Generació de codi LLVM

Compilador de Asl a LLVM

jpbasic_genc_01.asl:

```
func f1()
    var x1: int
    var y1: int
    write "err!!\n";
    if x1 == y1*2 then
        x1 = y1+3;
        y1 = x1 + y1*x1;
    endif
endfunc

func main()
    var x1: int
    x1 = 0;
    if x1 == 1 then
        f1();
    endif
    x1 = 4*5+6;
    write x1;
    write "\n";
endfunc
```

```
func f1()
    var x1: int
    var y1: int
    write "err!!\n";
    if x1 == y1*2 then
        x1 = y1+3;
        y1 = x1 + y1*x1;
    endif
endfunc

func main()
    var x1: int
    x1 = 0;
    if x1 == 1 then
        f1();
    endif
    x1 = 4*5+6;
    write x1;
    write "\n";
endfunc
```

```
function f1
  vars
    x1 integer
    y1 integer
  endvars


      writes "err!!\n"
      %1 = 2
      %2 = y1 * %1
      %3 = x1 == %2
      ifFalse %3 goto endif1
      %4 = 3
      %5 = y1 + %4
      x1 = %5
      %6 = y1 * x1
      %7 = x1 + %6
      y1 = %7
  label endif1 :
      return
endfunction
```

```
func f1()
    var x1: int
    var y1: int
    write "err!!\n";
    if x1 == y1*2 then
        x1 = y1+3;
        y1 = x1 + y1*x1;
    endif
endfunc

func main()
    var x1: int
    x1 = 0;
    if x1 == 1 then
        f1();
    endif
    x1 = 4*5+6;
    write x1;
    write "\n";
endfunc
```

```
function f1
  vars
    x1 integer
    y1 integer
  endvars

     writes "err!!\n"
     %1 = 2
     %2 = y1 * %1
     %3 = x1 == %2
     ifFalse %3 goto endif1
     %4 = 3
     %5 = y1 + %4
     x1 = %5
     %6 = y1 * x1
     %7 = x1 + %6
     y1 = %7
  label endif1 :
     return
endfunction
```

```
function main
  vars
    x1 integer
  endvars

     %1 = 0
     x1 = %1
     %2 = 1
     %3 = x1 == %2
     ifFalse %3 goto endif1
     call f1
  label endif1 :
     %4 = 4
     %5 = 5
     %6 = %4 * %5
     %7 = 6
     %8 = %6 + %7
     x1 = %8
     writei x1
     writes "\n"
     return
endfunction
```

jpbasic_genc_01.asl.c:

```
#include <stdio.h>

void f1() {
  int x1;
  int y1;
  printf("err!!\n");
  if (x1 == y1*2) {
    x1 = y1+3;
    y1 = x1 + y1*x1;
  }
}

int main() {
  int x1;
  x1 = 0;
  if (x1 == 1) {
    f1();
  }
  x1 = 4*5+6;
  printf("%d", x1);
  printf("\n");
}
```

```
$ clang  -S  -emit-llvm  -fno-discard-value-names              \
                         -O0  -Xclang=-disable-O0-optnone  \
                         jpbasic_genc_01.asl.c  -o -
```

```
func f1()
    var x1: int
    var y1: int
    write "err!!\n";
    if x1 == y1*2 then
        x1 = y1+3;
        y1 = x1 + y1*x1;
    endif
endfunc
```

```
function f1
  vars
    x1 integer
    y1 integer
  endvars

    writes "err!!\n"
    %1 = 2
    %2 = y1 * %1
    %3 = x1 == %2
    ifFalse %3 goto endif1
    %4 = 3
    %5 = y1 + %4
    x1 = %5
    %6 = y1 * x1
    %7 = x1 + %6
    y1 = %7
  label endif1 :
    return
endfunction
```

```
declare i32 @printf(i8*, ...)

@.str = constant [7 x i8] c"err!!\0A\00"

define void @f1() {
entry:
  %x1 = alloca i32
  %y1 = alloca i32
  %0 = load i32, i32* %x1
  %1 = load i32, i32* %y1
  %call = call i32 (i8*, ...) @printf(i8* @.str, ..., i64 0))
  %mul = mul nsw i32 %1, 2
  %cmp = icmp eq i32 %0, %mul
  br i1 %cmp, label %if.then, label %if.end

if.then:                                   ; preds = %entry
  %2 = load i32, i32* %y1
  %add = add nsw i32 %2, 3
  store i32 %add, i32* %x1
  %3 = load i32, i32* %x1
  %4 = load i32, i32* %y1
  %5 = load i32, i32* %x1
  %mul1 = mul nsw i32 %4, %5
  %add2 = add nsw i32 %3, %mul1
  store i32 %add2, i32* %y1
  br label %if.end

if.end:                                 ; preds = %if.then, %entry
  ret void
}
```

```
func main()
    var x1: int
    x1 = 0;
    if x1 == 1 then
        f1();
    endif
    x1 = 4*5+6;
    write x1;
    write "\n";
endfunc
```

```
function main
  vars
    x1 integer
  endvars

    %1 = 0
    x1 = %1
    %2 = 1
    %3 = x1 == %2
    ifFalse %3 goto endif1
    call f1
  label endif1 :
    %4 = 4
    %5 = 5
    %6 = %4 * %5
    %7 = 6
    %8 = %6 + %7
    x1 = %8
    writei x1
    writes "\n"
    return
endfunction
```

```
@.str.1 = constant [3 x i8] c"%d\00"
@.str.2 = constant [2 x i8] c"\0A\00"

declare i32 @printf(i8*, ...)

define i32 @main() {
entry:
  %retval = alloca i32
  %x1 = alloca i32
  store i32 0, i32* %retval
  store i32 0, i32* %x1
  %0 = load i32, i32* %x1
  %cmp = icmp eq i32 %0, 1
  br i1 %cmp, label %if.then, label %if.end

if.then:                                    ; preds = %entry
  call void @f1()
  br label %if.end

if.end:                                     ; preds = %if.then, %entry
  store i32 26, i32* %x1
  %1 = load i32, i32* %x1
  %call = call i32 (i8*, ...) @printf(i8* @.str1,..., i32 %1))
  %call1 = call i32 (i8*, ...) @printf(i8* @.str2,..., i64 0))
  %2 = load i32, i32* %retval
  ret i32 %2
}
```

jpbasic_genc_01.asl.c:

```
#include <stdio.h>

void f1() {
  int x1;
  int y1;
  printf("err!!\n");
  if (x1 == y1*2) {
    x1 = y1+3;
    y1 = x1 + y1*x1;
  }
}

int main() {
  int x1;
  x1 = 0;
  if (x1 == 1) {
    f1();
  }
  x1 = 4*5+6;
  printf("%d", x1);
  printf("\n");
}
```

```
$ clang  -S  -emit-llvm  -fno-discard-value-names  -O1  \
                              jpbasic_genc_01.asl.c  -o -
```

```
func f1()
    var x1: int
    var y1: int
    write "err!!\n";
    if x1 == y1*2 then
        x1 = y1+3;
        y1 = x1 + y1*x1;
    endif
endfunc
```

```
function f1
  vars
    x1 integer
    y1 integer
  endvars

     writes "err!!\n"
     %1 = 2
     %2 = y1 * %1
     %3 = x1 == %2
     ifFalse %3 goto endif1
     %4 =
     %5 = y1 + %4
     x1 = %5
     %6 = y1 * x1
     %7 = x1 + %6
     y1 = %7
  label endif1 :
     return
endfunction
```

```
declare i32 @printf(i8*, ...)

@.str = constant [7 x i8] c"err!!\0A\00"

define void @f1() {
entry:
  %x1 = alloca i32
  %y1 = alloca i32
  %0 = load i32, i32* %x1
  %1 = load i32, i32* %y1
  %call = call i32 (i8*, ...) @printf(i8* @.str, …, i64 0))
  %mul = mul nsw i32 %1, 2
  %cmp = icmp eq i32 %0, %mul
  br i1 %cmp, label %if.then, label %if.end

if.then:                                  ; preds = %entry
  %2 = load i32, i32* %y1
  %add = add nsw i32 %2, 3
  store i32 %add, i32* %x1
  %3 = load i32, i32* %x1
  %4 = load i32, i32* %y1
  %5 = load i32, i32* %x1
  %mul1 = mul nsw i32 %4, %5
  %add2 = add nsw i32 %3, %mul1
  store i32 %add2, i32* %y1
  br label %if.end

if.end:                               ; preds = %if.then, %entry
  ret void
}
```

```
func f1()
    var x1: int
    var y1: int
    write "err!!\n";
    if x1 == y1*2 then
        x1 = y1+3;
        y1 = x1 + y1*x1;
    endif
endfunc
```

```
function f1
  vars
    x1 integer
    y1 integer
  endvars

     writes "err!!\n"
     %1 = 2
     %2 = y1 * %1
     %3 = x1 == %2
     ifFalse %3 goto endif1
     %4 =
     %5 = y1 + %4
     x1 = %5
     %6 = y1 * x1
     %7 = x1 + %6
     y1 = %7
  label endif1 :
     return
endfunction
```

```
declare i32 @printf(i8*, ...)

@.str = constant [7 x i8] c"err!!\0A\00"

define void @f1() {
entry:
  %call = call i32 (i8*, ...) @printf(i8* @.str, …, i64 0))
  ret void
}
```

```
func main()
    var x1: int
    x1 = 0;
    if x1 == 1 then
        f1();
    endif
    x1 = 4*5+6;
    write x1;
    write "\n";
endfunc
```

```
function main
  vars
    x1 integer
  endvars

    %1 = 0
    x1 = %1
    %2 = 1
    %3 = x1 == %2
    ifFalse %3 goto endif1
    call f1
  label endif1 :
    %4 = 4
    %5 = 5
    %6 = %4 * %5
    %7 = 6
    %8 = %6 + %7
    x1 = %8
    writei x1
    writes "\n"
    return
endfunction
```

```
@.str.1 = constant [3 x i8] c"%d\00"
@.str.2 = constant [2 x i8] c"\0A\00"

declare i32 @printf(i8*, ...)

define i32 @main() {
entry:
  %retval = alloca i32
  %x1 = alloca i32
  store i32 0, i32* %retval
  store i32 0, i32* %x1
  %0 = load i32, i32* %x1
  %cmp = icmp eq i32 %0, 1
  br i1 %cmp, label %if.then, label %if.end

if.then:                                        ; preds = %entry
  call void @f1()
  br label %if.end

if.end:                                         ; preds = %if.then, %entry
  store i32 26, i32* %x1
  %1 = load i32, i32* %x1
  %call = call i32 (i8*, ...) @printf(i8* @.str1,..., i32 %1))
  %call1 = call i32 (i8*, ...) @printf(i8* @.str2,..., i64 0))
  %2 = load i32, i32* %retval
  ret i32 %2
}
```

```
func main()
    var x1: int
    x1 = 0;
    if x1 == 1 then
        f1();
    endif
    x1 = 4*5+6;
    write x1;
    write "\n";
endfunc
```

```
function main
  vars
    x1 integer
  endvars

     %1 = 0
     x1 = %1
     %2 = 1
     %3 = x1 == %2
     ifFalse %3 goto endif1
     call f1
  label endif1 :
     %4 = 4
     %5 = 5
     %6 = %4 * %5
     %7 = 6
     %8 = %6 + %7
     x1 = %8
     writei x1
     writes "\n"
     return
endfunction
```

```
@.str.2 = constant [2 x i8] c"\0A\00"

declare i32 @printf(i8*, …)
declare i32 @putchar(i32)

define i32 @main() {
entry:
  %call1 = call i32 (i8*, ...) @printf(i8* @.str2,..., i32 26))
  %putchar = call i32 @putchar(i32 10)
  ret i32 0
}
```

jpbasic_genc_01.asl:

```
func f1()
    var x1: int
    var y1: int
    write "err!!\n";
    if x1 == y1*2 then
        x1 = y1+3;
        y1 = x1 + y1*x1;
    endif
endfunc

func main()
    var x1: int
    x1 = 0;
    if x1 == 1 then
        f1();
    endif
    x1 = 4*5+6;
    write x1;
    write "\n";
endfunc
```

asl/main.cpp:

```
...
// uncomment the following lines to generate LLVM code
// and write it to a .ll file
  std::string llvmStr = mycode.dumpLLVM(types, symbols);
  std::string llvmFileName;
  if (argc == 2) { // read from <file>
    std::string inputFileName = std::string(argv[1]);
    std::size_t slashPos = inputFileName.rfind("/");
    std::size_t dotPos    = inputFileName.rfind(".");
    llvmFileName = inputFileName.substr(slashPos+1, dotPos-slashPos-1) + ".ll";
  }
  else {           // read fron std::cin
    llvmFileName = "output.ll";
  }
  std::ofstream myLLVMFile(llvmFileName, std::ofstream::out);
  myLLVMFile << llvmStr << std::endl;
...
```

```
$ ./asl  jpbasic_genc_01.asl
```

→ jpbasic_genc_01.**ll**

Optimització de LLVM

Passes d'anàlisi:

```
$ opt  -S  -enable-new-pm=0  --view-cfg          jpbasic_genc_01.ll

$ opt  -S  -enable-new-pm=0  --view-callgraph  jpbasic_genc_01.ll
```

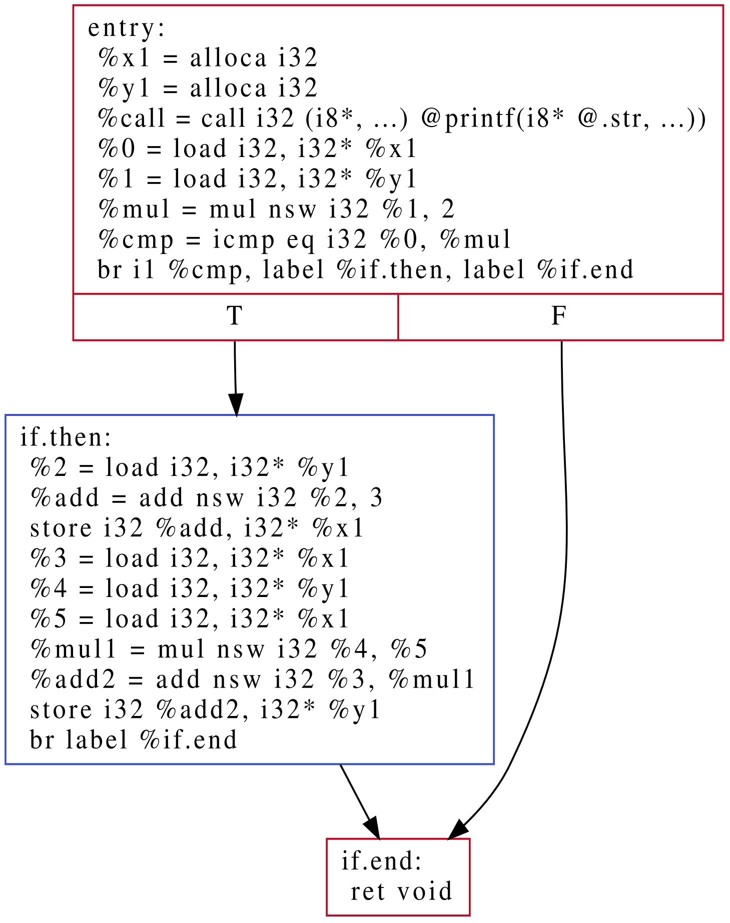
Passes de transformació:

```
$ opt  -S  -enable-new-pm=0  --mem2reg  --sccp  --adce  jpbasic_genc_01.ll
```

+info: https://llvm.org/docs/Passes.html

Passes d'anàlisi:

```
$ opt  -S  -enable-new-pm=0  --view-cfg   \
             jpbasic_genc_01.ll
```



```
entry:
 %x1 = alloca i32
 %y1 = alloca i32
 %call = call i32 (i8*, ...) @printf(i8* @.str, ...))
 %0 = load i32, i32* %x1
 %1 = load i32, i32* %y1
 %mul = mul nsw i32 %1, 2
 %cmp = icmp eq i32 %0, %mul
 br i1 %cmp, label %if.then, label %if.end
```

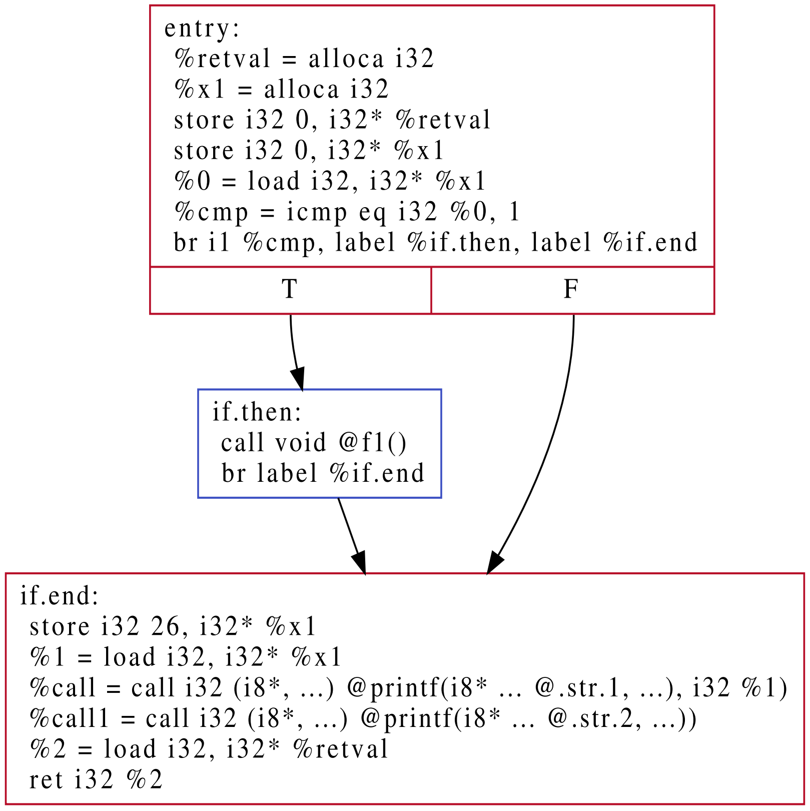
```
if.then:
 %2 = load i32, i32* %y1
 %add = add nsw i32 %2, 3
 store i32 %add, i32* %x1
 %3 = load i32, i32* %x1
 %4 = load i32, i32* %y1
 %5 = load i32, i32* %x1
 %mul1 = mul nsw i32 %4, %5
 %add2 = add nsw i32 %3, %mul1
 store i32 %add2, i32* %y1
 br label %if.end
```

```
if.end:
 ret void
```

CFG for 'f1' function

Passes d’anàlisi:

```
$ opt  -S  -enable-new-pm=0  --view-cfg   \
           jpbasic_genc_01.ll
```



```
entry:
 %retval = alloca i32
 %x1 = alloca i32
 store i32 0, i32* %retval
 store i32 0, i32* %x1
 %0 = load i32, i32* %x1
 %cmp = icmp eq i32 %0, 1
 br i1 %cmp, label %if.then, label %if.end
```

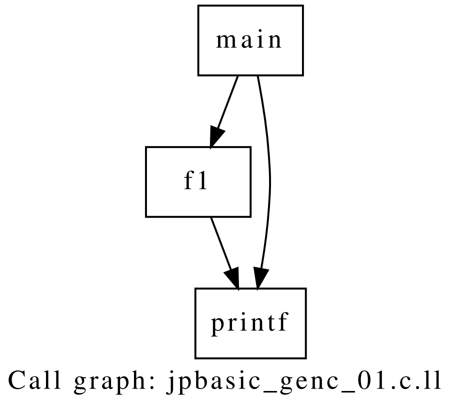
```
if.then:
 call void @f1()
 br label %if.end
```

```
if.end:
 store i32 26, i32* %x1
 %1 = load i32, i32* %x1
 %call = call i32 (i8*, ...) @printf(i8* ... @.str.1, ...), i32 %1)
 %call1 = call i32 (i8*, ...) @printf(i8* ... @.str.2, ...))
 %2 = load i32, i32* %retval
 ret i32 %2
```

CFG for ’main’ function

Passes d'anàlisi:

```
$ opt  -S  -enable-new-pm=0  --view-callgraph   \
        jpbasic_genc_01.ll
```



Passes de transformació:

```
$ opt  -S  -enable-new-pm=0        \
           --mem2reg               \
           jpbasic_genc_01.ll
```

Altres passes:

```
--adce
--sccp
--simplifycfg
--instcombine
--licm
--loop-reduce
...
```

```
$ opt  --help
$ opt  --print-passes
```

```
@.str = constant [7 x i8] c"err!!\0A\00"
@.str.1 = constant [3 x i8] c"%d\00"
@.str.2 = constant [2 x i8] c"\0A\00"

declare i32 @printf(i8*, ...)

define void @f1() {
entry:
  %call = call i32 (i8*, ...) @printf(i8* ... @.str, ...))
  %mul = mul nsw i32 undef, 2
  %cmp = icmp eq i32 undef, %mul
  br i1 %cmp, label %if.then, label %if.end

if.then:                                  ; preds = %entry
  %add = add nsw i32 undef, 3
  %mul1 = mul nsw i32 undef, %add
  %add2 = add nsw i32 %add, %mul1
  br label %if.end

if.end:                                   ; preds = %if.then, %entry
  ret void
}

define i32 @main() {
entry:
  %cmp = icmp eq i32 0, 1
  br i1 %cmp, label %if.then, label %if.end

if.then:                                  ; preds = %entry
  call void @f1()
  br label %if.end

if.end:                                   ; preds = %if.then, %entry
  %call = call i32 (i8*, ...) @printf(i8* ... @.str.1, ...), i32 noundef 26)
  %call1 = call i32 (i8*, ...) @printf(i8* ...@.str.2, ...))
  ret i32 0
}
```

Passes de transformació:

```
$ opt  -S  -enable-new-pm=0          \
       --mem2reg  --adce   \
       jpbasic_genc_01.ll
```

```
$ opt  --help
$ opt  --print-passes
```

Altres passes:

```
--sccp
--simplifycfg
--instcombine
--licm
--loop-reduce
...
```

```
@.str = constant [7 x i8] c"err!!\0A\00"
@.str.1 = constant [3 x i8] c"%d\00"
@.str.2 = constant [2 x i8] c"\0A\00"

declare i32 @printf(i8*, ...)

define void @f1() {
entry:
  %call = call i32 (i8*, ...) @printf(i8* ... @.str, ...))
  br label %if.end

if.then:                              ; No predecessors!
  br label %if.end

if.end:                               ; preds = %entry, %if.then
  ret void
}

define i32 @main() {
entry:
  %cmp = icmp eq i32 0, 1
  br i1 %cmp, label %if.then, label %if.end

if.then:                              ; preds = %entry
  call void @f1()
  br label %if.end

f.end:                                ; preds = %entry, %if.then
  %call = call i32 (i8*, ...) @printf(i8* ... @.str.1, ...), i32 26)
  %call1 = call i32 (i8*, ...) @printf(i8* ... @.str.2, ...))
  ret i32 0
}
```

Passes de transformació:

```
$ opt  -S  -enable-new-pm=0          \
           --mem2reg  --sccp  --adce  \
           jpbasic_genc_01.ll
```

```
$ opt  --help
$ opt  --print-passes
```

Altres passes:

```
--simplifycfg
--instcombine
--licm
--loop-reduce
...
```

```
@.str = constant [7 x i8] c"err!!\0A\00"
@.str.1 = constant [3 x i8] c"%d\00"
@.str.2 = constant [2 x i8] c"\0A\00"

declare i32 @printf(i8*, ...)

define void @f1() {
entry:
  %call = call i32 (i8*, ...) @printf(i8* ... @.str, ...))
  br label %if.end

if.then:                              ; No predecessors!
  br label %if.end

if.end:                               ; preds = %entry, %if.then
  ret void
}

define i32 @main() {
entry:
  br label %if.end

if.then:                             ; No predecessors!
  br label %if.end

if.end:                              ; preds = %entry, %if.then
  %call = call i32 (i8*, ...) @printf(i8* ... @.str.1, ...), i32 26)
  %call1 = call i32 (i8*, ...) @printf(i8* ... @.str.2, ...))
  ret i32 0
}
```

Passes de transformació:

```
$ opt  -S  -enable-new-pm=0          \
           --mem2reg  --sccp  --adce  \
           jpbasic_genc_01.ll
```

Altres passes:

```
--simplifycfg
--instcombine
--licm
--loop-reduce
...
```

```
@.str = constant [7 x i8] c"err!!\0A\00"
@.str.1 = constant [3 x i8] c"%d\00"
@.str.2 = constant [2 x i8] c"\0A\00"

declare i32 @printf(i8*, ...)

define void @f1() {
entry:
  %call = call i32 (i8*, ...) @printf(i8* ... @.str, ...))
  br label %if.end

if.then:                                ; No predecessors!
  br label %if.end

if.end:                                 ; preds = %entry, %if.then
  ret void
}

define i32 @main() {
entry:
  br label %if.end

if.then:                               ; No predecessors!
  br label %if.end

if.end:                                ; preds = %entry, %if.then
  %call = call i32 (i8*, ...) @printf(i8* ... @.str.1, ...), i32 26)
  %call1 = call i32 (i8*, ...) @printf(i8* ... @.str.2, ...))
  ret i32 0
}
```

Passes de transformació:

```
$ opt  -S  -enable-new-pm=0          \
           --mem2reg  --sccp  --adce   \
           --simplifycfg
           jpbasic_genc_01.ll
```

Altres passes:

```
--instcombine
--licm
--loop-reduce
...
```

```
@.str = constant [7 x i8] c"err!!\0A\00"
@.str.1 = constant [3 x i8] c"%d\00"
@.str.2 = constant [2 x i8] c"\0A\00"

declare i32 @printf(i8*, ...)

define void @f1() {
entry:
  %call = call i32 (i8*, ...) @printf(i8* ... @.str, ...))
  ret void
}

define di32 @main() {
entry:
  %call = call i32 (i8*, ...) @printf(i8* ... @.str.1, ...), i32 26)
  %call1 = call i32 (i8*, ...) @printf(i8* ... @.str.2, ...))
  ret i32 0
}
```