v8中LinearScanRegisterAllocation的伪代码和源码分析

陆亚涵

PLCT实验室

中科院软件所

2021/4/30

例子

```
function test(a, b) {
    return a + b;
};

%PrepareFunctionForOptimization(test);
test(100,50);
%OptimizeFunctionOnNextCall(test);
test(100,50);
```

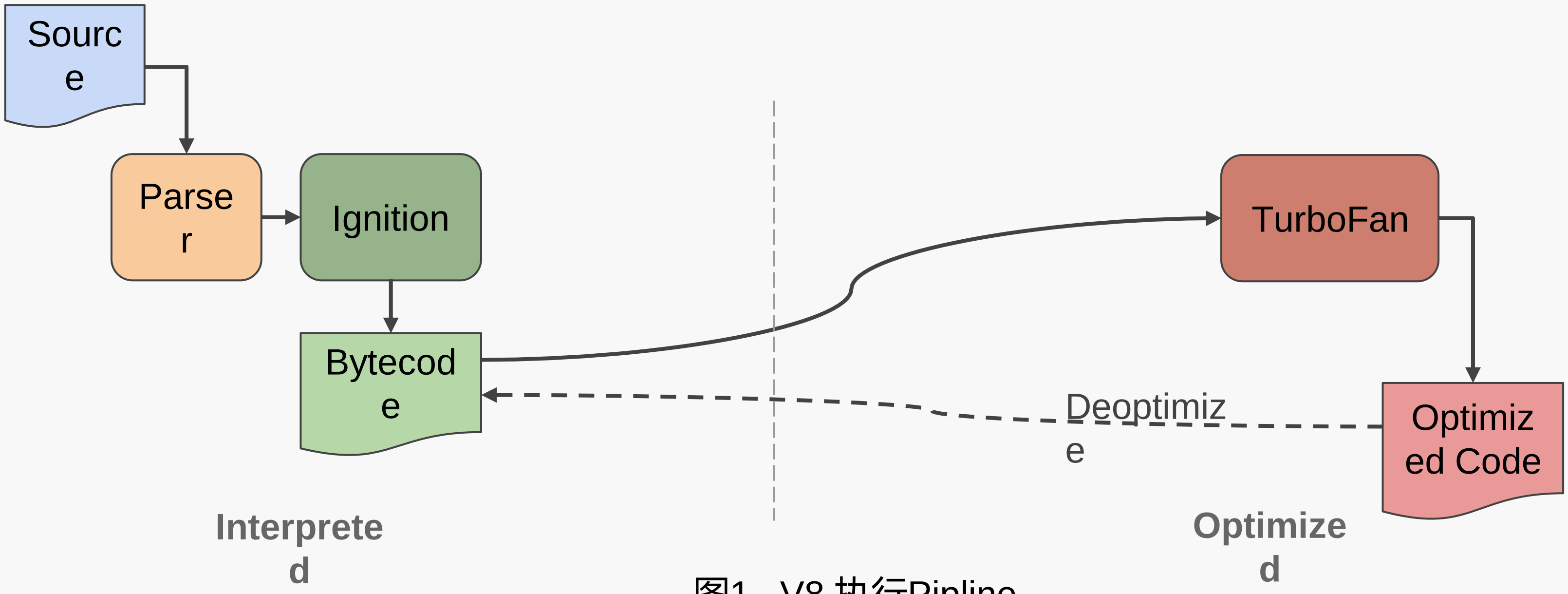


图1 V8 执行Pipline
来自 https://v8.dev/docs/ignition

例子

```
function test(a, b) {
    return a + b;
};

%PrepareFunctionForOptimization(test);
test(100,50);
%OptimizeFunctionOnNextCall(test);
test(100,50);
```

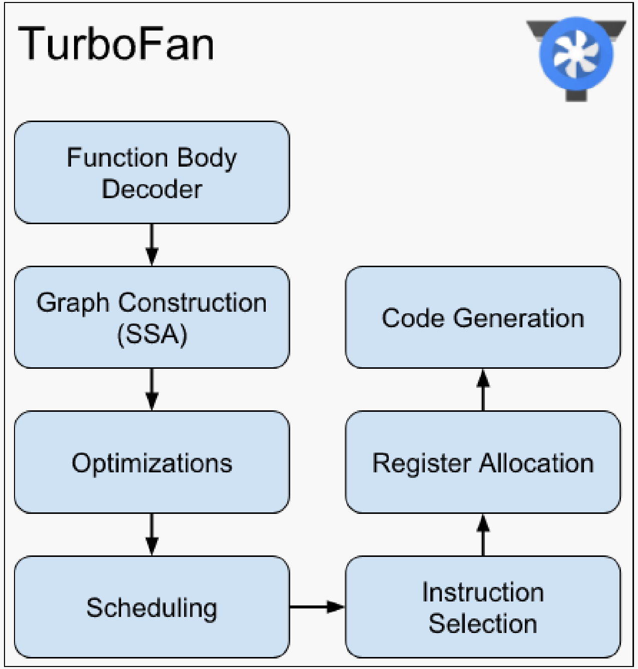


图2 Turbofan Pipline
https://v8.dev/blog/liftoff

例子

```
function test(a, b) {
    return a + b;
};

%PrepareFunctionForOptimization(test);
test(100,50);
%OptimizeFunctionOnNextCall(test);
test(100,50);
```

V8生成的opcode块

```
16: gap () ()
    RiscvTst && deoptimize if not equal v6(R) #1 #2 [immediate:2] v5(-) v6(-) v7(-) v8(S) v7(-)
17: gap () ()
    RiscvTst && deoptimize if not equal v7(R) #1 #1 [immediate:1] v5(-) v6(-) v7(-) v8(S) v7(-)
18: gap () (v4(R) = v7(-);)
    v4(0) = RiscvSar64 v4(R) #32
19: gap () (v3(R) = v6(-);)
    v3(0) = RiscvSar64 v3(R) #32
20: gap () (v2(R) = v3(-);)
    v2(0) = RiscvAdd64 && deoptimize if overflow v2(R) v4(R) #0 [immediate:0] v5(-) v6(-) v7(-) v8(S) v7(-)
21: gap () (v1 = v2(-);)
    v1(0) = ArchNop v1
22: gap () (v0(R) = v1(-);)
    v0(0) = RiscvShl64 v0(R) #32
23: gap () ([a0|R|w64] = v0(-);)
    ArchRet #0 [a0|R|w64]
```

例子

```
function test(a, b) {
    return a + b;
};

%PrepareFunctionForOptimization(test);
test(100,50);
%OptimizeFunctionOnNextCall(test);
test(100,50);
```

V8生成的opcode块

```
     16: gap ([a2|R|t] = [stack:-2|t];) ()
          RiscvTst && deoptimize if not equal [a2|R|t] #1 #2 [immediate:2] [stack:-1|t] [a2|R|t] [stack:-3|t] [stack:5|t] [stack:-3|t]
      17: gap ([a3|R|t] = [stack:-3|t];) ()
          RiscvTst && deoptimize if not equal [a3|R|t] #1 #1 [immediate:1] [stack:-1|t] [a2|R|t] [a3|R|t] [stack:5|t] [a3|R|t]
      18: gap ([a4|R|w32] = [a3|R|t];) ()
          [a4|R|w32] = RiscvSar64 [a4|R|w32] #32
      19: gap ([a5|R|w32] = [a2|R|t];) ()
          [a5|R|w32] = RiscvSar64 [a5|R|w32] #32
      20: gap () ()
          [a5|R|w64] = RiscvAdd64 && deoptimize if overflow [a5|R|w64] [a4|R|w32] #0 [immediate:0] [stack:-1|t] [a2|R|t] [a3|R|t] [stack:5|t]
[a3|R|t]
      21: gap () ()
          [a5|R|w64] = ArchNop [a5|R|w64]
      22: gap () ()
          [a5|R|w64] = RiscvShl64 [a5|R|w64] #32
      23: gap ([a0|R|w64] = [a5|R|w64];) ()
          ArchRet #0 [a0|R|w64]
```

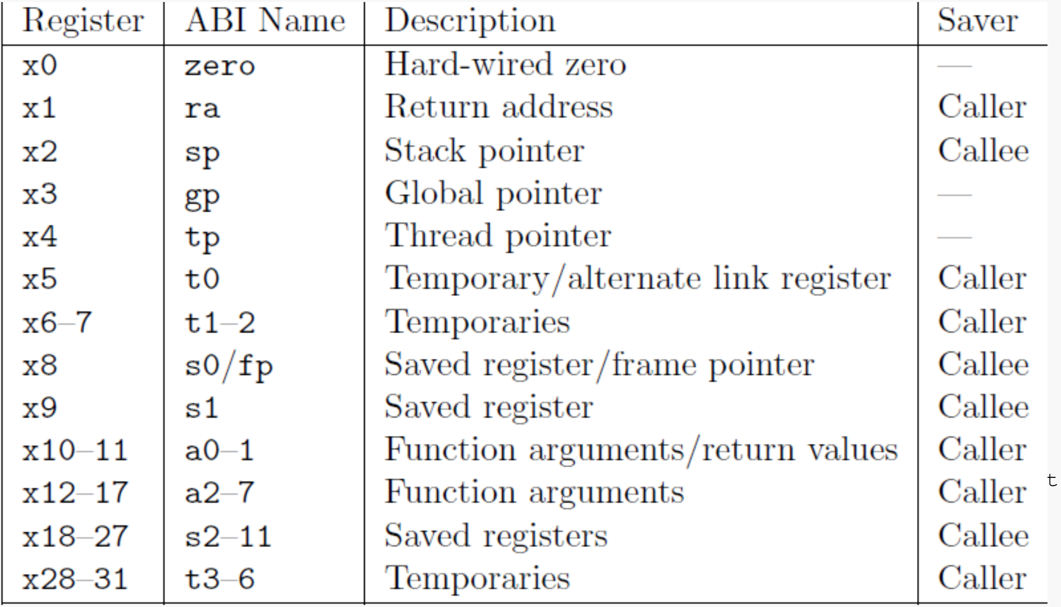
例子

```
function test(a, b) {
    return a + b;
};

%PrepareFunctionForOptimization(test);
test(100,50);
%OptimizeFunctionOnNextCall(test);
test(100,50);
```

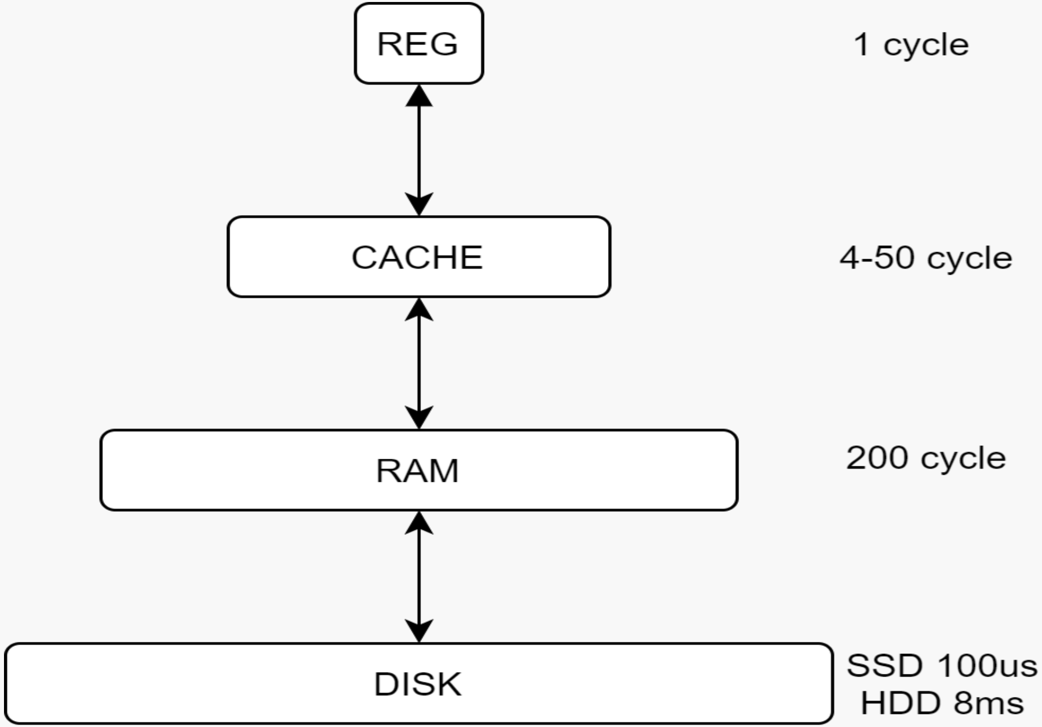
V8生成的opcode块

```
    16: gap ([a2|R|t] = [stack:-2|t];) ()
        RiscvTst && deoptimize if not equal [a2|R|t] #1
     17: gap ([a3|R|t] = [stack:-3|t];) ()
        RiscvTst && deoptimize if not equal [a3|R|t] #1
     18: gap ([a4|R|w32] = [a3|R|t];) ()
         [a4|R|w32] = RiscvSar64 [a4|R|w32] #32
     19: gap ([a5|R|w32] = [a2|R|t];) ()
         [a5|R|w32] = RiscvSar64 [a5|R|w32] #32
     20: gap () ()
         [a5|R|w64] = RiscvAdd64 && deoptimize if overfl                                                      t]
[a3|R|t]
     21: gap () ()
         [a5|R|w64] = ArchNop [a5|R|w64]
     22: gap () ()
         [a5|R|w64] = RiscvShl64 [a5|R|w64] #32
     23: gap ([a0|R|w64] = [a5|R|w64];) ()
        ArchRet #0 [a0|R|w64]
```



Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5	t0	Temporary/alternate link register	Caller
x6–7	t1–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller

计算机存储层次


REG
1 cycle
CACHE
4-50 cycle
RAM
200 cycle
DISK
SSD 100us
HDD 8ms

Global register allocation

Graph Coloring

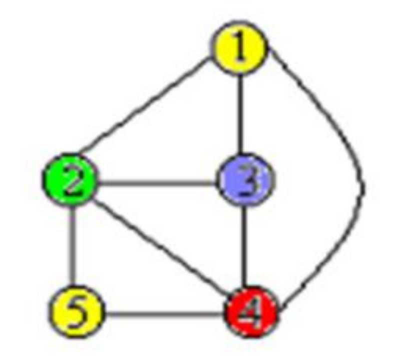


图3 图着色算法

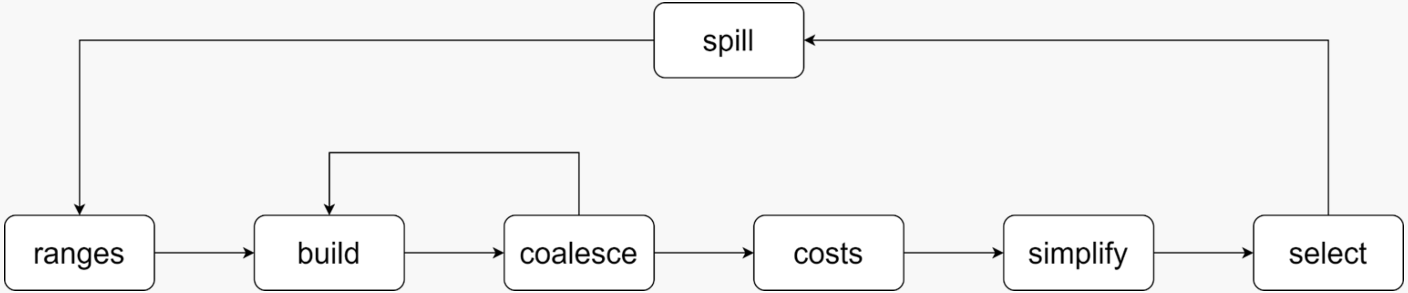
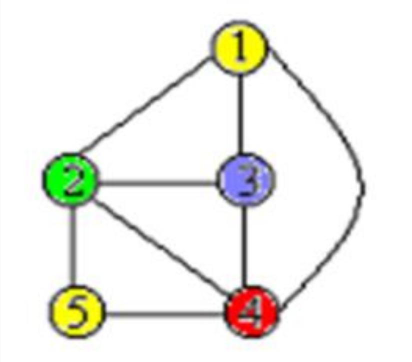


图4　图着色算法的寄存器分示意图

Global register allocation

Graph Coloring



时间复杂度O(n^2)

图3 图着色算法

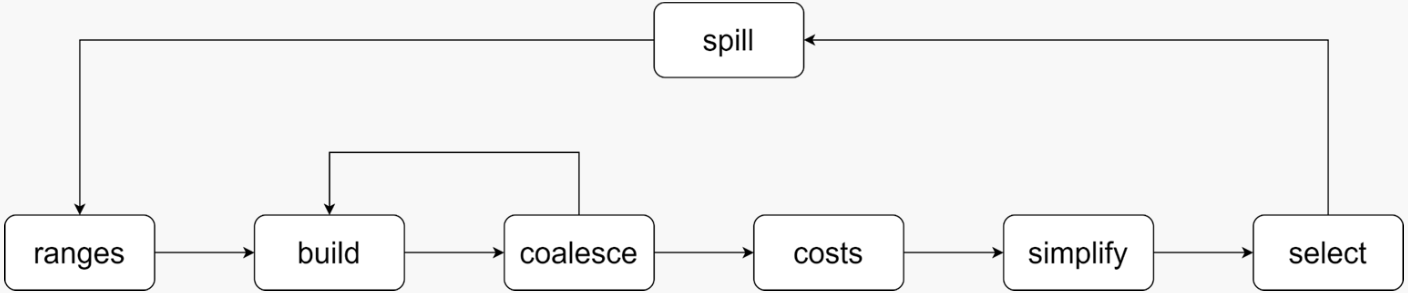
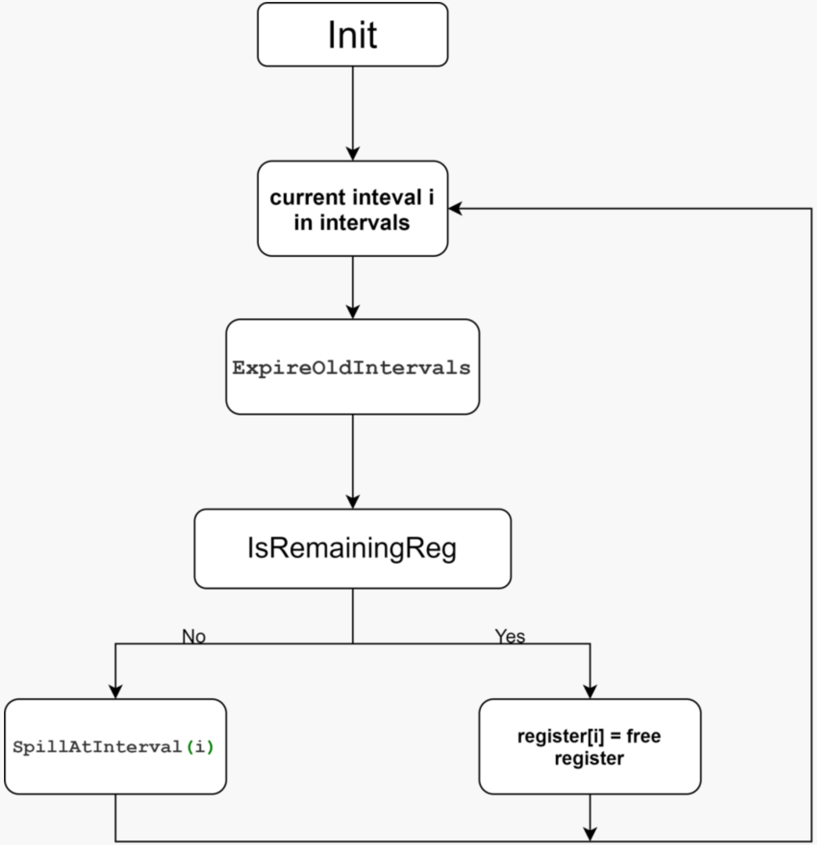


图4　图着色算法的寄存器分示意图

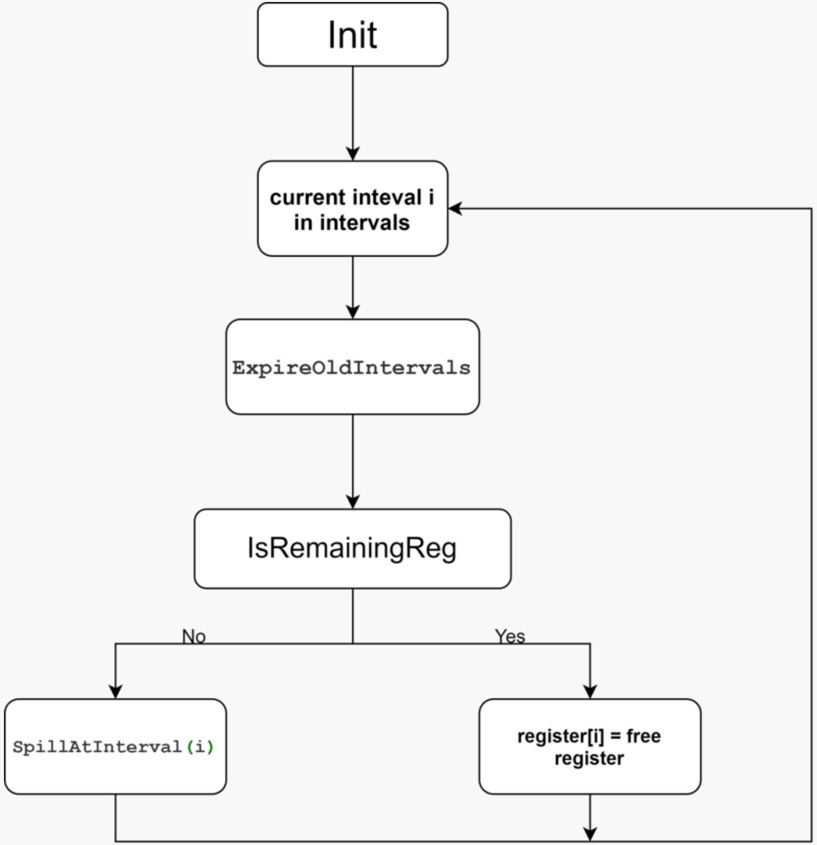
Global register allocation

Linear Scan



Global register allocation

Linear Scan



时间复杂度O(N)

LinearScanRegisterAllocation 的伪代码分析

线性扫描算法(linear scan)最早由Poletto和Sarkar[1]提出，在gcc、llvm和Java HotSpot中得到了实现。

- 简化了基于图着色的分配问题，是对一个有序的生命期序列(interval)的分配。
- 提高了寄存器分配的速度（线性速度），而没有过度降低对寄存器的利用。

[1] Poletto M, Sarkar V. Linear scan register allocation[J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 1999, 21(5): 895-913.

Linear Scan Register Allocation

```
LinearScanRegisterAllocation
   active ← {}
   for each live interval i, in order of increasing start point do
       ExpireOldIntervals(i)
       if length(active) = R then  // 如果寄存器已经分配完毕
           SpillAtInterval(i)      // spill register
       else
           // 将一个未被分配的寄存器分配给当前变量i, 并将i加入到active集合中
           register[i] ← a register removed from pool of free registers
           add i to active, sorted by increasing end point

```

Linear Scan Register Allocation

```
16: gap () ()
    RiscvTst && deoptimize if not equal v6(R) #1 #2 [immediate:2] v5(-) v6(-) v7(-) v8(S) v7(-)
17: gap () ()
    RiscvTst && deoptimize if not equal v7(R) #1 #1 [immediate:1] v5(-) v6(-) v7(-) v8(S) v7(-)
18: gap () (v4(R) = v7(-);)
    v4(0) = RiscvSar64 v4(R) #32
19: gap () (v3(R) = v6(-);)
    v3(0) = RiscvSar64 v3(R) #32
20: gap () (v2(R) = v3(-);)
    v2(0) = RiscvAdd64 && deoptimize if overflow v2(R) v4(R) #0 [immediate:0] v5(-) v6(-) v7(-) v8(S) v7(-)
21: gap () (v1 = v2(-);)
    v1(0) = ArchNop v1
22: gap () (v0(R) = v1(-);)
    v0(0) = RiscvShl64 v0(R) #32
23: gap () ([a0|R|w64] = v0(-);)
    ArchRet #0 [a0|R|w64]
```

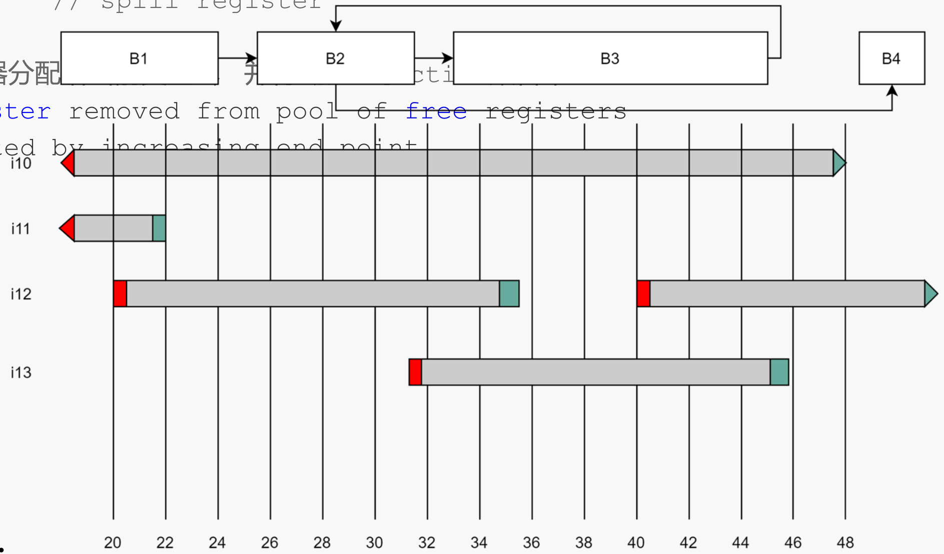
v4的interval为[18,21)

V1的interval[21,22)

Linear Scan Register Allocation

```
LinearScanRegisterAllocation
    active ← {}
    for each live interval i, in order of increasing start point do
        ExpireOldIntervals(i)
        if length(active) = R then   // 如果寄存器已经分配完毕
            SpillAtInterval(i)       // spill register
        else
            // 将一个未被分配的寄存器分配
            register[i] ← a register removed from pool of free registers
            add i to active, sorted by increasing end point

```



current=i13 active = {i10， i12， i13};

Linear Scan Register Allocation

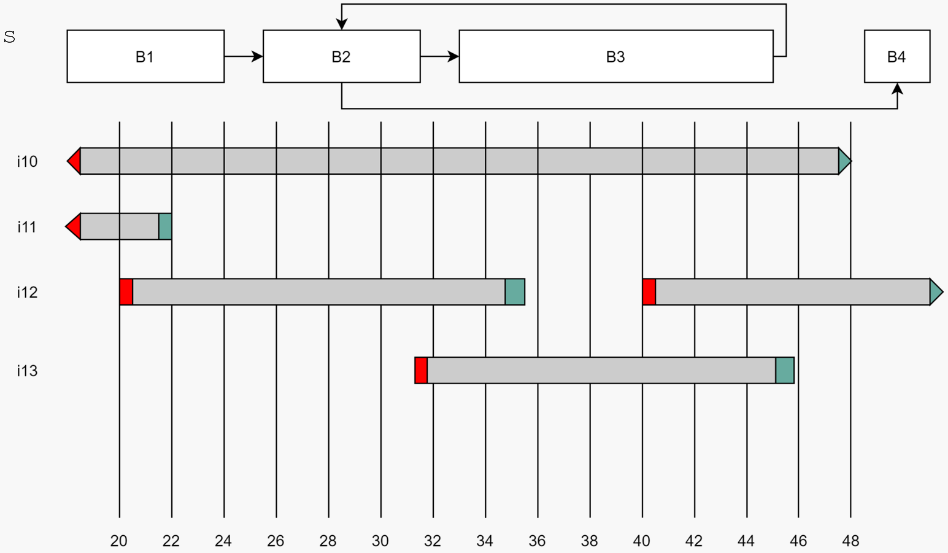
```
LinearScanRegisterAllocation
   active ← {}
   for each live interval i, in order of increasing start point do
       ExpireOldIntervals(i)
       if length(active) = R then  // 如果寄存器已经分配完毕
           SpillAtInterval(i)      // spill register
       else
           // 将一个未被分配的寄存器分配给当前变量i, 并将i加入到active集合中
           register[i] ← a register removed from pool of free registers
           add i to active, sorted by increasing end point

```

Linear Scan Register Allocation

根据i更新active集合

```
ExpireOldIntervals(i)
   for each interval j in active, in order of increasing end point do
       //如果变量J的生命周期在i开始前结束，那么将J的变量加入到未分配的寄存器池中
       if endpoint[j] ≥ startpoint[i] then
           return
       remove j from active
       add register[j] to pool of free registers
```



Linear Scan Register Allocation

```
//为变量i溢出一个寄存器
SpillAtInterval(i)
    spill ← last interval in active //从active取最后一个interval
    if endpoint[spill] > endpoint[i] then // 如果spill结束位置比i还大
        register[i] ← register[spill]  // spill的寄存器分配给i
        location[spill] ← new stack location //将spill溢出到内存中
        remove spill from active
        add i to active, sorted by increasing end point
    else
        location[i] ← new stack locationv // 无法溢出，将其放到堆栈中
```

Optimized interval splitting Linear Scan

[1] Wimmer C, Mössenböck H. Optimized interval splitting in a linear scan register allocator[C]//Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments. 2005: 132-141.

Optimized interval splitting Linear Scan

```
LinearScan
    unhandled = list of intervals storted by increasiong start positions
    active = {}; inactive = {}; handled = {};
    while unhandled != {} do
           current = pick and remove first interval from unhandled
           position = start position of current
           //根据current更新active/inactive/handled的状态
           for each interval it in active do
               if it ends before position then
                   move it from active to handled
               else if it dose not cover position then
                   move it from active to inactive
           for each interval it in inactive do
               if it ends before position then
                   move it from inactive to hanlded
               else if covers position then
                   move it from inactive to active
           //尝试用尚未分配的寄存器分配
           TryAllocationFreeReg
           //若分配失败则调用AllocateBlockedReg
           if allocation failed then AllocateBlockedReg
           if current has a register assigned then add current to active
```

Optimized interval splitting Linear Scan

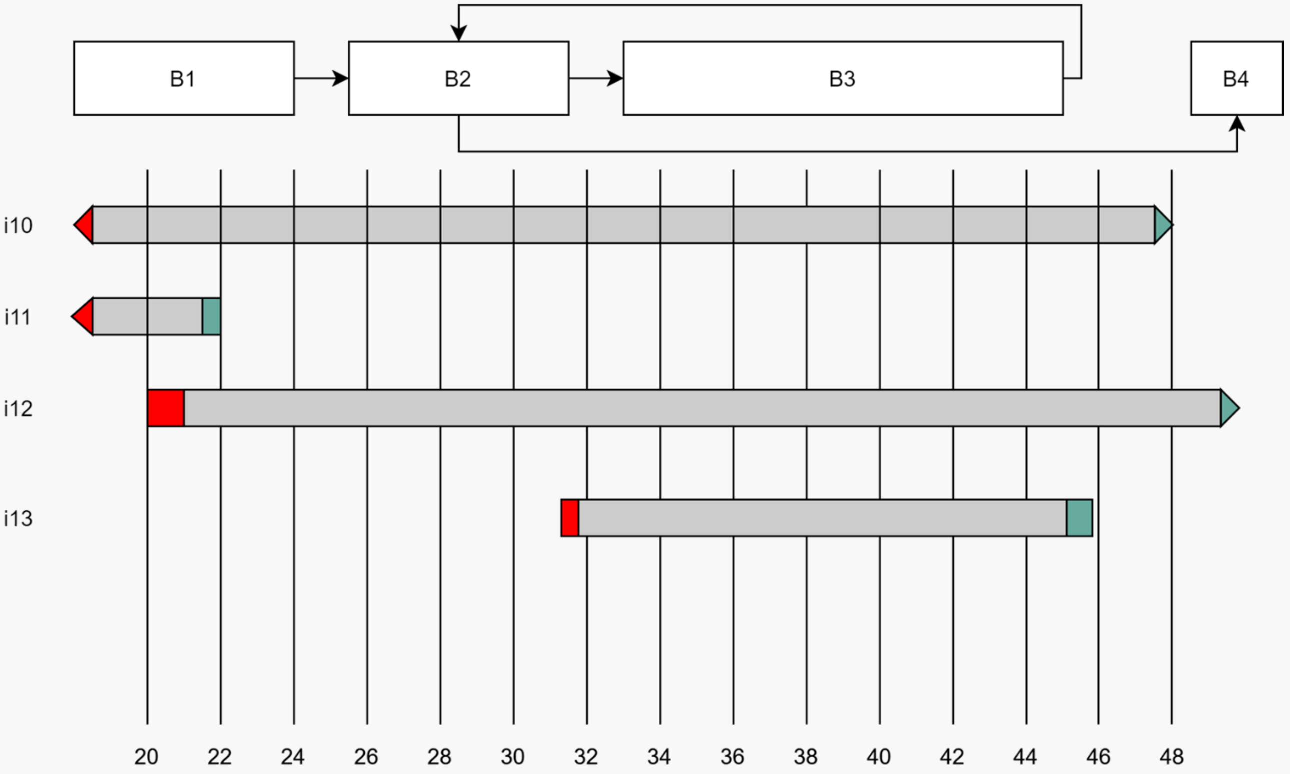
interval被分为四类:

- Unhandled: interval在position之后开始的
- Active:包含position的interval且已经被分配了寄存器
- Inactive: interval在position之前开始且在position之后结束但没有包含position(由于interval可以有不连续的区间组成)
- Handled: 在position之前结束的interval或者interval被溢出到内存当中了.

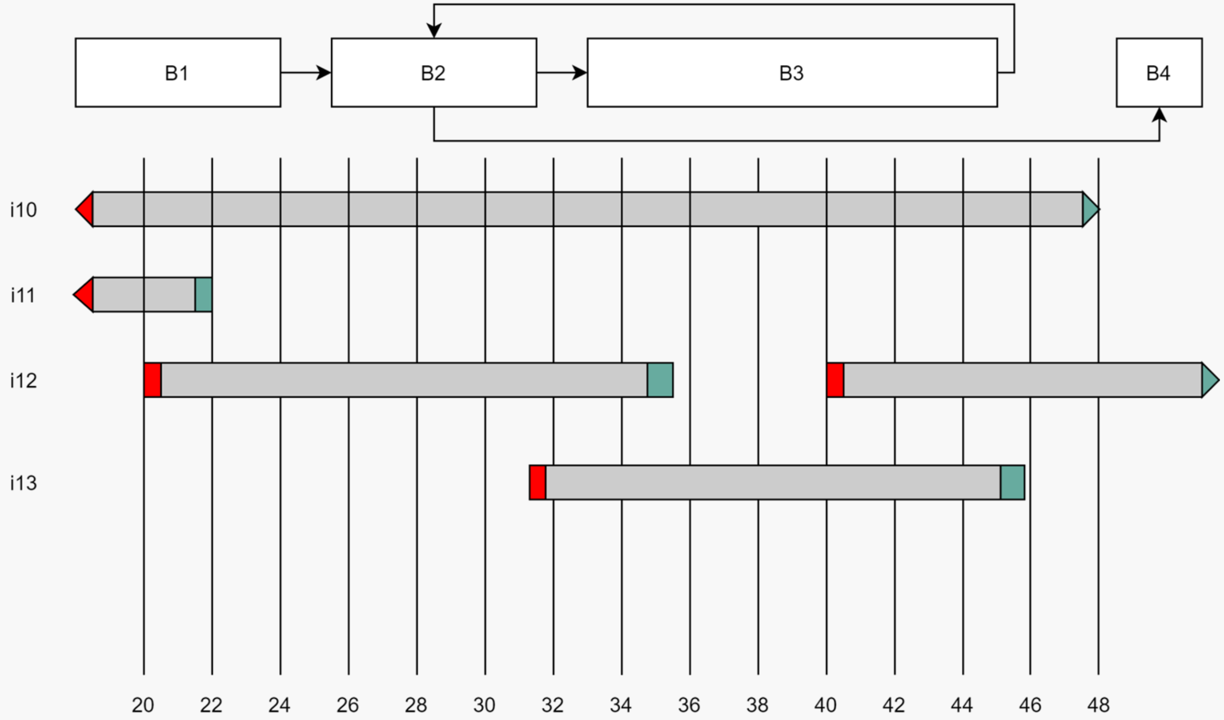
Optimized interval splitting Linear Scan

```
LinearScan
    unhandled = list of intervals storted by increasiong start positions
    active = {}; inactive = {}; handled = {};
    while unhandled != {} do
        current = pick and remove first interval from unhandled
        position = start position of current
        //根据current更新active/inactive/handled的状态
        for each interval it in active do
            if it ends before position then
                move it from active to handled
            else if it dose not cover position then
                move it from active to inactive
        for each interval it in inactive do
            if it ends before position then
                move it from inactive to hanlded
            else if covers position then
                move it from inactive to active
        //尝试用尚未分配的寄存器分配
        TryAllocationFreeReg
        //若分配失败则调用AllocateBlockedReg
        if allocation failed then AllocateBlockedReg
        if current has a register assigned then add current to active
```

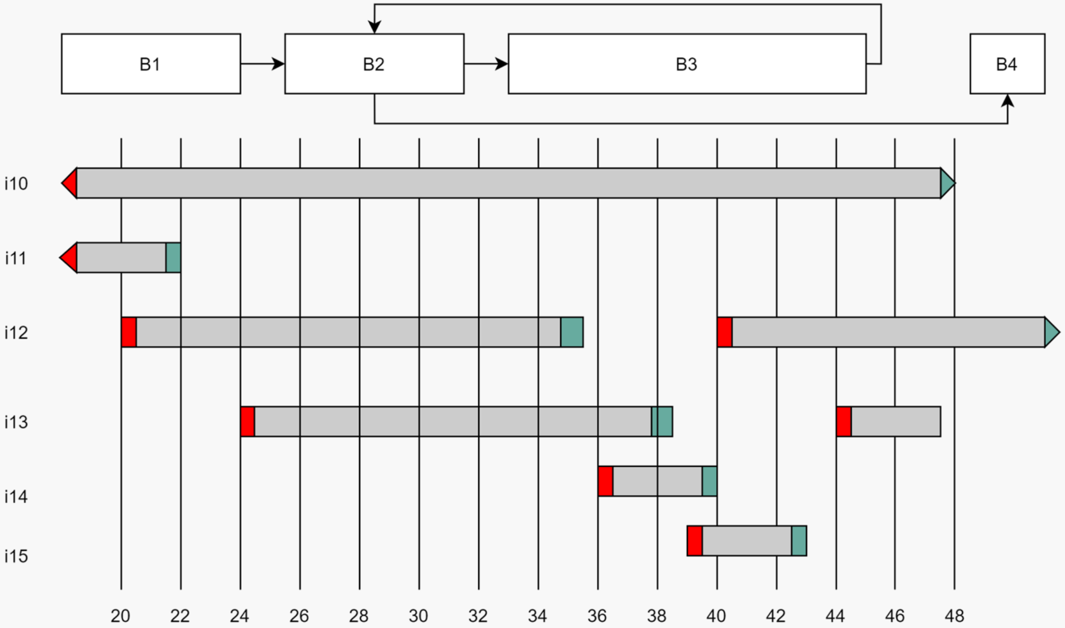
Optimized interval splitting Linear Scan



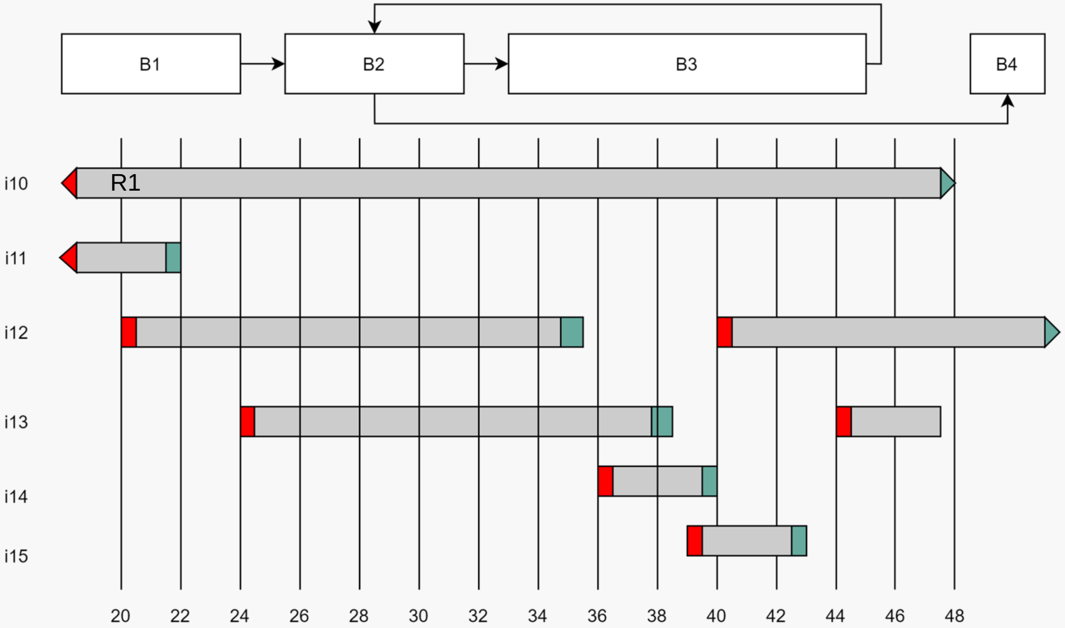
Optimized interval splitting Linear Scan



Optimized interval splitting Linear Scan

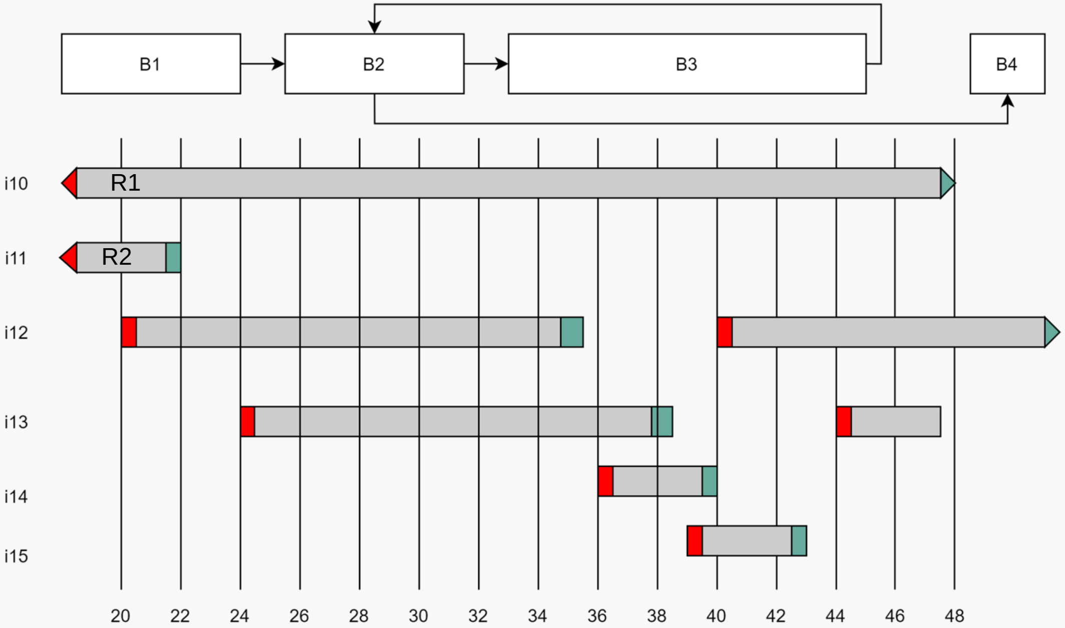


Optimized interval splitting Linear Scan



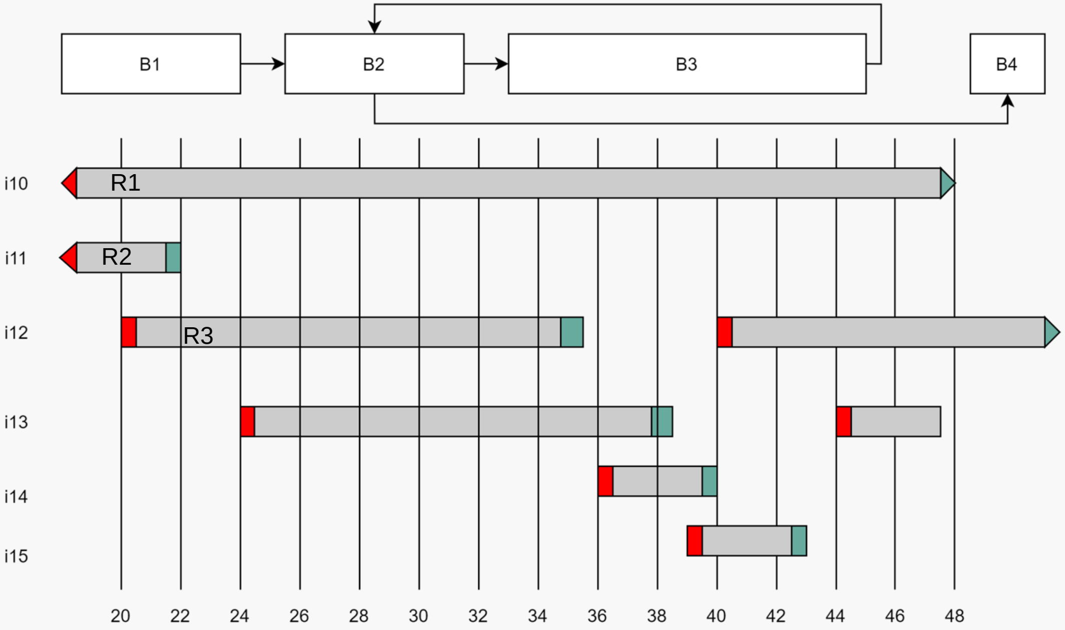
假设只有三个寄存器R1, R2, R3
开始时 unhandled = {i10, i11, i12, i13, i14, i15}。active = {}
current = i10： 分配R1给它， unhandled = {i11, i12, i13, i14, i15}, active = {i10}

Optimized interval splitting Linear Scan



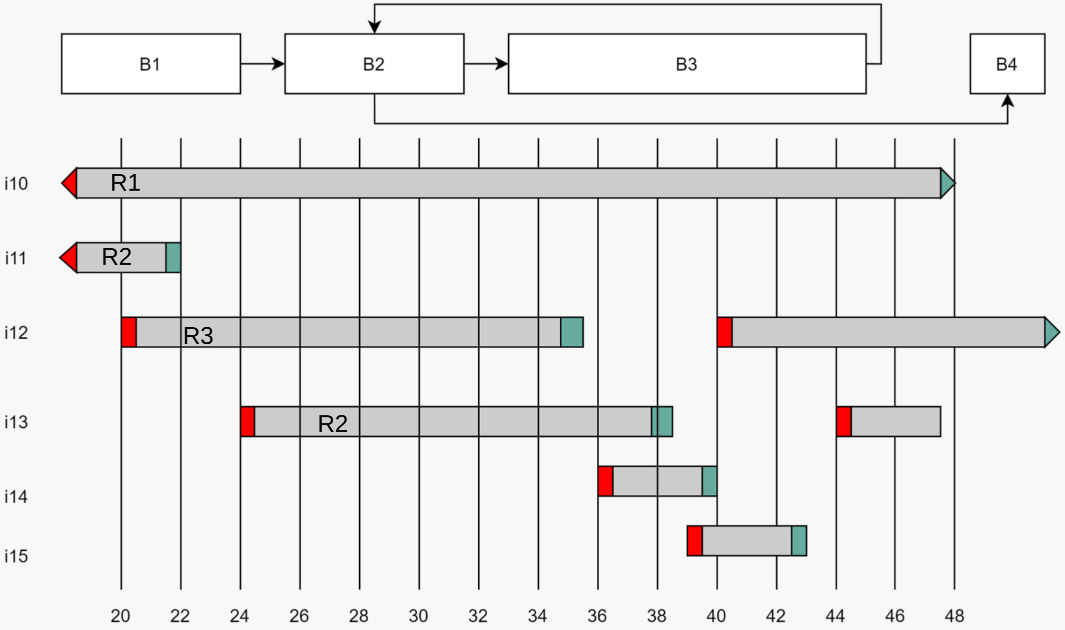
current = i11： 分配R2给它， unhandled = {i12, i13, i14, i15}, active = {i10, i11}

Optimized interval splitting Linear Scan



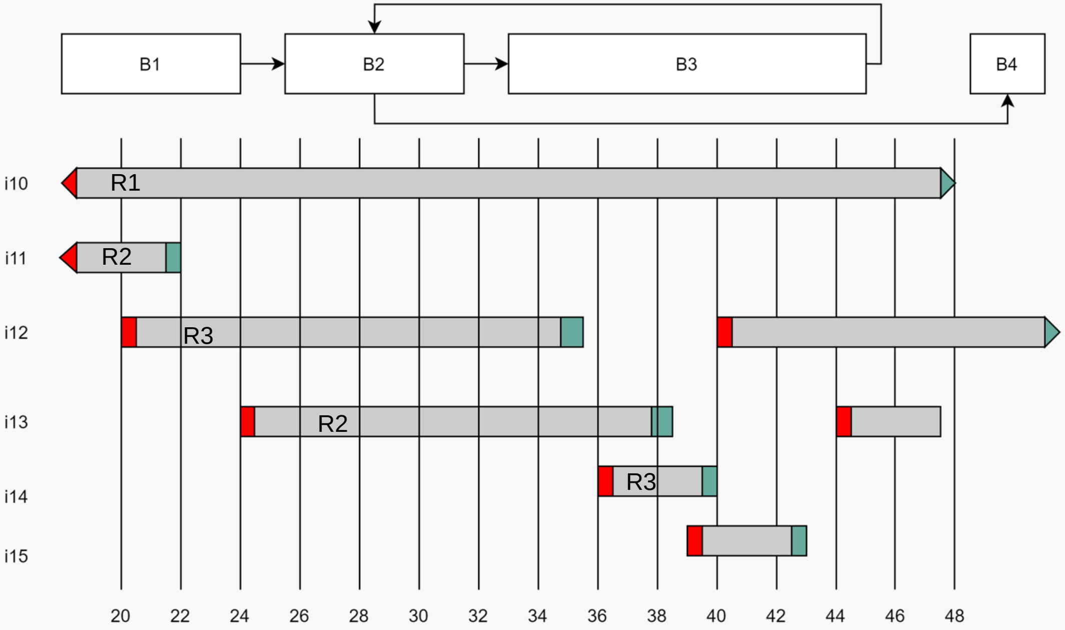
current = i12： 分配R3给它，unhandled = {i13, i14, i15}, active = {i10, i11, i12};

Optimized interval splitting Linear Scan



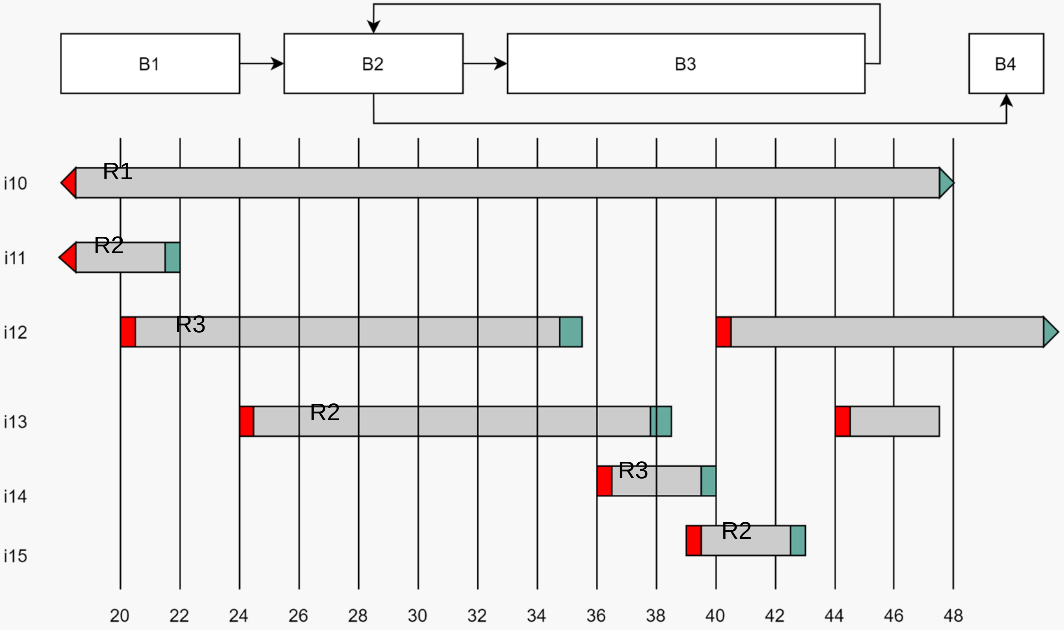
current = i13： 此时，i11已经结束了，可以把R2回收，并分配给i13，
unhandled = { i14, i15}, active = {i10，i12， i13}；

Optimized interval splitting Linear Scan



current = i14： i14可以放在i12的空隙中，则i14可以分配为R3；

Optimized interval splitting Linear Scan



current = i15： i15可以分配在i13的空隙中，则i15可以分配为R2；

Optimized interval splitting Linear Scan

```
LinearScan
    unhandled = list of intervals storted by increasiong start positions
    active = {}; inactive = {}; handled = {};
    while unhandled != {} do
           current = pick and remove first interval from unhandled
           position = start position of current
           //根据current更新active/inactive/handled的状态
           for each interval it in active do
               if it ends before position then
                   move it from active to handled
               else if it dose not cover position then
                   move it from active to inactive
           for each interval it in inactive do
               if it ends before position then
                   move it from inactive to hanlded
               else if covers position then
                   move it from inactive to active
           //尝试用尚未分配的寄存器分配
           TryAllocationFreeReg
           //若分配失败则调用AllocateBlockedReg
           if allocation failed then AllocateBlockedReg
           if current has a register assigned then add current to active
```

Optimized interval splitting Linear Scan

```
TryAllocationFreeReg
    set freeUntilPos of all physical registers to maxint
    // 初始化freeUntilPos
    for each interval it in active do
        freeUntilPos[it.reg] = 0
    for each interval it inactive interesting with current do
        freeUntilPos[it.reg] = next intersection of it with current

    //根据freeUntilPos挑选一个reg进行分配
    reg = register with highest freeUntilPos[reg] then
    if freeUntilPos[reg] = 0 then
         allocation failed
    else if current ends before freeUntilPos[reg] then
          current.reg = reg
    else
         current.reg = reg
         split current before freeUntilPos[reg]
```

Optimized interval splitting Linear Scan

```
AllocateBlockReg
    set nextUsePos of all physical registers to maxInt
    for each interval it in active do
        nextUsePos[it.reg] = next use of it after start of current
    for each interval it in inactive intersecting with current do
        nextUsePos[it.reg] = next use of it after start of current
    reg = regsiter with highest nextUsePos
    if first usage of current is after nextUsePos[reg] then
        assign spill slot to current
        split current before its first use position that requires a register
    else
        current.reg = reg
        split active interval for reg at position
        split ant inactive interval for reg at the end of its lifetime hole

    if current intersects with the fixed interval for reg then
        split current before this intersectionn
```

v8中的实现

https://paste.ubuntu.com/p/9g5qkWXPDh/