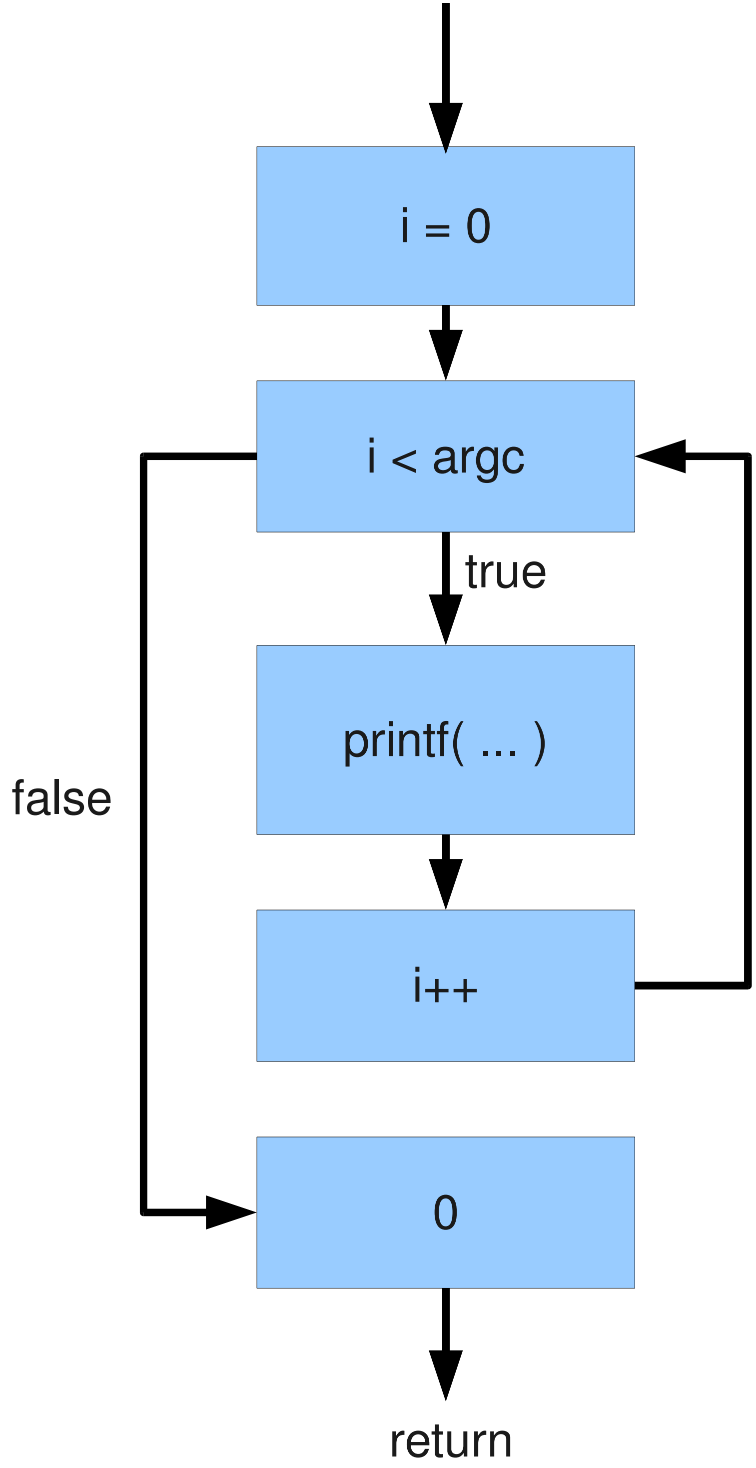
Intermediate Code

- A 'pseudo machine' is targetted
- Often several front ends target this 'pseudo machine'
  - Sharing a common optimizer and native code generator
- Most semantic information is stripped
  - gcc is a prime example of this
  - Digital Mars compilers do it too
- Interpreters tend to go no further
- An interpreter like the Java VM and .NET execute intermediate code
  - Optimization and native code gen passes are done at runtime using a JIT

Intermediate Code Generation

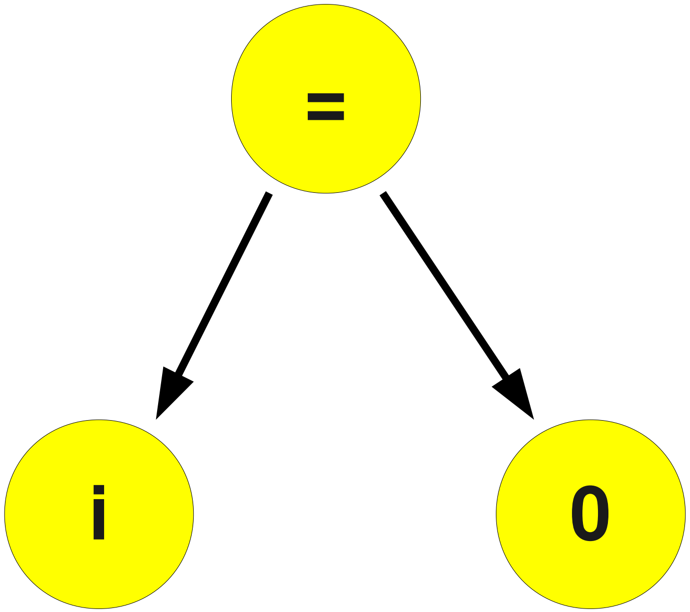
Symbols resulting from semantic analysis are "walked" to generate intermediate code.

The statements are converted into basic blocks connected by edges.

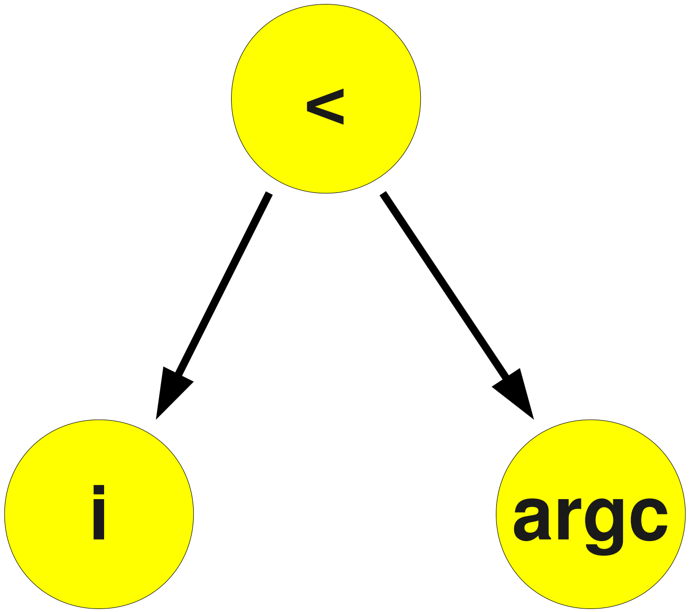

i = 0
i < argc
true
printf(...)
false
i++
0
return

Expressions to Trees

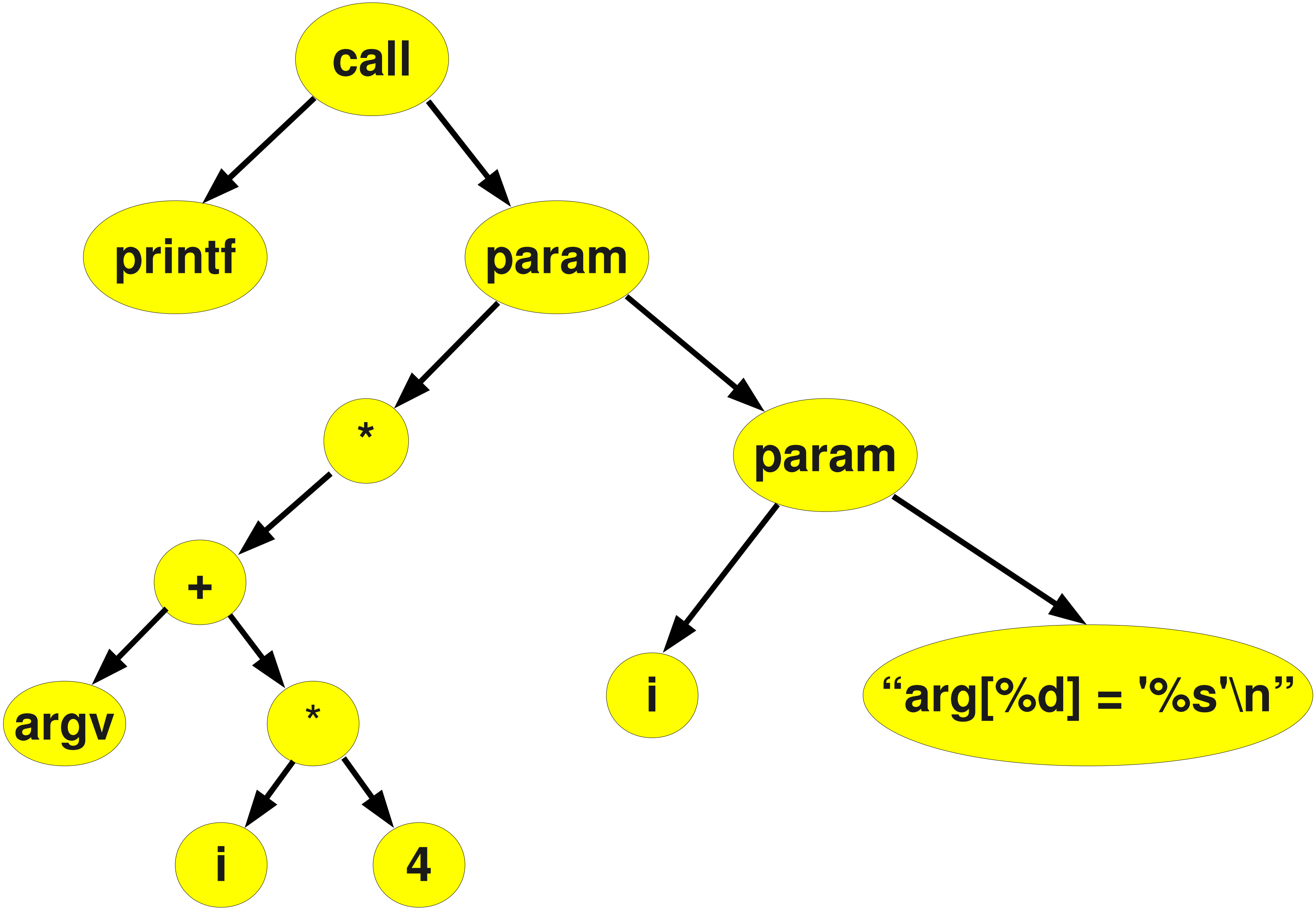
i = 0



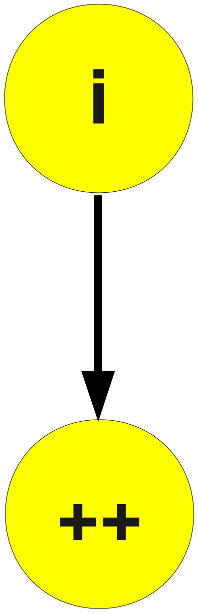
i < argc



```
printf("arg[%d] = '%s'\n", i, argv[i]);
```



i++



i

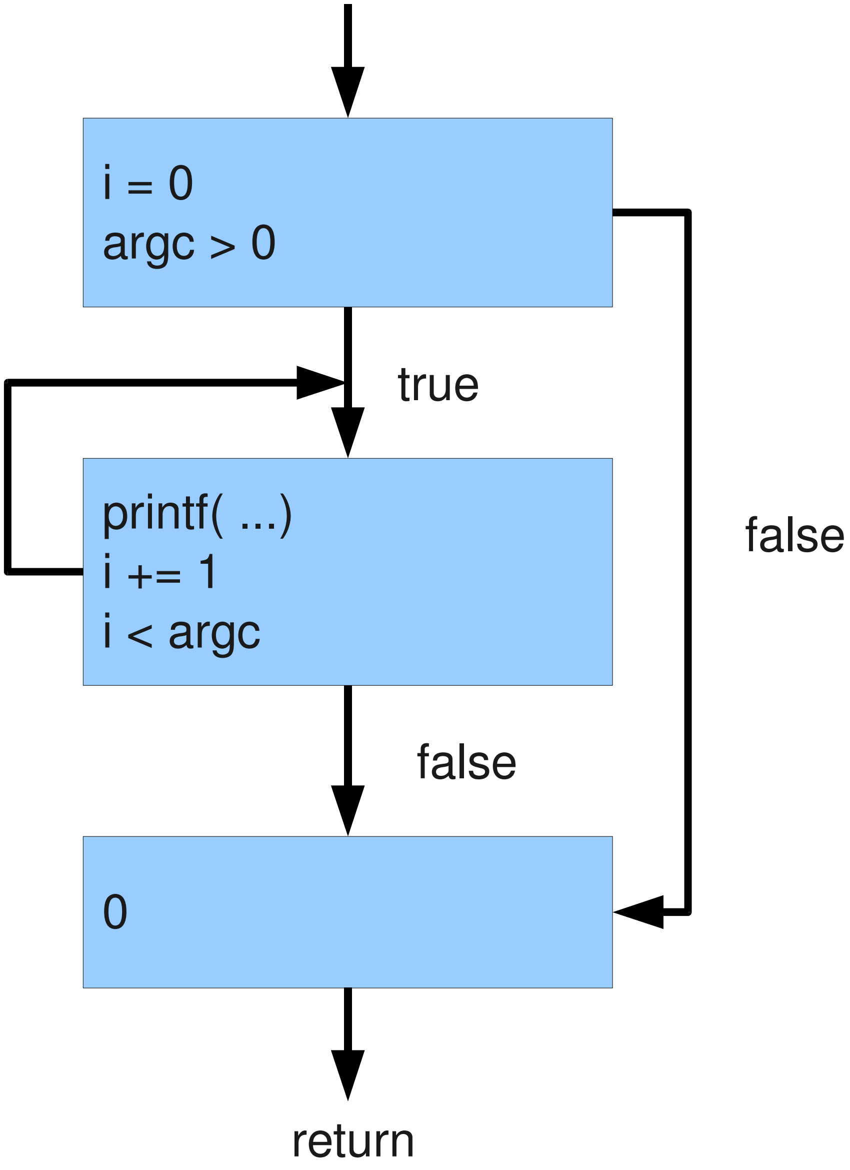
++

0

0

Optimization

Rewrites the expressions and the basic blocks to an optimized form



Note the loop rotation and the replacement of

i++

with

i += 1

Reason for Loop Rotation

- one less jump
- can sometimes eliminate loop header, such as the loop:

```
for (int i = 0; i < 10; i++)
{ body }
```

is rewritten as:

```
int i = 0;
do {
  body
} while (++i < 10);
```

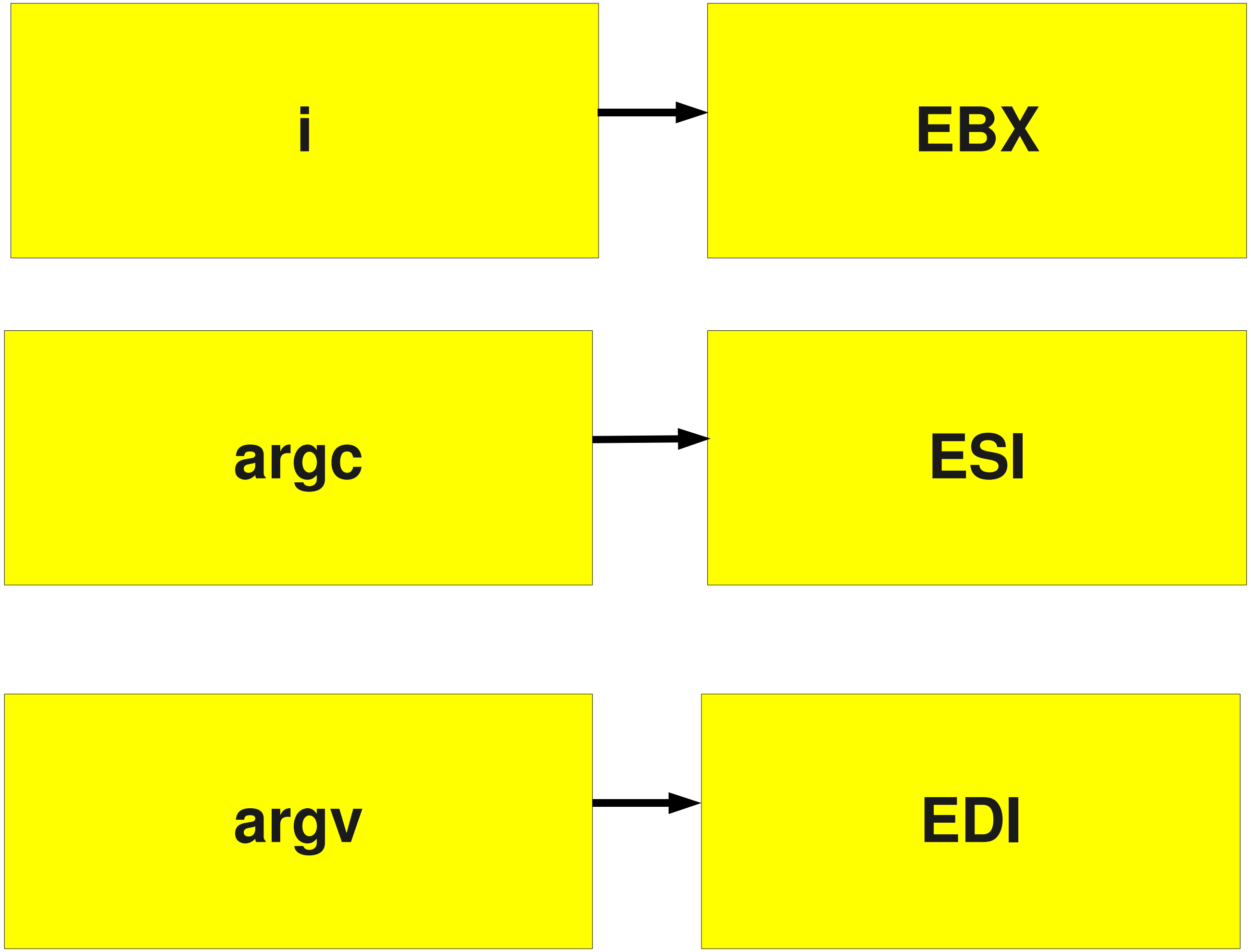
Other Optimizations

- Constant propagation
- Copy propagation
- Common subexpressions
- Strength reduction
- Constant folding
- Loop induction variables

More Optimizations

- Very busy expressions
- Tail call elimination
- Dead assignment elimination
- Live variable analysis
- Code hoisting
- Data flow analysis

Register Assignment



i → EBX

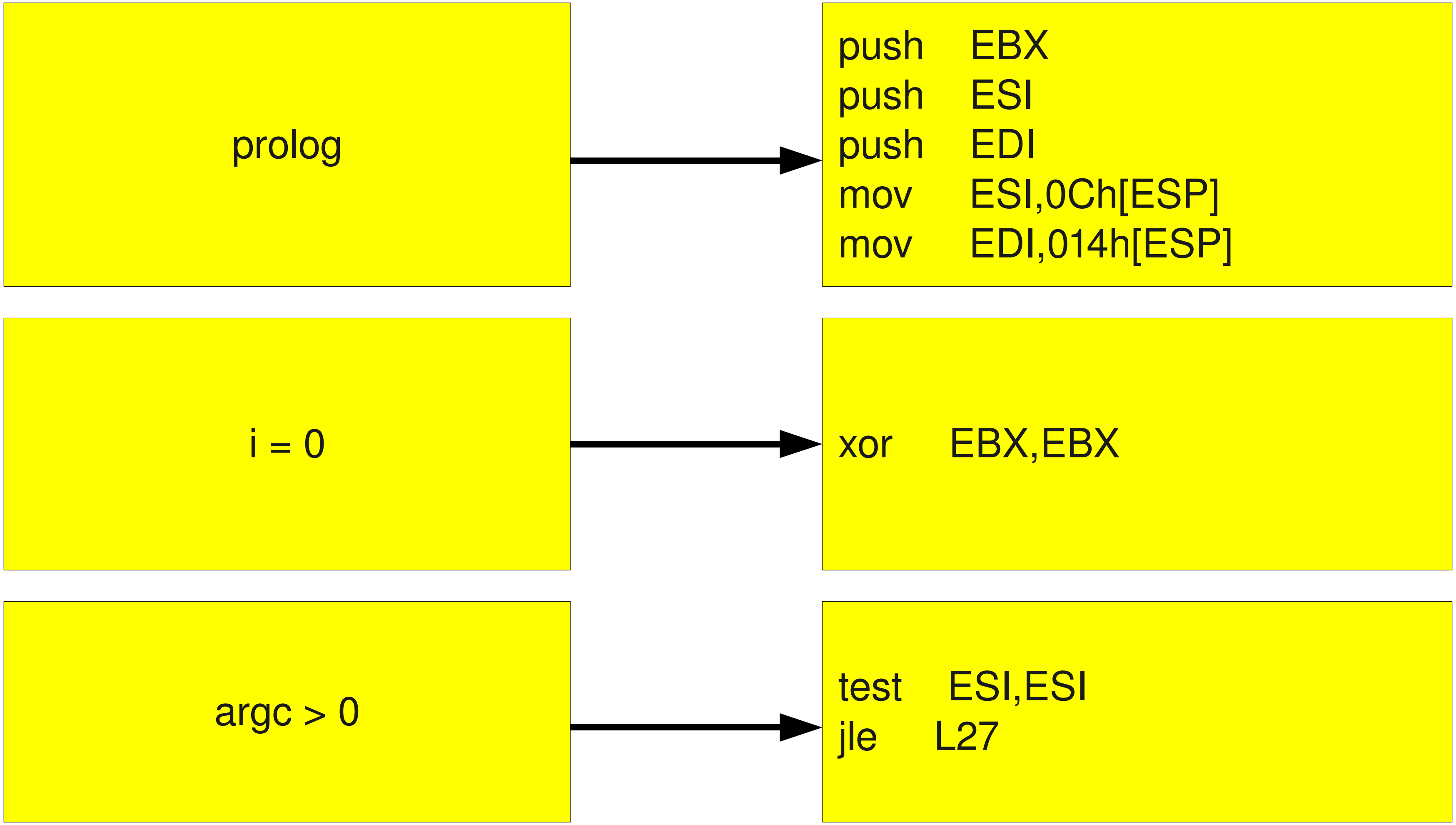
argc → ESI

argv → EDI

Register Assignment Methods

- Registers reserved for variables
  - Good when there are many registers
- By live analysis
  - More advanced
  - Best when there are few registers

Instruction Selection



prolog

```
push   EBX
push   ESI
push   EDI
mov    ESI,0Ch[ESP]
mov    EDI,014h[ESP]
```

i = 0

```
xor    EBX,EBX
```

argc > 0

```
test   ESI,ESI
jle    L27
```

`printf("arg[%d] = '%s'\n", i, argv[i])` →

```
push    [EBX*4][EDI]
push    EBX
push    offset _DATA
call    _printf
add     ESP,0Ch
```

`i += 1` →

```
inc     EBX
```

`i < argc` →

```
cmp     EBX,ESI
jl      L11
```

`0` →

```
xor     EAX,EAX
```

epilog →

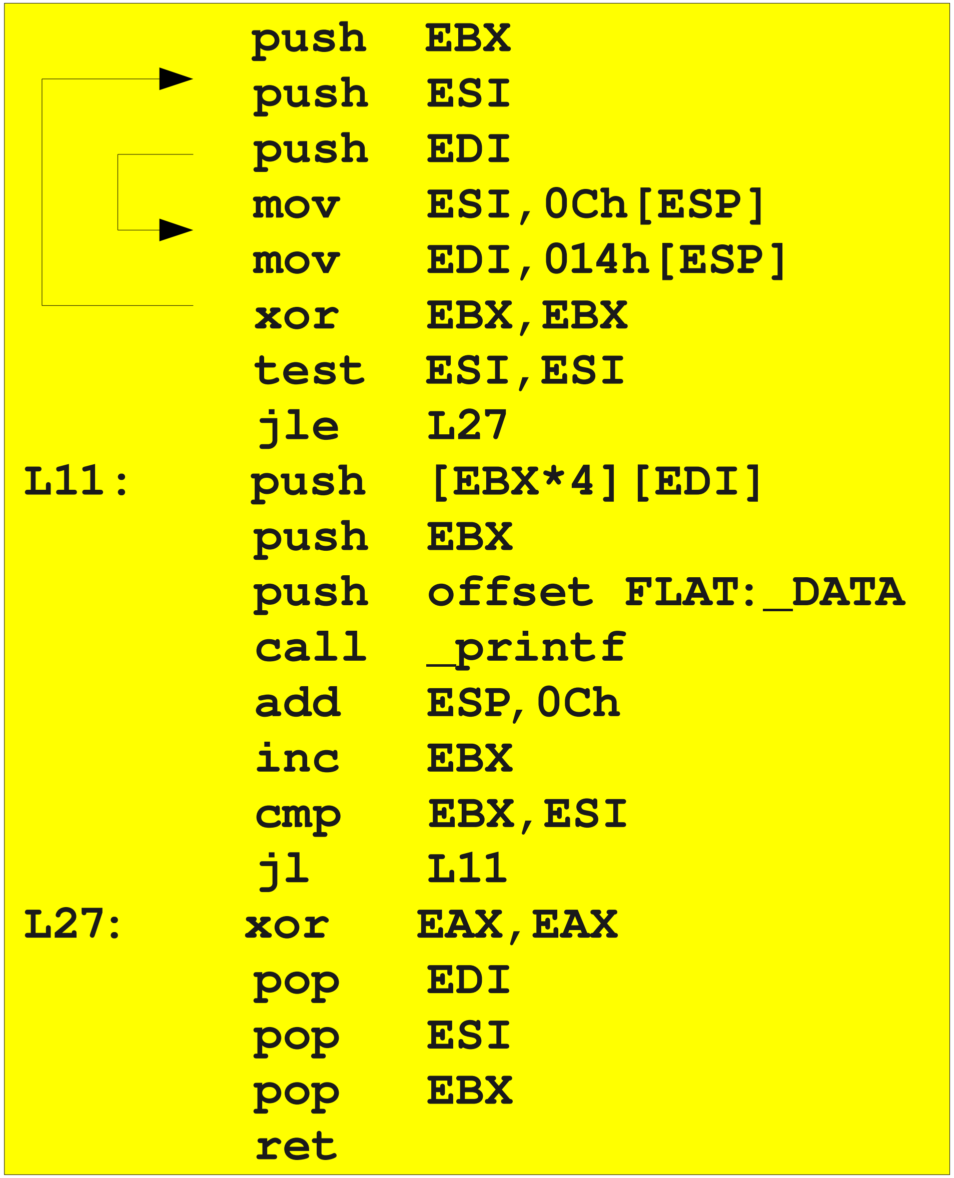
```
pop     EDI
pop     ESI
pop     EBX
ret
```

Putting It Together

```
        push    EBX
        push    ESI
        push    EDI
        mov     ESI,0Ch[ESP]
        mov     EDI,014h[ESP]
        xor     EBX,EBX
        test    ESI,ESI
        jle     L27
L11:    push    [EBX*4][EDI]
        push    EBX
        push    offset FLAT:_DATA
        call    _printf
        add     ESP,0Ch
        inc     EBX
        cmp     EBX,ESI
        jl      L11
L27:    xor     EAX,EAX
        pop     EDI
        pop     ESI
        pop     EBX
        ret
```

Instruction Scheduling

- Do register loads as soon as possible
- Do register reads as late as possible



```
        push   EBX
        push   ESI
        push   EDI
        mov    ESI,0Ch[ESP]
        mov    EDI,014h[ESP]
        xor    EBX,EBX
        test   ESI,ESI
        jle    L27
L11:    push   [EBX*4][EDI]
        push   EBX
        push   offset FLAT:_DATA
        call   _printf
        add    ESP,0Ch
        inc    EBX
        cmp    EBX,ESI
        jl     L11
L27:    xor    EAX,EAX
        pop    EDI
        pop    ESI
        pop    EBX
        ret
```

```
        push EBX
        xor  EBX,EBX
        push ESI
        mov  ESI,0Ch[ESP]
        test ESI,ESI
        push EDI
        mov  EDI,014h[ESP]
        jle  L27
L11:    push [EBX*4][EDI]
        push EBX
        push offset FLAT:_DATA
        call _printf
        inc  EBX
        add  ESP,0Ch
        cmp  EBX,ESI
        jl   L11
L27:    pop  EDI
        xor  EAX,EAX
        pop  ESI
        pop  EBX
        ret
```

Object File Generation

```
_TEXT  segment dword use32 public 'CODE'      ;size is 45
_TEXT  ends
_DATA  segment dword use32 public 'DATA'      ;size is 16
_DATA  ends
CONST  segment dword use32 public 'CONST'     ;size is 0
CONST  ends
_BSS   segment dword use32 public 'BSS' ;size is 0
_BSS   ends
FLAT   group
includelib SNN.lib
       extrn  __acrtused_con
       extrn  _printf

       public _main
_TEXT  segment
       assume CS:_TEXT
_main:
       53           push   EBX
       31 DB        xor    EBX,EBX
       56           push   ESI
       8B 74 24 0C  mov    ESI,0Ch[ESP]
       85 F6        test   ESI,ESI
       57           push   EDI
       8B 7C 24 14  mov    EDI,014h[ESP]
       7E 16        jle    L27
L11:   FF 34 9F     push   [EBX*4][EDI]
       53           push   EBX
       68 00 00 00 00      push   offset FLAT:_DATA
       E8 00 00 00 00      call   near ptr _printf
       43           inc    EBX
       83 C4 0C     add    ESP,0Ch
       39 F3        cmp    EBX,ESI
       7C EA        jl     L11
L27:   5F           pop    EDI
       31 C0        xor    EAX,EAX
       5E           pop    ESI
       5B           pop    EBX
       C3           ret
_TEXT  ends
_DATA  segment
D0     db     061h,072h,067h,05bh,025h,064h,05dh,020h
       db     03dh,020h,027h,025h,073h,027h,00ah,000h
_DATA  ends
CONST  segment
CONST  ends
_BSS   segment
_BSS   ends
       end
```

Object File Contents

Intel Object Module Format

Header	THEADR, COMENT
Symbol Table	EXTDEF, PUB386
Code/Data Sections	SEG386, GRPDEF, LNAMES
Section Contents	LED386
Fixups	FIX386
Footer	MODEND

```
THEADR   8 74 65 73 74 2e 63 70 70                                .test.cpp
COMENT   0 9d 37 6e 4f                                            ..7nO
COMENT   0 a1  1 43 56                                            ...CV
LNAMES   0  4 46 4c 41 54  5 5f 54 45 58 54  4 43 4f 44           ..FLAT._TEXT.COD
        45  5 5f 44 41 54 41  4 44 41 54 41  5 43 4f 4e           E._DATA.DATA.CON
        53 54  4 5f 42 53 53  3 42 53 53                          ST._BSS.BSS
SEG386 a9 33  0  0  0  3  4  1                                    .3......
SEG386 a9 10  0  0  0  5  6  1                                    ........
SEG386 a9  0  0  0  0  7  7  1                                    ........
SEG386 a9  0  0  0  0  8  9  1                                    ........
GRPDEF   2                                                        .
COMENT   0 9f 53 4e 4e                                            ..SNN
EXTDEF   e 5f 5f 61 63 72 74 75 73 65 64 5f 63 6f 6e  0           .__acrtused_con.
         7 5f 70 72 69 6e 74 66  0                                ._printf.
PUB386   0  1  5 5f 6d 61 69 6e  0  0  0  0  0                    ..._main.....
LED386   1  0  0  0  0 c8  4  0  0 c7 45 fc  0  0  0  0           ..........E.....
        8b 45 fc 3b 45  8 7d 1c 8b 4d fc 8b 55  c ff 34           .E.;E.}..M..U..4
        8a 51 68  0  0  0  0 e8  0  0  0  0 83 c4  c ff           .Qh.............
        45 fc eb dc 31 c0 c9 c3                                   E...1...
FIX386 a4 23 16  1  2 e4 1e 14  1  2                              .#........
LED386   2  0  0  0  0 61 72 67 5b 25 64 5d 20 3d 20 27           .....arg[%d] = '
        25 73 27  a  0                                            %s'..
MODEND   0                                                         .
```

Other Compiler Topics

- Exception handling
- Thread local storage
- Closures
- Virtual functions

More Topics

- RAII
- Position Independent Code
- Concurrency
- Runtime Library
- Symbolic debug information

Conclusion

- Compilers are complex, but can be broken down into well understood passes
- Best way to learn a language is to write a compiler for it
- Compilers are fun