Hugues Cassé <casse@irit.fr>
FSI - Université de Toulouse

Cours 5 : optimisations

Compilation, Optimisation et Maîtrise du Code

Introduction

- non nécessaires $\Longrightarrow$ code exécutable
- amélioration de la qualité du code
  - temps d'exécution (plus fréquent)
  - place mémoire
    (environnement contraint – système embarqué)
  - déterminisme du code (temps-réel)
  - etc

Note : nom des présentations en anglais $\Longrightarrow$ correspondance avec les options des compilateurs

Simplifications algébriques

A) $x+0=0+x=x-0=x$

B) $0-x=-x$

C) $x\times 1=1\times x=x\ /\ 1=x$

D) $x\times 0=0\times x=0\ /\ x=0$

E) $-(-x))=x$

F) $x+(y)=x-y$

G) $x\times 2^k=x\ <<\ k$

I) $x\ /\ 2^k=x\ >>\ k$

J) $x\ <<\ 0=x\ >>\ 0=x$

K) $x\ <<\ |\tau|=$
$x\ >>\ |\tau|=0$

L) $x^2=x\times x$

M) $x^0=1$

Optimisation à lucarne

sur les quadruplets

- recherche d'un modèle de simplification algébrique
- fenêtre de quelques instructions
- reconnaissance $\Longrightarrow$ remplacement

sur le DAG $\Longrightarrow$ insertion dans les modèles d'instructions

$$
\begin{array}{lll}
\ldots & & \ldots \\
v_3 \leftarrow 8 & (v_i \leftarrow 2^k) & \Longrightarrow \quad v_3 \leftarrow 8 \\
 & & \quad v_{51} \leftarrow 3 \\
v_4 \leftarrow v_1 \times v_3 & (v_j \leftarrow v_k \times v_i) & \quad v_4 \leftarrow v_1 << v_{51} \\
\ldots & & \ldots
\end{array}
$$

NOTE : $v_3 \leftarrow 8$ sera éliminée par une autre phase d'optimisation (*dead-code elimination*)

Cas des branchements

goto (l_i) $\Longrightarrow$ λ
label (l_i)

if $v_i\ \omega\ v_j$ goto l_k $\Longrightarrow$ if $\neg(v_i\ \omega\ v_j)$ goto l_l
goto l_l
label l_k label l_k

Opérations propres à une architecture

$$v_i \leftarrow v_j \times 10 \iff \begin{cases} \text{ADD } v_i, v_j, v_j \\ \text{ADD } v_i, v_i, v_i, \text{LSL } \#2 \end{cases}$$

$$v_i \leftarrow v_j \times 14 \iff \begin{cases} \text{ADD } v_i, v_j, v_j \\ \text{RSB } v_i, v_i, v_i, \text{LSL } \#3 \end{cases}$$

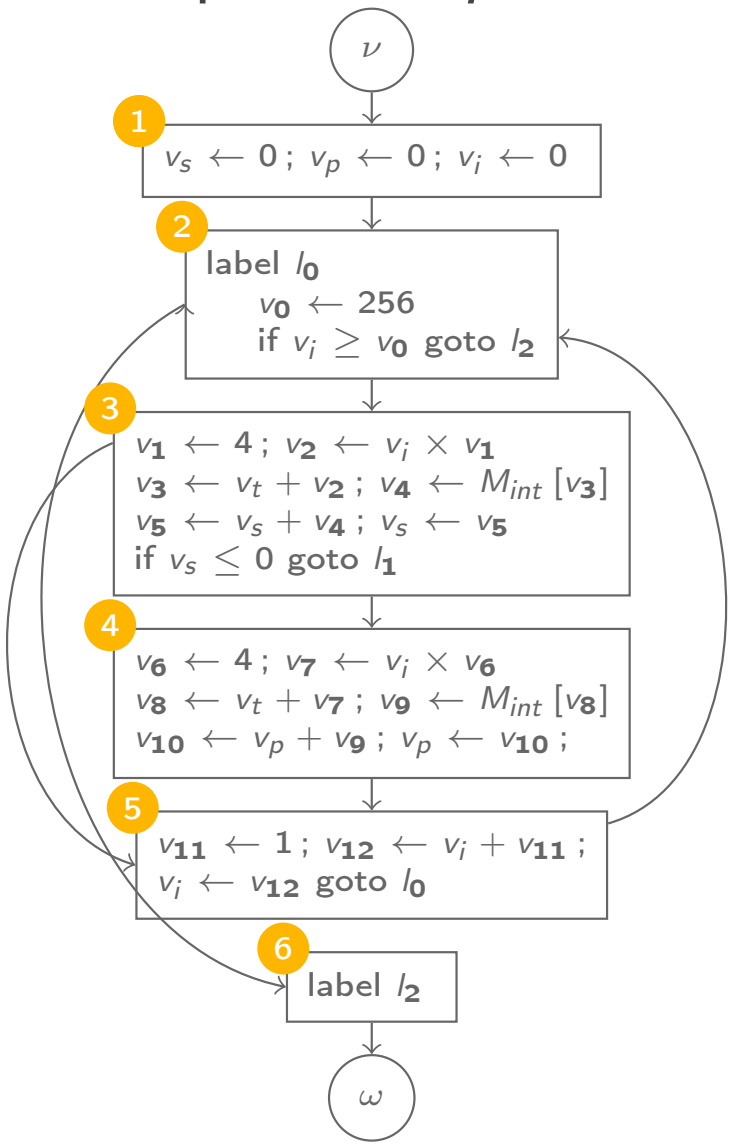
limite : $\text{temps}_{\text{code optimisé}} < \text{temps}_{\text{instruction}}$

Pourquoi y a-t-il de telles expressions ?

- code automatiquement généré
- utilisation intensive des macro-définitions
- conservation de la lisibilité
- résultat d'optimisations précédentes

```
#define KO   (1 << 10)
#define MO   (KO << 10)
 ...
 memory_size = 4 * MO;
 ...
```

Exemple : *expression disponibles*

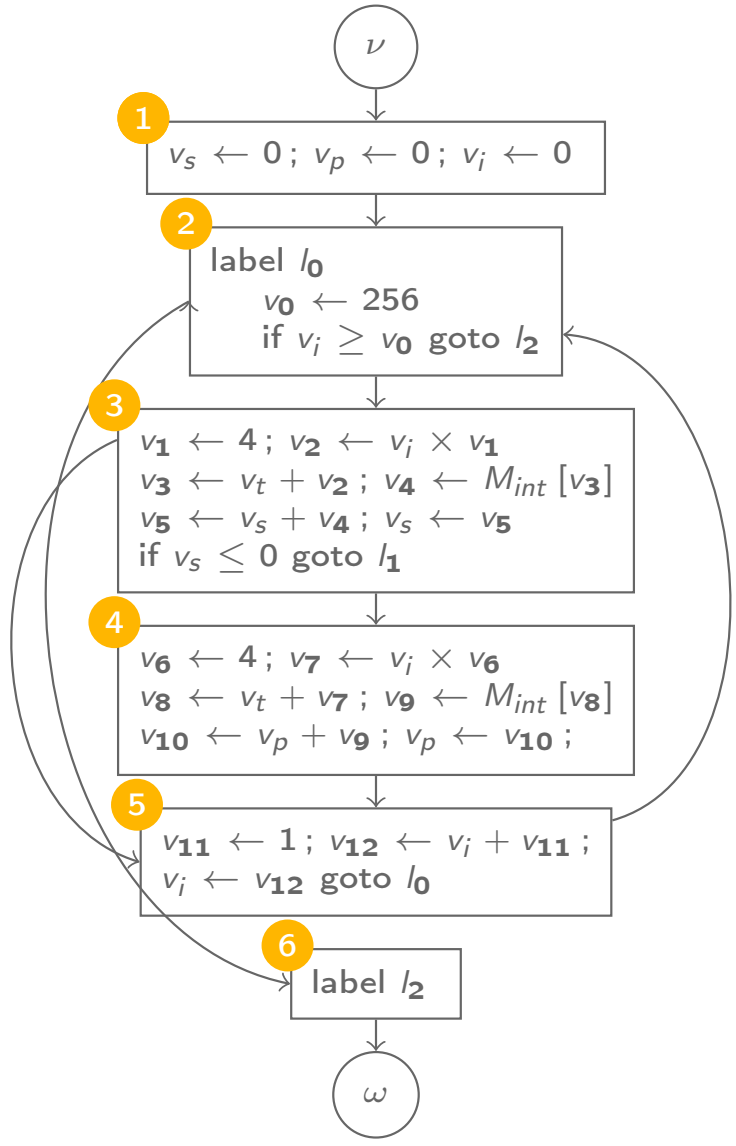



```
s = 0;
p = 0;
i = 0;
while(i < 256) {
   s += t[i];
   if(s > 0)
      p += t[i];
   i++;
}
```

Attention !

Optimisation :

- comme $v_1 \leftarrow 4$ (BB 3) est disponible avant BB 4
- supprimer $v_6 \leftarrow 4$
- remplacer v_6 par v_1



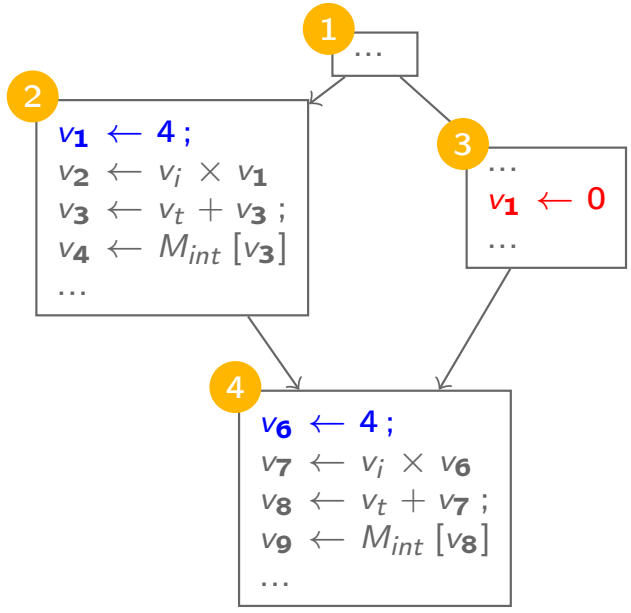
Attention !

Optimisation :

- comme $v_1 \leftarrow 4$ (BB 3) est disponible avant BB 4
- supprimer $v_6 \leftarrow 4$
- remplacer v_6 par v_1

Est-on sûr que c'est toujours possible ?

- contre-exemple
- $v_1 \leftarrow 4$?
- Doit être disponible sur tous les chemins !



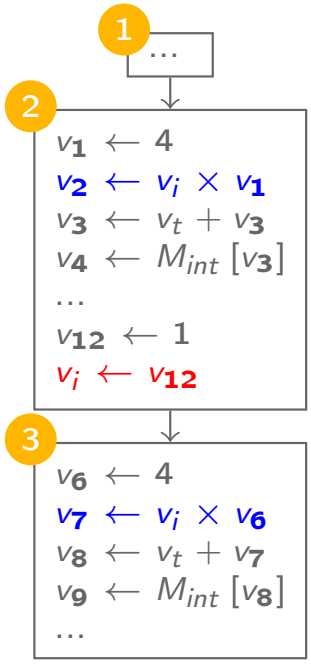
Attention !

Optimisation :

- comme $v_1 \leftarrow 4$ (BB 3) est disponible avant BB 4
- supprimer $v_6 \leftarrow 4$
- remplacer v_6 par v_1

Est-on sûr que c'est toujours possible ?

- contre-exemple
- $v_2 \leftarrow v_i \times v_1$?
- Non car v_i est modifié !



Analyse

Soit un CFG $G = \langle V, E, \nu, \omega \rangle$
Pour optimiser $b \in V$,

- expressions (quadruplets) disponibles en entrée de b – IN_b
- $IN_b \subset Q$ est un ensemble car plusieurs chemins mènent à b
- IN_b doit être complet : prendre en compte tous les chemins

Pour calculer IN_b :

- ajouter les quadruplets de $b - v_i \leftarrow e_1$
- supprimer les quadruplets affectés dans b
  $(v_i \leftarrow e_1,\ v_i \leftarrow e_2,\ \ldots)$
- suppriler les quadruplets dont un opérande est affecté dans b
  $(v_j \leftarrow v_i\ \omega\ v_k,\ v_k \leftarrow v_l\ \omega\ v_i,\ \ldots)$

Problèmes :

- calculer pour tous les chemins pour chaque $b \in V$!

Idée : calculer tout en parallèle !

Poussons l'analyse :

$\mathcal{D} \subseteq 2^Q$ ensemble des expressions (quadruplets)

$IN_b \subseteq \mathcal{D}$ expressions en entrée de b

$OUT_b \subseteq \mathcal{D}$ expressions en sortie de b

$KILL_b \subseteq \mathcal{D}$ expressions supprimées par b

$GEN_b \subseteq \mathcal{D}$ expression ajoutées par b

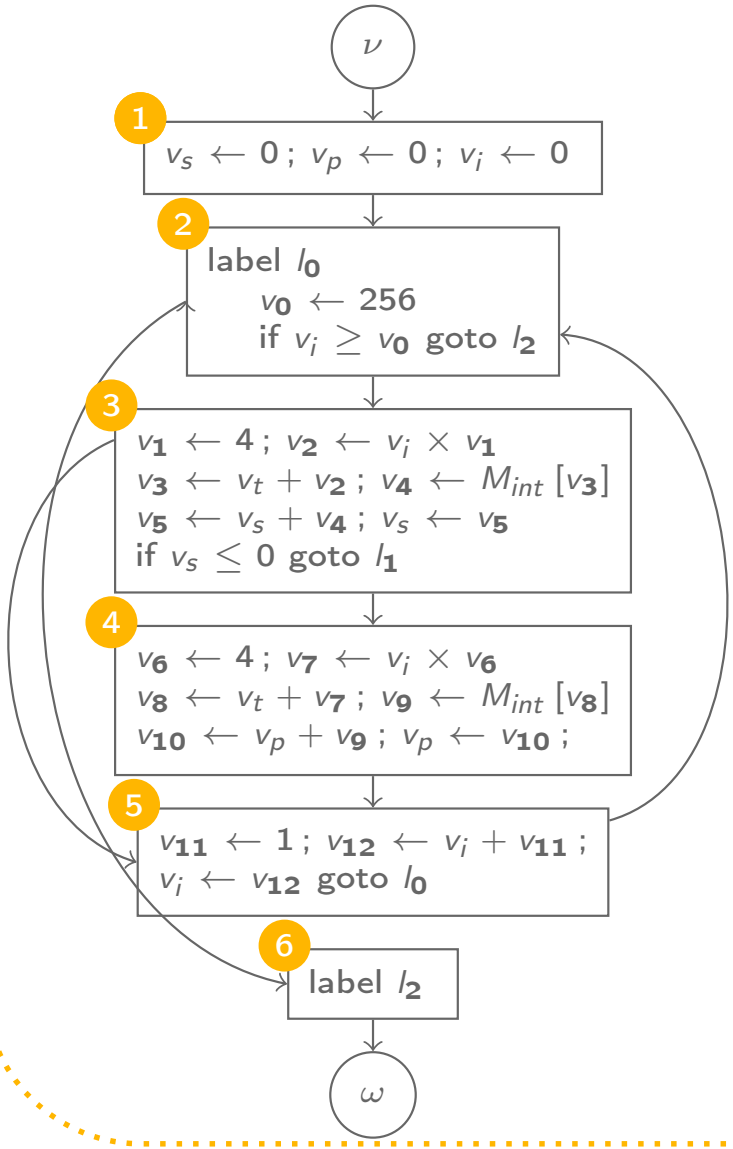
Definition

calcul des IN_v :

$$IN_b = \bigcap_{b' \in PRED(b)} OUT_b$$

$$OUT_b = (IN_b \setminus KILL_b) \cup GEN(b)$$

Suite de l'exemple



$$\mathcal{D} = \{v_s \leftarrow 0, v_p \leftarrow 0, \ldots\}$$

$$GEN_1 = \{v_s \leftarrow 0, v_p \leftarrow 0, v_i \leftarrow 0\}$$

$$KILL_1 = \{v_s \leftarrow 0, v_p \leftarrow 0, v_i \leftarrow 0,$$
$$v_s \leftarrow v_5, v_p \leftarrow v_{10}, v_i \leftarrow v_1,$$
$$v_5 \leftarrow v_s + v_4, v_{10} \leftarrow v_p + v_9,$$
$$v_{12} \leftarrow v_i + v_{11}\}$$

$$GEN_2 = \{v_0 \leftarrow 256\}$$

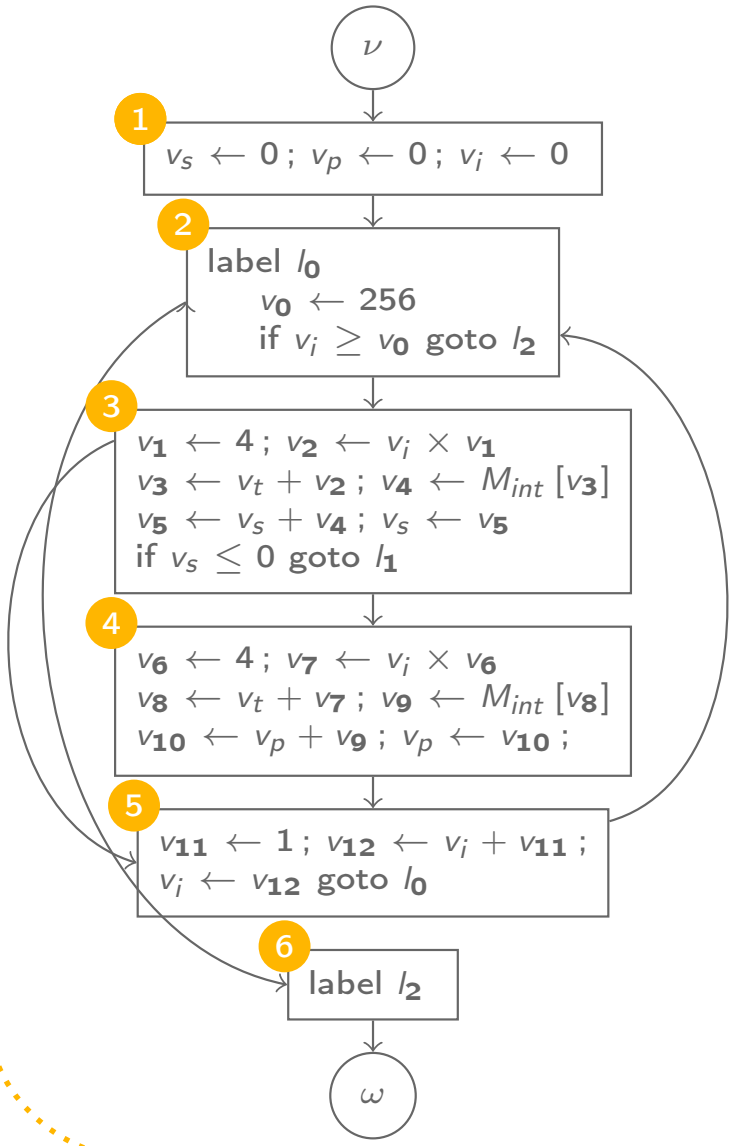
$$KILL_2 = \{v_0 \leftarrow 256\}$$

$$GEN_3 = \{v_1 \leftarrow 4, v_2 \leftarrow v_i \times v_1,$$
$$v_3 \leftarrow v_t + v_2, v_9 \leftarrow M_{int}[v_3],$$
$$v_5 \leftarrow v_s + v_4, v_s \leftarrow v_5\}$$

$$KILL_3 = \{v_1 \leftarrow 4, v_2 \leftarrow v_i \times v_1,$$
$$v_3 \leftarrow v_t + v_2, v_9 \leftarrow M_{int}[v_3],$$
$$v_5 \leftarrow v_s + v_4, v_s \leftarrow v_5,$$
$$v_s \leftarrow 0\}$$

$$\ldots$$

Système d'équation



$$
\begin{aligned}
IN_\nu &= \emptyset \\
OUT_\nu &= (IN_\nu \setminus KILL_\nu) \cup GEN_\nu \\
IN_1 &= OUT_\nu \\
OUT_1 &= (IN_1 \setminus KILL_1) \cup GEN_1 \\
IN_2 &= OUT_1 \cap OUT_5 \\
OUT_2 &= (IN_2 \setminus KILL_2) \cup GEN_2 \\
IN_3 &= OUT_2 \\
OUT_3 &= (IN_3 \setminus KILL_3) \cup GEN_3 \\
IN_4 &= OUT_3 \\
OUT_4 &= (IN_4 \setminus KILL_4) \cup GEN_4 \\
IN_5 &= OUT_4 \cap OUT_3 \\
OUT_5 &= (IN_5 \setminus KILL_5) \cup GEN_5 \\
IN_6 &= OUT_2 \\
OUT_6 &= (IN_6 \setminus KILL_6) \cup GEN_6 \\
IN_\omega &= OUT_6 \\
OUT_\omega &= (IN_\omega \setminus KILL_\omega) \cup GEN_\omega
\end{aligned}
$$

Comment le résoudre ?

Theorem

(Knaster-Tarsky) Si $(A, \sqsubseteq, \sqcup, \sqcap, \top, \bot)$ *est un treillis et* $F : A \mapsto A$ *monotone, alors il existe un point fixe* $X \in A$ *vérifiant* $F(X) = X$.

- peu importe l'ordre du calcul, F convergera vers un point fixe,
- pour avoir le plus petit, il faut partir de $\bot$
- calculer $F^{n+1} = F(F^n)$ avec $F^0 = \bot$ jusqu'à atteindre ce point fixe

Application à un BB

- $(2^{\mathcal{D}}, \supseteq, \cap, \cup, \mathcal{D}, \emptyset)$ est un treillis
- F_b fonction des OUT_i, paramètre $X_i = OUT_i$, prédécesseur de b
- F_b produit un nouveau OUT_b
- $F_b(X_{1 \leq i \leq n}) = ((\bigcap_{1 \leq i \leq n} X_i) \setminus KILL_b) \cup GEN_b$
  calcul de OUT_b en fonction des OUT_i
- F_b est (dé)croissant
  - $\forall X, Y, X', Y' \subseteq \mathcal{D} \ \wedge \ X \supseteq X' \ \wedge \ Y \supseteq Y'$
  - $(X \cap Y) \supseteq (X' \cap Y') \rightarrow$ intersection décroissante
  - $(X \cup Y) \supseteq (X' \cup Y') \rightarrow$ union décroissante
  - $\setminus$ n'est généralement pas décroissante mais $KILL_b$ constant !
    $X \setminus KILL_b \supseteq X' \setminus KILL_b \rightarrow$ différence décroissante

Élargissement au CFG

- $\mathcal{D}_G = 2^{\mathcal{D}^{|V|}}$ – ensemble des OUT_b du CFG
- $(X_1, X_2, ...) \sqsubseteq (Y_1, Y_2, ...)$ – ordre lexicographique
- $(X_1, X_2, ...) \sqcap (Y_1, Y_2, ...) = (X_1 \cap Y_1, X_2 \cap Y_2, ...)$
- $(X_1, X_2, ...) \sqcup (Y_1, Y_2, ...) = (X_1 \cup Y_1, X_2 \cup Y_2, ...)$
- $\bot = (\mathcal{D}, \mathcal{D}, ...)$
- $\top = (\emptyset, \emptyset, ...)$
- $(\mathcal{D}_G, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ est un treillis !

Élargissement à notre système d'équation

- $F_G : \mathcal{D}_G \mapsto \mathcal{D}_G$ tel que
- $F_G(OUT_1, OUT_2, ...) =$
  $(F_1(OUT_1, OUT_2, ...), F_2(OUT_1, ...), ...)$
- F_G croissant dans $\mathcal{D}_G$ car F_b croissant dans $2^{\mathcal{D}}$
- solution pour les OUT_b = plus petit point fixe
  - $F_G^0 = \perp_G$
  - $F_G^{n+1} = F_G(F_G^n)$

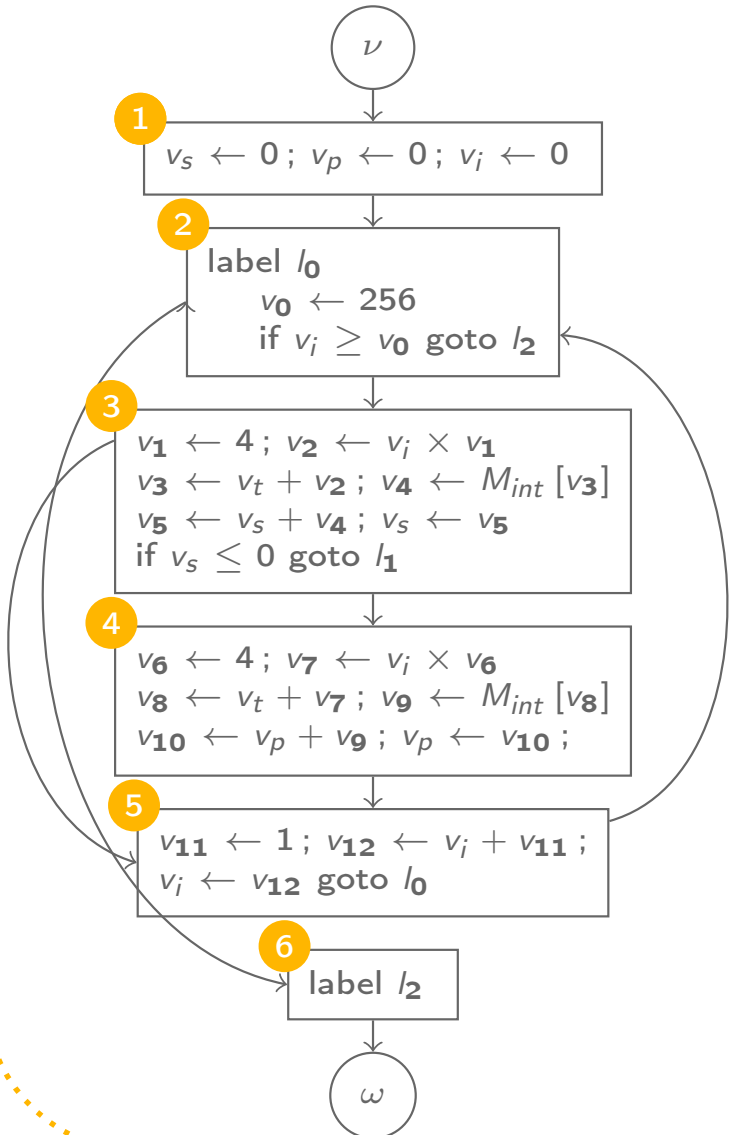
Algorithme

for all $b \in V$ **do**
  $IN_b \leftarrow \bot$; $OUT_b \leftarrow \bot$
end for
$changed \leftarrow true$
while *changed* **do**
  $changed \leftarrow false$
  for all $b \in V$ **do**
    $IN \leftarrow \bigsqcap_{b' \in PRED(b)} OUT_{b'}$
    $OUT \leftarrow (IN \setminus KILL_b) \cup GEN_b$
    if $OUT \neq OUT_b$ **then**
      $changed \leftarrow true$
      $IN_b \leftarrow IN$; $OUT_b \leftarrow OUT$
    end if
  end for
end while

Algorithme avec liste de travail

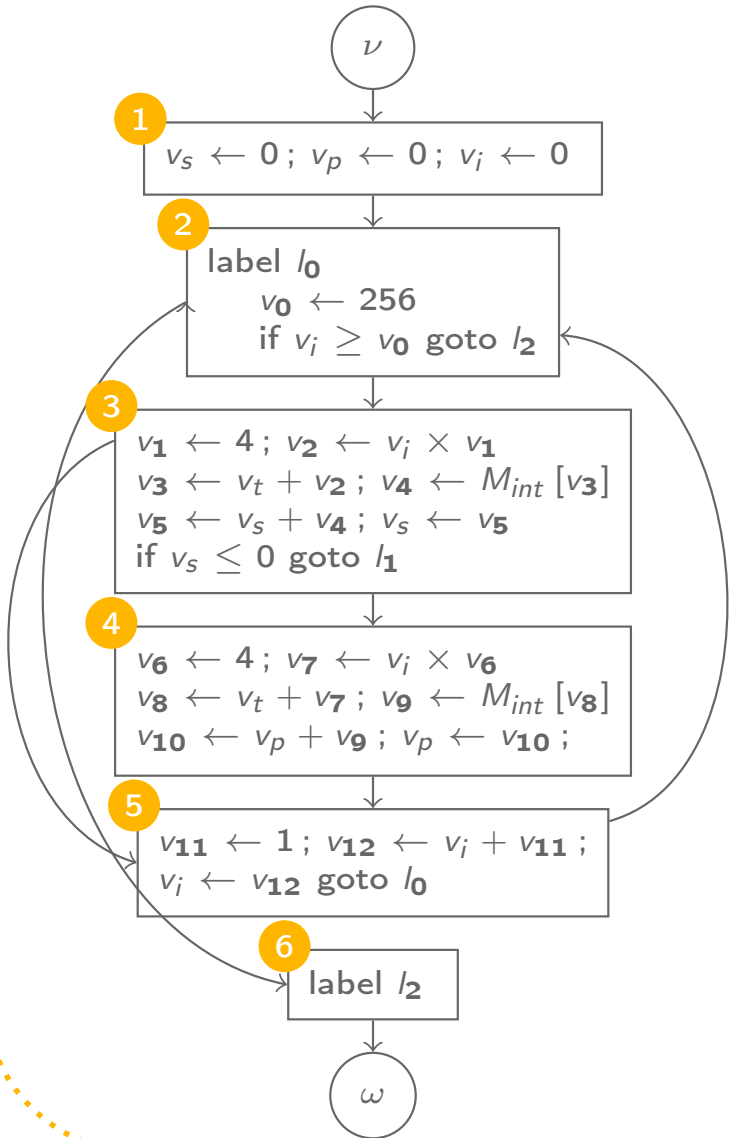
for all $b \in V$ **do**
  $IN_b \leftarrow \bot$; $OUT_b \leftarrow \bot$
end for
$wl \leftarrow \{\nu\}$
while $wl \neq \emptyset$ **do**
  let $b \in wl$; $wl \leftarrow wl \setminus \{b\}$
  $IN \leftarrow \bigsqcap_{b' \in PRED(b)} OUT_{b'}$
  $OUT \leftarrow (IN \setminus KILL_b) \cup GEN_b$
  if $OUT \neq OUT_b$ **then**
    $IN_b \leftarrow IN$; $OUT_b \leftarrow OUT$
    $wl \leftarrow wl \cup SUCC(b)$
  end if
end while

Exemple (1)



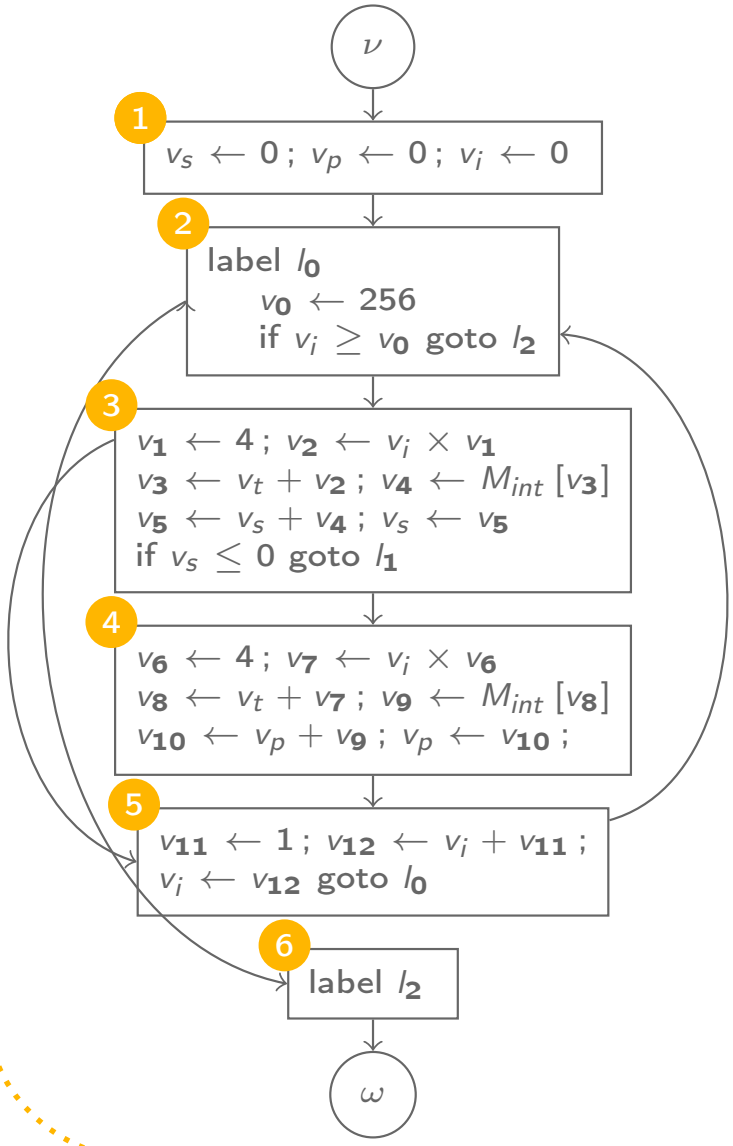
$$
\begin{aligned}
IN_1 &= \emptyset \\
OUT_1 &= \{v_s \leftarrow 0, v_p \leftarrow 0, v_i \leftarrow 0\} \\
IN_2 &= OUT_1 \cap OUT_5 = OUT_1 \cap \mathcal{D} \\
OUT_2 &= \{v_s \leftarrow 0, v_p \leftarrow 0, v_i \leftarrow 0, \\
&\quad v_0 \leftarrow 256\} \\
IN_3 &= OUT_2 \\
OUT_3 &= \{v_p \leftarrow 0, v_i \leftarrow 0, v_0 \leftarrow 256, \\
&\quad v_1 \leftarrow 4, v_2 \leftarrow v_i \times v_1, \\
&\quad v_3 \leftarrow v_t + v_2, v_4 \leftarrow M_{int}\,[v_3], \\
&\quad v_s \leftarrow v_5\} \\
IN_4 &= OUT_3 \\
OUT_4 &= \{v_i \leftarrow 0, v_0 \leftarrow 256, \\
&\quad v_1 \leftarrow 4, v_2 \leftarrow v_i \times v_1, \\
&\quad v_3 \leftarrow v_t + v_2, v_4 \leftarrow M_{int}\,[v_3], \\
&\quad v_s \leftarrow v_5, v_6 \leftarrow 4, \\
&\quad v_7 \leftarrow v_i \times v_6, v_8 \leftarrow v_t + v_7, \\
&\quad v_9 \leftarrow M_{int}[v_8], v_p \leftarrow v_{10}\}
\end{aligned}
$$

Exemple (2)



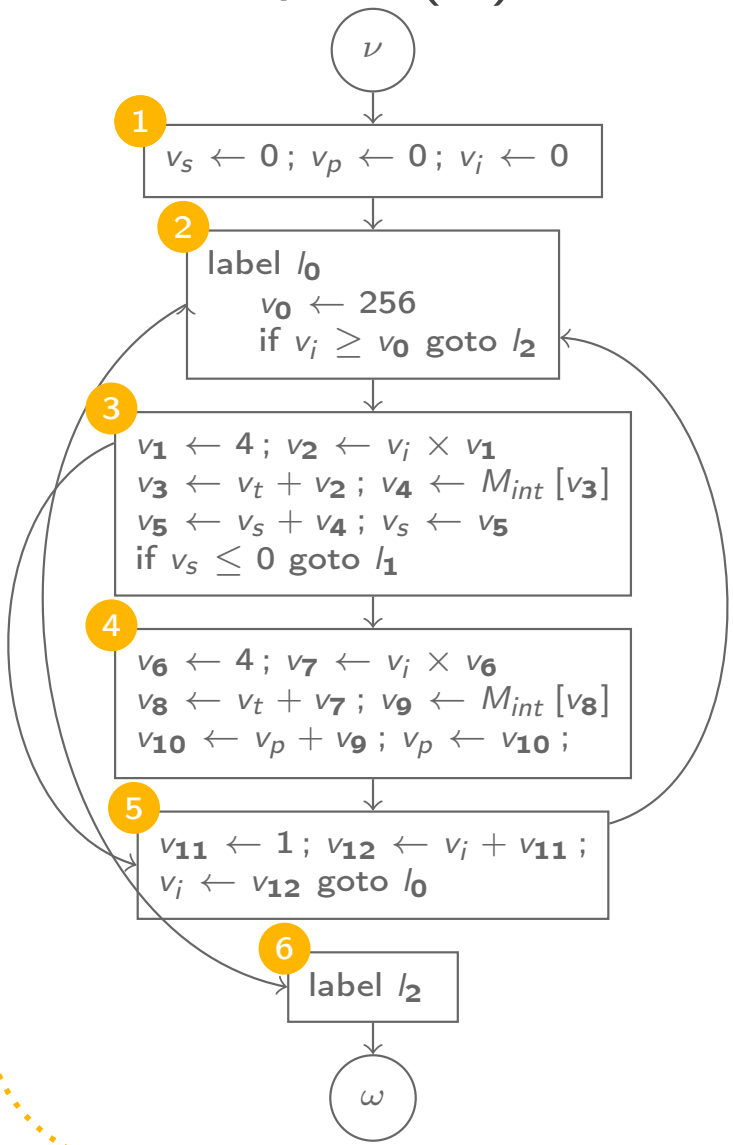
$$
\begin{aligned}
&\ldots\\
OUT_3 = \{&v_p \leftarrow 0, v_i \leftarrow 0, v_0 \leftarrow 256,\\
&v_1 \leftarrow 4, v_2 \leftarrow v_i \times v_1,\\
&v_3 \leftarrow v_t + v_2, v_4 \leftarrow M_{int}\,[v_3]\,,\\
&v_s \leftarrow v_5\}\\
IN_4 = {}& OUT_3\\
OUT_4 = \{&v_i \leftarrow 0, v_0 \leftarrow 256,\\
&v_1 \leftarrow 4, v_2 \leftarrow v_i \times v_1,\\
&v_3 \leftarrow v_t + v_2, v_4 \leftarrow M_{int}\,[v_3]\,,\\
&v_s \leftarrow v_5, v_6 \leftarrow 4,\\
&v_7 \leftarrow v_i \times v_6, v_8 \leftarrow v_t + v_7,\\
&v_9 \leftarrow M_{int}[v_8], v_p \leftarrow v_{10}\}\\
IN_5 = \{&v_i \leftarrow 0, v_0 \leftarrow 256, v_1 \leftarrow 4,\\
&v_2 \leftarrow v_i \times v_1, v_3 \leftarrow v_t + v_2,\\
&v_4 \leftarrow M_{int}\,[v_3]\}\\
OUT_5 = \{&v_0 \leftarrow 256, v_1 \leftarrow 4,\\
&v_2 \leftarrow v_i \times v_1, v_3 \leftarrow v_t + v_2,\\
&v_4 \leftarrow M_{int}\,[v_3]\,, v_i \leftarrow v_{12}\}
\end{aligned}
$$

Exemple (3)



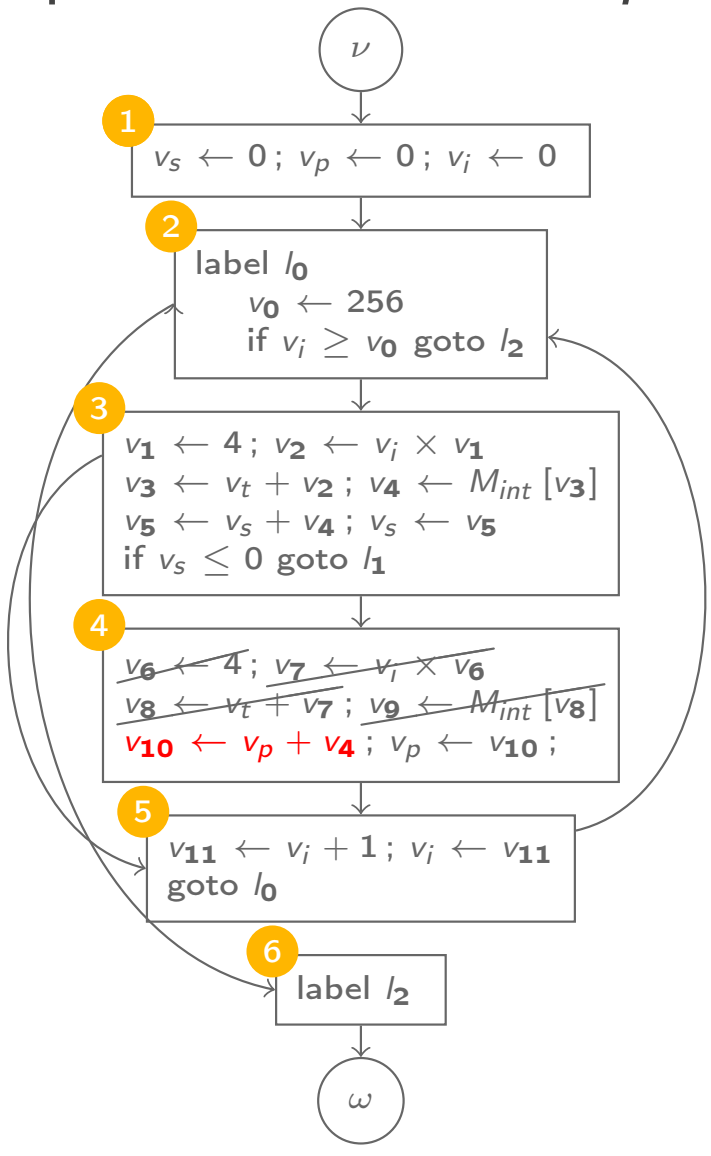
$$
\begin{aligned}
&\ldots \\
OUT_1 &= \{v_s \leftarrow 0, v_p \leftarrow 0, v_i \leftarrow 0\} \\
&\ldots \\
OUT_5 &= \{v_0 \leftarrow 256, v_1 \leftarrow 4, \\
&\quad v_2 \leftarrow v_i \times v_1, v_3 \leftarrow v_t + v_2, \\
&\quad v_4 \leftarrow M_{int}[v_3], v_i \leftarrow v_{12}\} \\
IN_2 &= \emptyset \\
OUT_2 &= \{v_0 \leftarrow 256\} \\
OUT_3 &= \{v_0 \leftarrow 256, v_1 \leftarrow 4, \\
&\quad v_2 \leftarrow v_i \times v_1, v_3 \leftarrow v_t + v_2, \\
&\quad v_4 \leftarrow M_{int}[v_3], v_s \leftarrow v_5\} \\
OUT_4 &= \{v_0 \leftarrow 256, v_1 \leftarrow 4, \\
&\quad v_2 \leftarrow v_i \times v_1, v_3 \leftarrow v_t + v_2, \\
&\quad v_4 \leftarrow M_{int}[v_3], v_s \leftarrow v_5, \\
&\quad v_6 \leftarrow 4, v_7 \leftarrow v_i \times v_6, \\
&\quad v_8 \leftarrow v_t + v_7, v_4 \leftarrow M_{int}[v_3], \\
&\quad v_p \leftarrow v_{10}\}
\end{aligned}
$$

Exemple (4)



$$
\begin{aligned}
&\ldots \\
OUT_3 = \{&v_0 \leftarrow 256, v_1 \leftarrow 4, \\
&v_2 \leftarrow v_i \times v_1, v_3 \leftarrow v_t + v_2, \\
&v_4 \leftarrow M_{int}\,[v_3]\,, v_s \leftarrow v_5\} \\
&\ldots \\
OUT_4 = \{&v_0 \leftarrow 256, v_1 \leftarrow 4, \\
&v_2 \leftarrow v_i \times v_1, v_3 \leftarrow v_t + v_2, \\
&v_4 \leftarrow M_{int}\,[v_3]\,, v_s \leftarrow v_5, \\
&v_6 \leftarrow 4, v_7 \leftarrow v_i \times v_6, \\
&v_8 \leftarrow v_t + v_7, v_4 \leftarrow M_{int}\,[v_3]\,, \\
&v_p \leftarrow v_{10}\} \\
IN_5 = \{&v_0 \leftarrow 256, v_1 \leftarrow 4, \\
&v_2 \leftarrow v_i \times v_1, v_3 \leftarrow v_t + v_2, \\
&v_4 \leftarrow M_{int}\,[v_3]\,, v_s \leftarrow v_5\}
\end{aligned}
$$

Optimisation : *expressions disponibles*



$$IN_1 = \emptyset$$

$$IN_2 = \emptyset$$

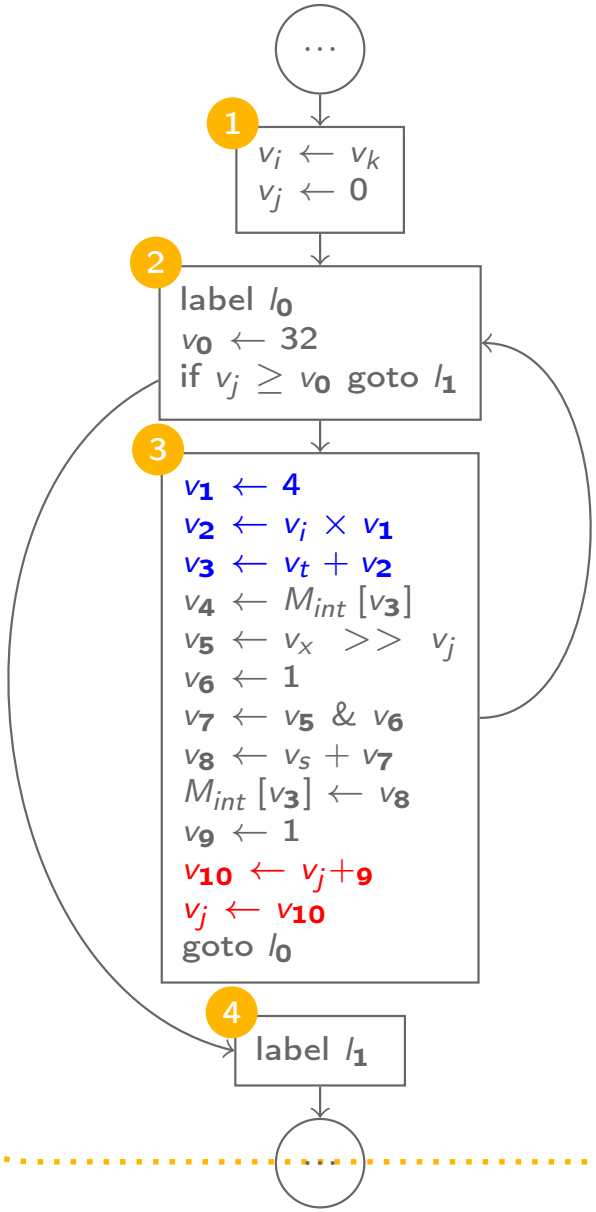
$$IN_3 = \{v_0 \leftarrow 256\}$$

$$IN_4 = \{v_0 \leftarrow 256, v_1 \leftarrow 4,$$
$$v_2 \leftarrow v_i \times v_1, v_3 \leftarrow v_t + v_2,$$
$$v_4 \leftarrow M_{int}\,[v_3]\,, v_s \leftarrow v_5\}$$

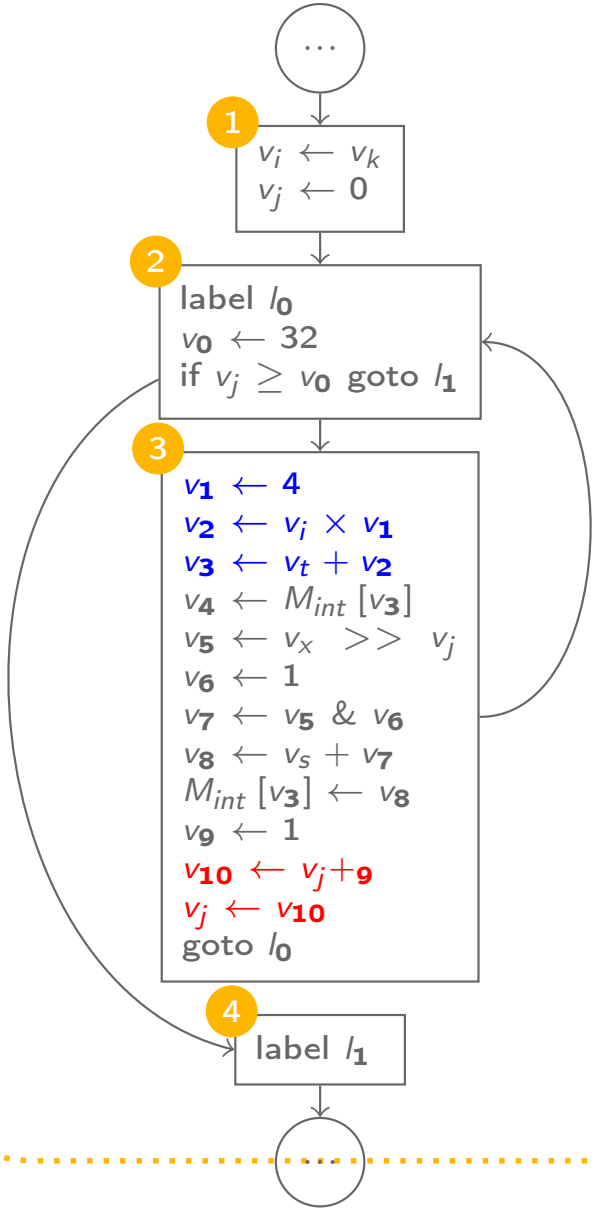
$$IN_5 = \{v_0 \leftarrow 256, v_1 \leftarrow 4,$$
$$v_2 \leftarrow v_i \times v_1, v_3 \leftarrow v_t + v_2,$$
$$v_4 \leftarrow M_{int}\,[v_3]\,, v_s \leftarrow v_5\}$$

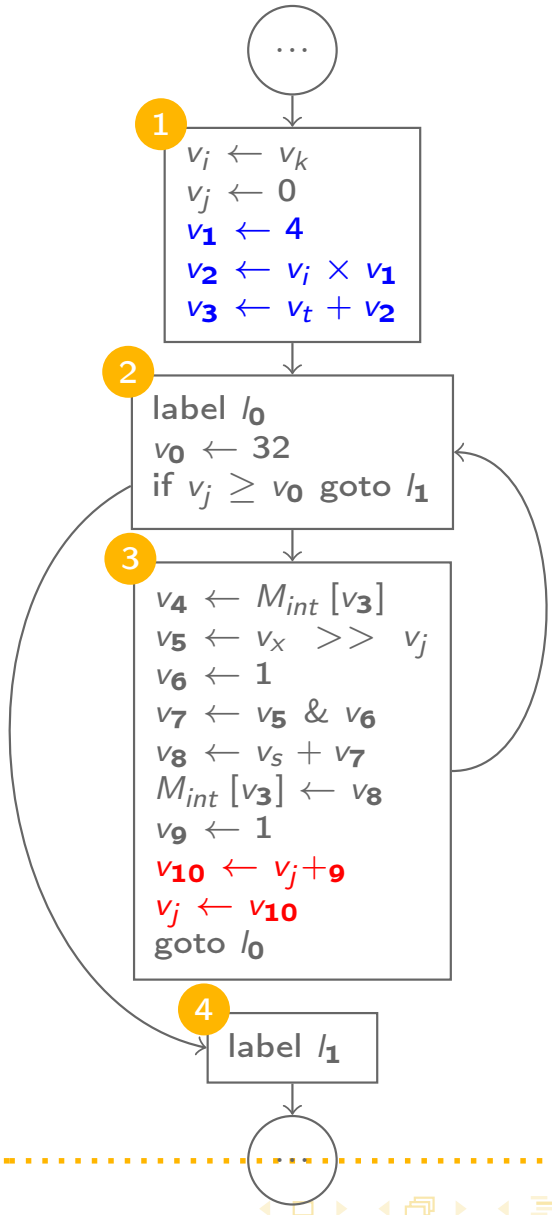
$$IN_6 = \emptyset$$

extraction d'invariant



extraction d'invariant

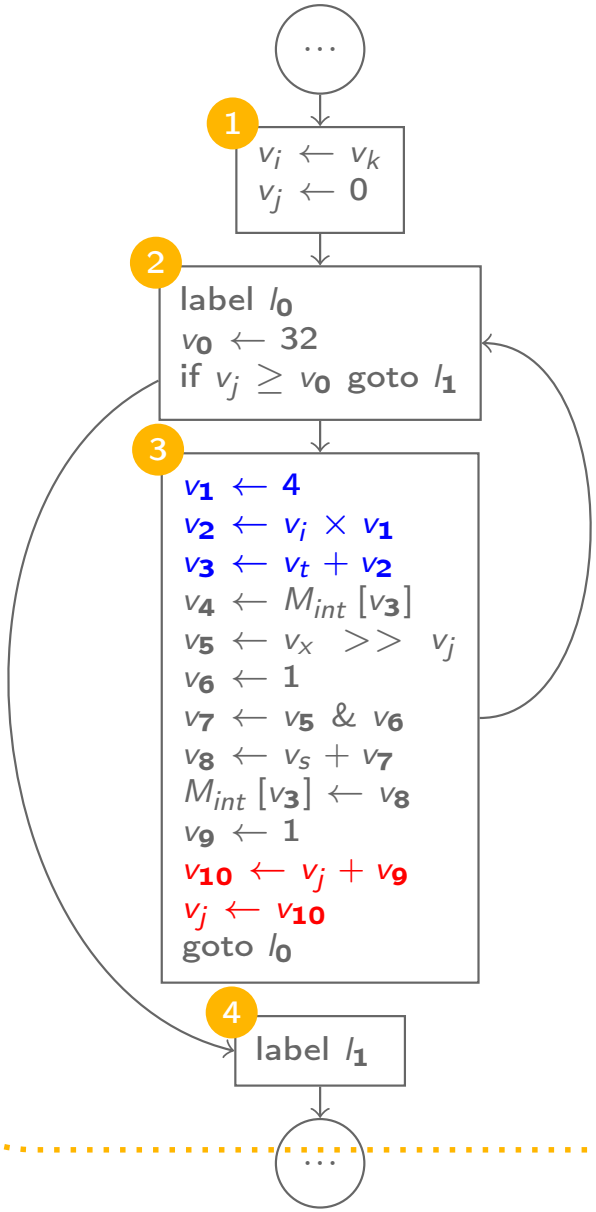




chaînes utilisation-définition

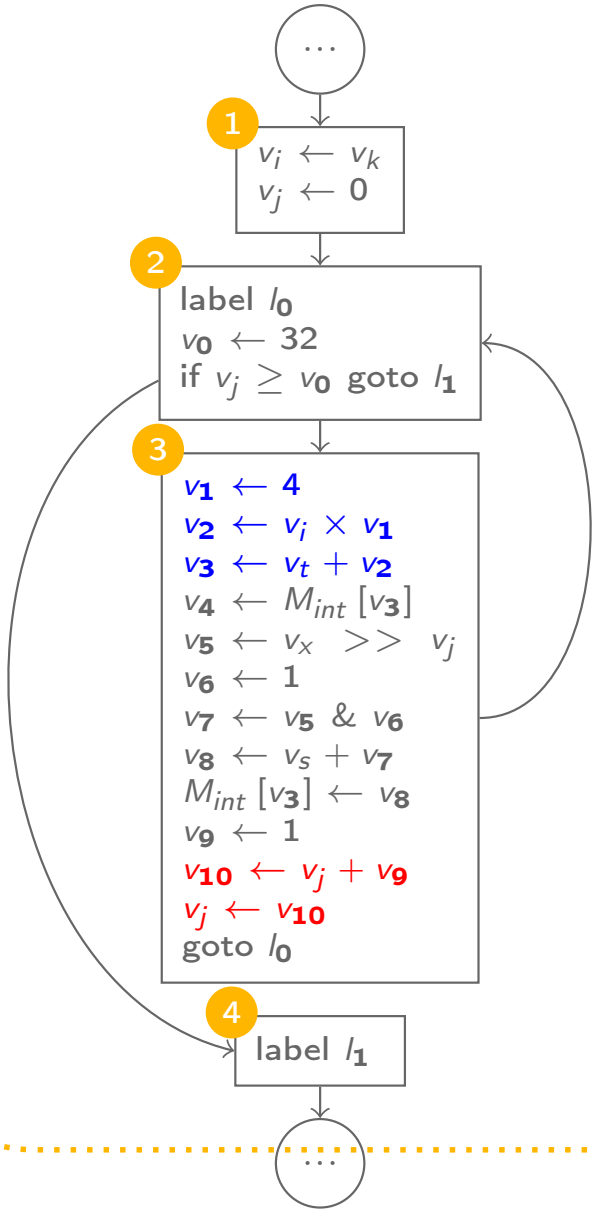
- Quels quadruplets définissent un registre ?
- définition des registres quel que soit le chemin
- $\mathcal{D} \subset Q$ – quadruplets du programme
- $KILL_b = \{v \leftarrow e \in \mathcal{D} \mid \exists v \leftarrow e' \in b\}$
- $GEN_b = \{v \leftarrow e \mid v \leftarrow e \in b\}$
- $IN_b = \bigcup_{b' \in PRED(b)} OUT_b$
  définition quels que soient les chemins
- $OUT_b = (IN_b \setminus KILL_b) \cup GEN_b$
- $v_i \leftarrow v_j \ \omega \ v_k$ dans b est invariant $\iff$
  toutes les définitions de v_j et v_k dans IN_b sont hors de la boucle
- treillis $(2^{\mathcal{D}}, \subseteq, \cup, \cap, \emptyset, \mathcal{D})$

Optimisation : *extraction d'invariant*



$$IN_3 = \{v_i \leftarrow v_k, v_j \leftarrow v_k, ...\}$$

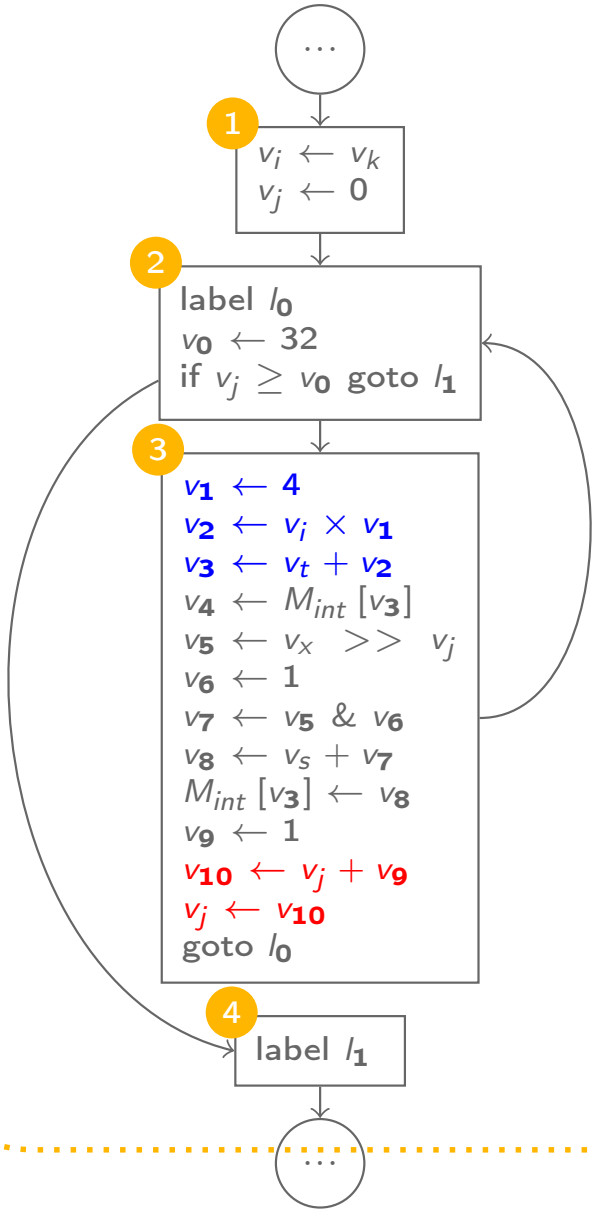
Optimisation : *extraction d'invariant*



$IN_3 = \{v_i \leftarrow v_k, v_j \leftarrow v_k, ...\}$

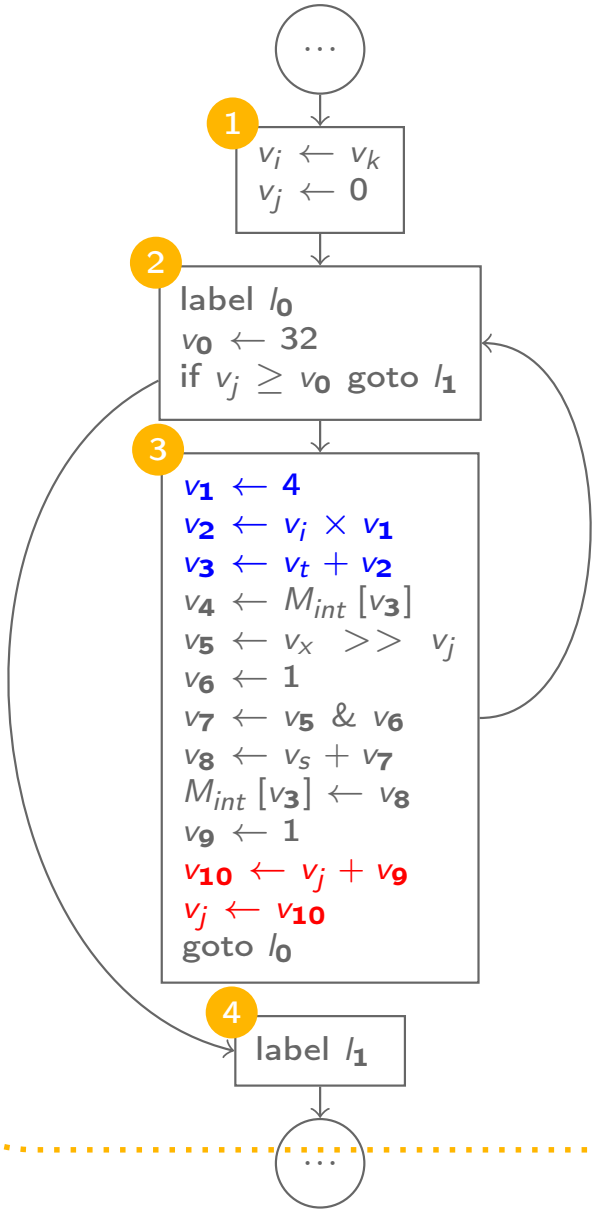
$v_1 \leftarrow 4$ invariant (par définition)

Optimisation : *extraction d'invariant*



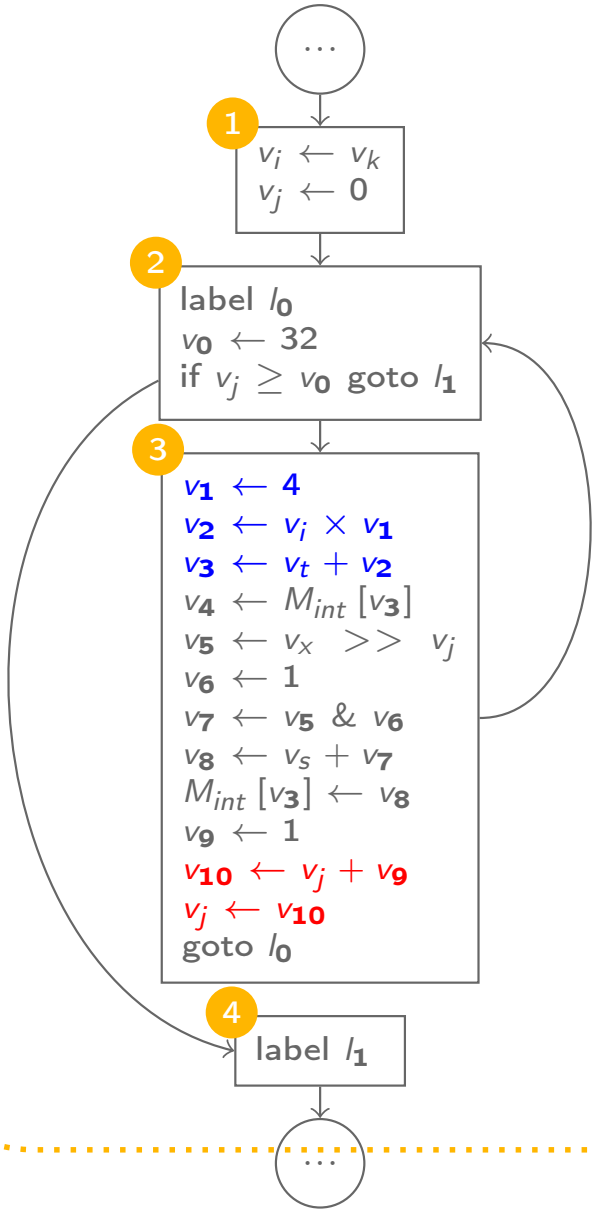
$IN_3 = \{v_i \leftarrow v_k, v_j \leftarrow v_k, ...\}$
$v_1 \leftarrow 4$ invariant (par définition)
$IN_3 = IN_3 \cup \{v_1 \leftarrow 4\}$

Optimisation : *extraction d'invariant*



$IN_3 = \{v_i \leftarrow v_k, v_j \leftarrow v_k, ...\}$
$v_1 \leftarrow 4$ invariant (par définition)
$IN_3 = IN_3 \cup \{v_1 \leftarrow 4\}$
$v_2 \leftarrow v_i \times v_1$ invariant car
$v_1 \leftarrow 4$ et $v_i \leftarrow 0$ sont hors boucle !

Optimisation : *extraction d'invariant*



$IN_3 = \{v_i \leftarrow v_k, v_j \leftarrow v_k, ...\}$
$v_1 \leftarrow 4$ invariant (par définition)
$IN_3 = IN_3 \cup \{v_1 \leftarrow 4\}$
$v_2 \leftarrow v_i \times v_1$ invariant car $v_1 \leftarrow 4$ et $v_i \leftarrow 0$ sont hors boucle !

$v_{10} \leftarrow v_j + v_9$ non-invariant car v_j défini par :

- $v_j \leftarrow 0$ hors boucle
- $v_j \leftarrow v_{10}$ dans la boucle

Calcul de domination

- $\forall b, b' \in V,\ b\ dom\ b' \iff \forall p \in \nu \rightarrow^* b' \implies b \in p$
- $IN_b =$ ensemble des dominants de $b \implies$
  $b\ dom\ b' \iff b \in IN_{b'}$
- $\mathcal{D} = 2^V$
- $KILL_b = \emptyset$
- $GEN_b = \{b\}$ – un BB se domie lui-même
- $IN_b = \bigcap_{b' \in PRED(b)} OUT_{b'}$
- treillis $(2^V, \supseteq, \cap, \cup, V, \emptyset)$

Calcul de vivacité

- $\mathcal{D} = \mathcal{V}$ – registres virtuels du programme
- v dans b est vivante s'il y a une utilisation de v dans les chemins partant de de b
- astuce : parcourir G en arrière
- $KILL_b = \{v \in \mathcal{V} \mid v \leftarrow e \in b\}$
- $GEN_b = \{v \in \mathcal{V} \mid v' \leftarrow f(v) \in b\}$
- $IN_b = \bigcup_{b' \in SUCC(b)} OUT_{b'}$
  v vivante quel que soit le chemin
- test de vivacité de v : $\forall b \in V, alive(v, b) \iff v \in IN_b$
- treillis $(2^{\mathcal{V}}, \subseteq, \cup, \cap, \emptyset, \mathcal{V})$

Construction de treillis

- ordre plat sur $\mathcal{S}$: $\langle \mathcal{S} \cup \{\bot, \top\}, \sqsubseteq, \cup, \cap, \bot, \top \rangle$ treillis !
  $\forall x \in \mathcal{S} \cup \{\bot, \top\}, \bot \sqsubseteq x \wedge x \sqsubseteq \top$
- soit 2 treillis $\langle \mathcal{S}_A, \sqsubseteq_A, \sqcup_A, \sqcap_A, \top_A, \bot_A \rangle$ et
  $\langle \mathcal{S}_B, \sqsubseteq_B, \sqcup_B, \sqcap_B, \top_B, \bot_B \rangle$ alors
  $\langle \mathcal{S}_A \times \mathcal{S}_B, \sqsubseteq, \sqcup, \sqcap, (\top_A, \top_B), (\bot_A, \bot_B) \rangle$ treillis !
  $\forall (a, b), (a', b') \in \mathcal{S}_A \times \mathcal{S}_B$,
  - $(a, b) \sqsubseteq (a', b') \iff (a \neq a' \wedge a \sqsubseteq_A a') \vee (a = a' \wedge b \sqsubseteq_A b')$
  - $(a, b) \sqcup (a', b') = (a \sqcup_A a', b \sqcup_B b')$
  - $(a, b) \sqcap (a', b') = (a \sqcap_A a', b \sqcap_B b')$
- soit un treillis $\langle \mathcal{S}, \sqsubseteq, \sqcup, \sqcap, \top, \bot \rangle$ et un ensemble $\mathcal{X}$,
  $\langle \mathcal{F} = 2^{\mathcal{X} \to \mathcal{S}}, \sqsubseteq^*, \sqcup^*, \sqcap^*, \lambda x.\top, \lambda x.\bot \rangle$ treillis
  $\forall f_1, f_2 \in \mathcal{F}$,
  - $f_1 \sqsubseteq f_2 \iff \forall x \in \mathcal{X}, f_1 \sqsubseteq^* f_2$
  - $f_1 \sqcup^* f_2 = \lambda x. f_1(x) \sqcup f_2(x)$, $f_1 \sqcap^* f_2 = \lambda x. f_1(x) \sqcap f_2(x)$

Bilan : optimisation

Optimisation :

- = transformation de programme
- qui conserve la sémantique du programme
- qui vérifie certaines conditions
- condition = propriétés obtenues par des analyses du programme

Analyse du programme (analyse statique) :

- propriété = { ensemble de valeurs possibles }
- analyser tous les chemins en parallèle
- usage d'un treillis $\Longrightarrow$ méthode de calcul et preuve de terminaison

Bilan : *Analyse Itérative de Flot de Donnée*

- définition du domaine $\mathcal{D}$
- définition de $KILL_b$ et GEN_b
  $OUT_b = (IN_b \setminus KILL_b) \cup GEN_b$
- sens de parcours du CFG
  - en avant : $IN_b = \sqcup_{b' \in PRED(b)} OUT_b$
  - en arrière : $IN_b = \sqcup_{b' \in SUCC(b)} OUT_b$
- agrégation au niveau des chemins
  - sur au moins un des chemins (analyse *MAY*)
    $\sqcup = \cup \implies (\mathcal{D}, \subseteq, \cup, \cap, \emptyset, \mathcal{D})$
  - sur tous les chemins (analyse *MUST*)
    $\sqcup = \cap \implies (\mathcal{D}, \supseteq, \cap, \cup, \mathcal{D}, \emptyset)$

Bilan : on sait tout faire ?

- Non !
- problème : ensemble infinis $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$, ...

```
int sum(int n) {
  int s = 0;
  while(n > 0) {
    s += n;
    n--;
  }
  return s;
}
```

  Valeurs de s ou n ?

- ensembles abstraits $\Rightarrow$ *Interprétation Abstraite* (Cousot, 1978)

Optimisation des sous-programmes

- *tail call optimization* – appel de sous-programme en fin $\Longrightarrow$ branchement simple
- *inlining* – remplacement d'un appel par le code du sous-programme
  - dirigé par inline
  - taille du sous-programme $<$ seuil
- *leaf routine optimization* – simplification des sous-programme ne faisant pas d'appel
  - SP constant $\Longrightarrow$ FP non utilisé
  - pas de sauvegarde du LR
- *recursion elimination* – remplacement par une boucle si possible (langages fonctionnels – OCAML)

Pipeline : ré-ordonnancement des instructions

Version 1

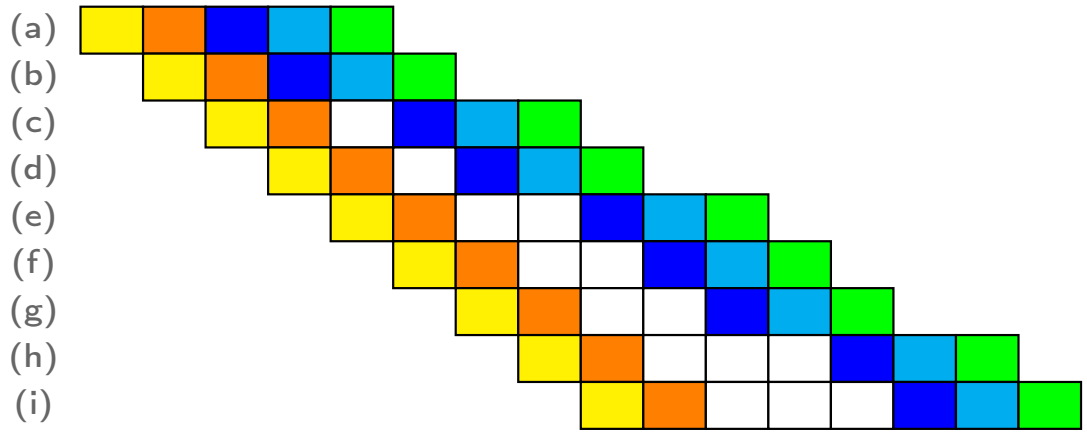
(a) LDR $R_1, [\mathrm{FP}, \#4]$
(b) LDR $R_2, [\mathrm{FP}, \#16]$
(c) ADD R_3, R_1, R_2
(d) LDR $R_4, [\mathrm{FP}, \#20]$
(e) SUB R_5, R_3, R_4
(f) STR $R_5, [\mathrm{FP}, \# - 8]$
(g) LDR $R_6, [\mathrm{FP}, \#16]$
(h) CMP $R_6, \#0$
(i) BNE l_3

Version 2

(a) LDR $R_1, [\mathrm{FP}, \#4]$
(b) LDR $R_2, [\mathrm{FP}, \#16]$
(d) LDR $R_4, [\mathrm{FP}, \#20]$
(g) LDR $R_6, [\mathrm{FP}, \#16]$
(c) ADD R_3, R_1, R_2
(h) CMP $R_6, \#0$
(e) SUB R_5, R_3, R_4
(f) STR $R_5, [\mathrm{FP}, \# - 8]$
(i) BNE l_3

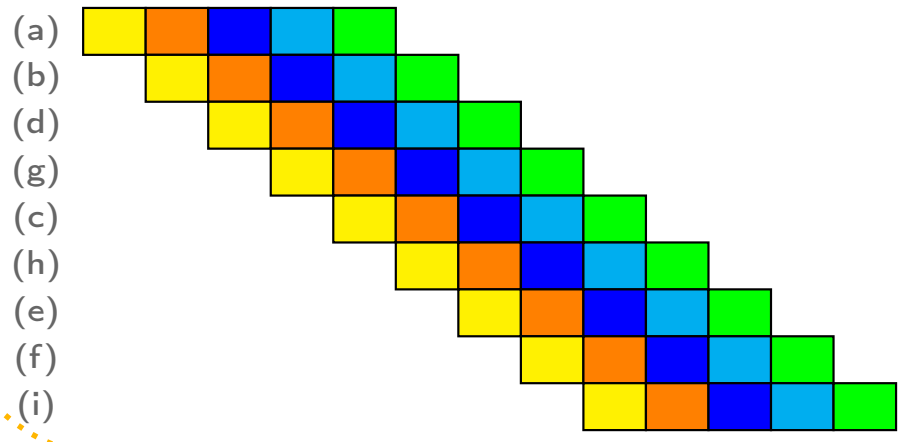
Exécution dans le pipeline

pipeline : FE DE EX ME WB

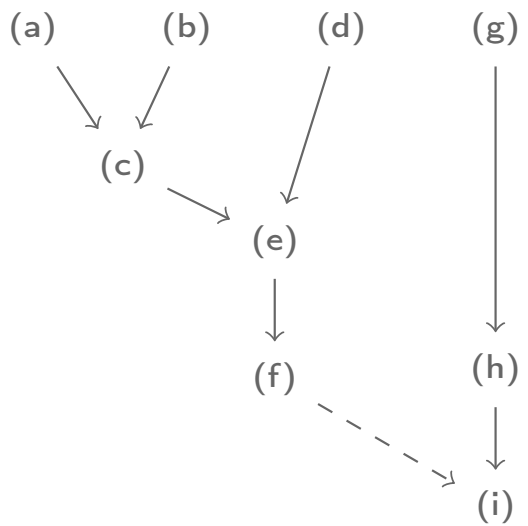


Version 1 (16 cycles)

Version 2 (13 cycles)

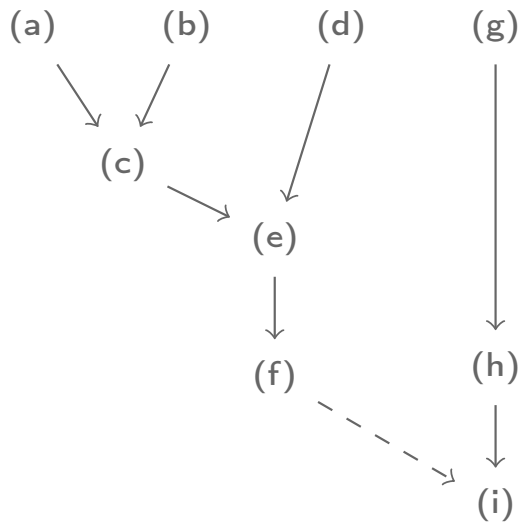


Ordonnancement par liste



- graphe de dépendance (DAG)
- sélection d'une instruction après ses dépendances
- sélection en largeur d'abord

Table de réservation



FE	DE	EX	ME	WB
(a)				
(b)	(a)			
(d)	(b)	(a)		
(g)	(d)	(b)	(a)	
(c)	(g)	(d)	(b)	(a)
(h)	(c)	(g)	(d)	(b)
(e)	(h)	(c)	(g)	(d)
(f)	(e)	(h)	(c)	(g)
(i)	(f)	(e)	(h)	(c)
	(i)	(f)	(e)	(h)
		(i)	(f)	(e)
			(i)	(f)
				(i)

Extraction de parallélisme

processeur superscalaire

- dans l'ordre
- dans le désordre (*reorder buffer*)

VLIW (*Very Long Instruction Word*)

- instructions parallèles groupées en *bundle*

Plus il existe de parallélisme
$\Longrightarrow$ efficacité de fonctionnement – ILP (*Instruction Level Parallelism*)
$\Longrightarrow$ augmenter la taille des blocs

Augmentation du parallélisme

Superbloc :

- agréger des blocs proche
- bloc suivant une sélection $\Longrightarrow$ dupliqué sur chaque branche

Déroulage de boucle :

- dupliquer une boucle N fois
- si taille inconnue, ajouter boucle pour itération restantes

Attention : augmentation de la taille du programme $\Longrightarrow$ inefficacité du cache d'instruction

Cache d'instruction

- conflit si 2 blocs de code correspondant à la même ligne de cache
- blocs proche ne font pas de conflit
- rapprocher les blocs / fonctions dont l'exécution est proche
- précharger un bloc avant son exécution (instruction spéciale $\Longrightarrow$ augmentation de la taille du code)

NOTE : toute optimisation réduisant la taille du code améliore le cache d'instruction

Cache de donnée

Transformations de boucle (grands tableaux) :

- fusion de boucle
- *tiling*
- restructuration

Analyse complexes :

- dépendances inter-itérations

```
for(i = 0; i < N; i++)
  t[i] = a[i] + b[i];
for(j = 0; j < N; j++)
  t[i] = t[i] + 1;
```

```
for(i = 0; i < N; i++) {
  t[i] = a[i] + b[i];
  t[i] = t[i] + 1;
}
```