リスト 1　dbgScaner.rb

```
# result
# 25,!18,30 | llvm行, メタ変数, C行

load("loadFile.rb")
load("relativeC.rb")

$WRITE = "./result/gdb.txt"

write_file = open($WRITE, "w")
read_file = open($FileName+".ll", "r")

read_file.each_line do |line| #1行単位で読み込み
  flag = 0
  token = line.split(/[ ,(\n)]/)
  token.each_with_index do |t, i|
    if t =~ /!dbg/
     flag = 1
     next
    end
    # p t
    if flag == 1
      print "dbg #{t}:#{read_file.lineno}\n"

      read_gen = open("./result/line.txt", "r")
      read_gen.each_line do |line|
        token = line.split(/[,(\n)]/)
        token.each {|t1| write_file.print "#{read_file.lineno},#{line.chomp}\n" if t == t1}
      end
      read_gen.close()

      flag = 0
    end
  end #end token
end # end each other line

read_file.close()
write_file.close()
```

リスト 2　grid.rb

```
class Grid
  def initialize
    @ret = nil
    @array = {}
  end
  attr_accessor :ret, :array

  def InstrGroup(this, thisFunction, thisLabel, thisLine)
    buf = ""
    count = 0
    3.times do |i|
      if this != nil && this.Function == thisFunction && this.Label == thisLabel
        buf = buf + "#{this.Opecode}"
        # print "#{this.Opecode}"
        this = this.NextLabel
        count += 1
      end
    end
    # print "\n"
    if count == 3 then
      # print "#{buf}\n"
      array[buf] = thisLine
      # print ",#{array[buf]}"
      write_file = open("./result/instr.txt", "a")
      write_file.print "#{buf},#{thisLine}\n"
      write_file.close()
    end
  end # end InstrGroup

  def BasicGroup(this, thisLabel, thisLine)
    buf = ""
      loop{
      if this != nil && this.Label == thisLabel then
        print this.Opecode
        buf = buf + "#{this.Opecode}"
        this = this.NextLabel
      else
        ret = this
          # if this != nil && this.Opecode == "ret"
            print "\n";
          # end
        break;
      end

      write_file = open("./result/instr.txt", "a")
      write_file.print "#{buf},#{thisLine}\n"
      write_file.close

      # print "\n";

```

```
        }
        ret = this
        return ret
    end # end BasicGroup

    def FunctionGroup(this, thisFunction)
        loop{
            if this != nil && this.Function == thisFunction then
                print this.Opecode
                this = this.NextLabel
            else
                ret = this
                print "\n";
                break;
            end
        }
        return ret
    end # end FuntionGroup

end # end Class
```

リスト 3　iden.rb

```
# 予約命令
$iden = Array.new;
# $iden << "define"

#Terminator Instructions
# $iden << "ret"
$iden << "switch"
# $iden << "br"
$iden << "indirectbr"
$iden << "invoke"
$iden << "resume"
$iden << "catchpad"
$iden << "catchendpad"
$iden << "catchret"
$iden << "cleanupendpad"
$iden << "cleanupret"
$iden << "terminatepad"
$iden << "unreachable"

# Binary Operations
$iden << "add"
$iden << "fadd"
$iden << "sub"
$iden << "fsub"
$iden << "mul"
$iden << "fmul"
$iden << "udiv"
$iden << "sdiv"
$iden << "fdiv"
$iden << "urem"
$iden << "srem"
$iden << "frem"

# Bitwise Binary Operations
$iden << "shl"
$iden << "lshr"
$iden << "ashr"
$iden << "and"
$iden << "or"
$iden << "xor"

# Vector Operations
$iden << "extractelement"
$iden << "insertelement"
$iden << "shufflevector"

# Aggregate Operations
$iden << "extractvalue"
$iden << "insertvalue"
$iden << "insertvalue"

#Memory Access and Addressing Operations
# $iden << "alloca"
$iden << "load"
$iden << "store"
$iden << "fence"
$iden << "cmpxchg"
$iden << "atomicrmw"
$iden << "getelementptr"

# Conversion Operations
$iden << "trunc" #to
$iden << "zext" #to
$iden << "sext" #to
$iden << "fptrunc" #to
$iden << "fpext" #to
$iden << "fptoui" #to
$iden << "fptosi" #to
```

```
$iden << ”uitofp” #to
$iden << ”sitofp” #to
$iden << ”ptrtoint” #to
$iden << ”inttoptr” #to
$iden << ”bitcast” #to
$iden << ”addrspacecast” #to

# Other Operations
$iden << ”icmp”
$iden << ”fcmp”
$iden << ”phi”
$iden << ”select”
$iden << ”call”
$iden << ”va_arg”
$iden << ”landingpad”
$iden << ”cleanuppad”
```

リスト 4　list.rb

```
# リスト構造
class List
  def initialize
    @Function = nil # 所属関数
    @Label = nil # 所属ラベル
    @Opecode = nil # 命令名
    @Line = nil # 行番号
    @NextLabel = nil # 次のラベル
    @OpecodetoArray = Array.new(GROUP_COUNT)
  end

  attr_accessor :Function, :Label, :Opecode, :Line, :NextLabel, :OpecodetoArray; #インスタンス変数の参照や更新

  def toArray(this, i, opecode)
    this.OpecodetoArray[i] = opecode
    print ”toArray:#{this.OpecodetoArray[i]}\n”
  end

  def add_last(function, label, opecode, line)
    this = self
    this = this.NextLabel until this.NextLabel.nil?
    this.NextLabel = List.new
    this.NextLabel.Function = function
    this.NextLabel.Label = label
    this.NextLabel.Opecode = opecode
    this.NextLabel.Line = line
  end

  def size
    this = self
    i = 0
    i += 1 while this = this.NextLabel
    return i
  end

  def each
    this = self.NextLabel
    self.size.times do
      # for var in this.OpecodetoArray do
        # print(Kconv.tosjis(”Color = ” + var + ”¥n”))
       # yield ”#{var}”
      # end
      # this.OpecodetoArray.each{ |var|
       # yield ”#{var}”
      # }
      # yield ”#{this.OpecodetoArray[0]}: ”
      # yield ”#{this.OpecodetoArray[1]}: ”
      # yield ”#{this.OpecodetoArray[2]}: ”
      yield ”#{this.Line}: ”
      yield ”#{this.Function}: ”
      yield ”#{this.Label}: ”
      yield ”#{this.Opecode}\n”
      this = this.NextLabel
    end
  end

  def cat(nextlabel = self.NextLabel)
    this = nextlabel
    thisFunction = this.Function
    thisLabel = this.Label
    thisLine = this.Line
    join = ’’
    # var_p(thisLabel, ””)
    ret = this.NextLabel

    grid = Grid.new
    ## BasicBlockの中のInstructionを3つに束ねグループ化
    grid.InstrGroup(this, thisFunction, thisLabel, thisLine)

    ## BasicBlockごとに分割
    # grid.BasicGroup(this, thisLabel, thisLine)

    ## functionごとに分割
    # grid.FunctionGroup(this, thisFunction)
    return ret
  end

end #class List
```

リスト 5　read.rb

```
GROUP_COUNT = 3
GROUP_COUNT.freeze

load("loadFile.rb")
load("iden.rb")
load("method.rb")
load("grid.rb")
load("list.rb")
system ("rm ./result/instr.txt")


# −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
# Main処理
function = "main"
label = 1
line = 1
opecode = "define"

token = Array.new
opelist = List.new

file = open($FileName+".ll", "r:utf−8")

file.each_line do |line| #1行単位で読み込み
  # puts "#{lineage}:#{line}" if line.include?("alloca")
  token = line.split(" ")
  token.each_with_index do |t, i|
    if t == "define"
      # p token[i+2].scan(/\@[a−zA−Z]+[a−zA−Z0−9]*/)
      label = 1
      function = token[i+2].scan(/\@[a−zA−Z_]+[_a−zA−Z0−9]*/)
      # p function[0]
    end
    label = t.scan(/[0−9]+/) if t =~ /<label>/
    # if t =~ /<label>/
      # label = t.scan(/[0−9]+/)
    # end
    # print "#{file.lineno} #{t}\n" if $iden.include?(t)
    if $iden.include?(t)
      # print "#{function[0]}: #{label[0]}: #{file.lineno} #{t}\n"
      opelist.add_last(function[0], label[0], t, file.lineno)
    end
  end #end token[]
end # end each other line


label = opelist.cat
while label != nil do
  label = opelist.cat(label)
end


# 表示
# opelist.each {|i| print i}

# −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

=begin
ruby read.rb | sort | uniq −c| sort −r

$ ruby read.rb | sort | uniq −c| sort

=end
```

リスト 6　test.rb

```
=begin
# 付属ブロックの有無で動作変更
def call_block2()
  if block_given?
    yield
  else
    puts "ブロックが付属されていません"
  end
end
# 実行
call_block2 { puts "ブロック内部" } #=> ブロック内部
call_block2() #=> ブロックが付属されていません
=end

buf = "aa"
```

```
scores = {} # 空のハッシュを作成
scores[buf] = ”zz” # キー”Alice”、値80のペアを追加
p scores[buf] # キー”Alice”の値を取り出し
                                # 3つのキー＋値からなるハッシュを作成
user = { :name => ”k−sato”, :email => ”k−sato@foo.xx.jp”,
         :address => ”Tokyo” }
p user[:name] # キー:nameの値を取り出し

class Foo
    @name
    def name
        @name
    end
    def name=(value) # 呼び出しは‘~.name = ”hogs”‘でOK
        @name = value
    end
end

obj = Foo.new()
p obj.name = ”zzz”
```

リスト 7　method.rb

```
# 配列の内容を表示 print_r(array);
def print_r(*array)
  array.each do |element|
    puts element
  end
end

def var_p(var, split)
  print sprintf(”%15s #{split}”, var)
end
```

リスト 8　loadFile.rb

```
# 対象ファイル
# $FileName = ”/Users/hiro/Program/C/common/if/A”
# $FileName = ”/Users/hiro/Program/C/common/game/gomoku/game”
$FileName = ”/Users/hiro/Program/C/Sample/Sample3/if−switchA”
$FileName = ”/Users/hiro/Program/C/Sample/Sample1/matricsA”
# $FileName = ”/Users/hiro/Program/C/Sample/Sample2/if−elseA”
# $FileName = ”/Users/hiro/Program/seminarl/C/BMP_Creater/Create”
# $FileName = ”/Users/hiro/Program/C/common/test/lisp”
```

リスト 9　relativeC.rb

```
load(”loadFile.rb”)

$WRITE = ”./result/line.txt”

$FILELINE = ‘grep ’’ #{$FileName}.c | wc −l‘.to_i # 行数数え

write_file = open($WRITE, ”w”)
read_file = open($FileName+”.ll”, ”r”)

read_file.each_line do |line| #1行単位で読み込み
  next unless line =~ /\A!/

  token = line.split(/[ ,]/)
  buf = ””
  token.each_with_index do |t, i|
    # next unless t =~ /\A![0−9]/
    if i == 0 && t =~ /\A![0−9]/
      buf = t
      next
    end
    if i == 4 && t.to_i <= $FILELINE && buf != ””
      print ”to : #{t.to_i} : #{buf}\n”;
      write_file.print ”#{buf},#{t.to_i}\n”;
      buf = nil
    end
  end #end token
end # end each other line

read_file.close()
write_file.close()

=begin
llvm, C
!1,0
!2,0
!3,0
!7,2
!8,2
!9,0
!10,4
!11,6
!13,7
```

```
!15,8
!16,8
!18,9
!20,10
!21,11
!23,13

=end
```

リスト 10　summarize.rb

```
load("read.rb")
instr_file = open("./result/instr.txt", "r")

def relativeMeta(num)
   buf = ""
   gdb_file = open("./result/gdb.txt", "r")
   gdb_file.each_line do |gdb_line|
      token = gdb_line.split(/[ ,(\n)]/)
      if token[0] == num
         buf = token[1]
      end
   end
   gdb_file.close()
   return buf
end

def relativeC(num)
   buf = ""
   gdb_file = open("./result/gdb.txt", "r")
   gdb_file.each_line do |gdb_line|
      token = gdb_line.split(/[ ,(\n)]/)
      if token[0] == num
         buf = token[2]
      end
   end
   gdb_file.close()
   return buf
end


instr_list = {}
line_list = {}
instr_file.each_line do |instr_line|
   token = instr_line.split(/[ ,(\n)]/)

   if instr_list.has_key?(token[0])
      instr_list[token[0]] += 1
      line_list[token[0]] = line_list[token[0]] + "," +token[1] # llvm
      # line_list[token[0]] = line_list[token[0]] + "," +relativeMeta(token[1]) # meta
      # line_list[token[0]] = line_list[token[0]] + "," +relativeC(token[1]) # C
      next
   end
   instr_list[token[0]] = 1
    line_list[token[0]] = token[1] # llvm
    # line_list[token[0]] = relativeMeta(token[1]) # meta
    # line_list[token[0]] = relativeC(token[1]) # C
end
instr_file.close()

# p instr_list.sort {|(k1, v1), (k2, v2)| v2 <=> v1 }

instr_file = open("./result/instr.txt", "r")
instr_file.each_line do |instr_line|
   token = instr_line.split(/[ ,(\n)]/)
   token[0]
end
instr_file.close()

for var in instr_list.sort{|(k1, v1), (k2, v2)| v2 <=> v1 } do
   print "#{var} #{line_list[var[0]]} \n"

   # print "#{var}"
   # gdb_file = open("./result/dbg.txt", "r")
   # gdb_file.each_line do |gdb_line|
   # token = gdb_line.split(/[ ,(\n)]/)
   # token[0]== line_list[var[0]]
   # gdb_file.close()
end



=begin
for var in line_list do
   print "line = #{var} \n"
end
=end
```