

```
type node = positive
  [@@ deriving show]
```

```
type instruction =|
| Inop of node
| Iop of operation * reg list * reg * node
| Iload of memory_chunk * addressing * reg list * reg * node
| Istore of memory_chunk * addressing * reg list * reg * node
| Icall of signature * (reg, ident) sum * reg list * reg * node
| Itailcall of signature * (reg, ident) sum * reg list
| Ibuiltin of external_function * reg builtin_arg list * reg builtin_res
  * node
| Icond of condition * reg list * node * node
| Ijumptable of reg * node list
| Ireturn of reg option
  [@@ deriving show]
```

```
main() {
  9: x3 = 5
  8: x2 = 3
  7: x1 = "add"(x3, x2)
  6: if (x1 <s x3) goto 4 else goto 5
  5: x2 = x2 + 1 (int)
    goto 3
  4: x2 = (- x2)
  3: x4 = 0
    goto 1
  2: x4 = 0
  1: return x4
}
```

Une opération

Le type de l'opération (int)

Type d'opération +

"1" pour +1

"2" pour x2

"2" pour x2

"3" pour l'instruction suivante 3

```
(Some (RTL.Iop (
  (Op.Olea
    (Op.Aindexed (BinNums.Zpos BinNums.Coq_xH))
    [(BinNums.Coq_x0 BinNums.Coq_xH)],
    (BinNums.Coq_x0 BinNums.Coq_xH),
    (BinNums.Coq_xI BinNums.Coq_xH))),
  Maps.PTree.Leaf)),
```