Hugues Cassé <casse@irit.fr>
M1 SIAME - FSI - Université de Toulouse

Cours 4 :
sélection des instructions et allocation des registres
COMC

Introduction

Résumé du processus :

1. front-end : langage source $\rightarrow$ AST
2. génération de code : AST $\rightarrow$ quadruplets

Dan ce cours :

- sélection des instructions :
  quadruplets $\rightarrow$ instructions machine
- allocation des registres :
  registres virtuels $\rightarrow$ registres machine

Plan

Sélection des instructions

$v_i \leftarrow v_j +_{int} v_k$ se traduit en $ADD\ v_i, v_j, v_k$

$v_i \leftarrow c$
$v_j \leftarrow fp + v_i$ se traduit en $LDR\ v_k, [fp, \#c]$
$v_k \leftarrow M_{int}[v_j]$

- Problème 1 : les quadruplets peuvent être intercalés !
- $\Rightarrow$ réorganiser l'ordre des instructions
- Problème 2 : préserver la sémantique du programme !

Dépendance de donnée

$\{x = 0 \ \wedge \ a = 2 \ \wedge \ b = 3\}$

```
x = a + b;
a = a + 1;
```

$\{x = 5 \ \wedge \ a = 3 \ \wedge \ b = 3\}$

$\Downarrow$

$\{x = 0 \ \wedge \ a = 2 \ \wedge \ b = 3\}$

```
a = a + 1;
x = a + b;
```

$\{x = 6 \wedge a = 3 \ \wedge \ b = 3\}$

Relation de précedence

$$\forall q_a, q_b \in Q^*, q_a \prec q_b \Leftrightarrow a < b \ \wedge \begin{cases} & def(q_a) \cap use(q_b) \neq \emptyset \\ \vee & use(q_a) \cap def(q_b) \neq \emptyset \end{cases}$$

avec $a, b \in \mathbb{N}$, $a < b$, numéro dans la séquence de quadruplets

Règle

préservation de $\prec$ lors de changement d'ordre des quadruplets
$\Leftrightarrow$ préserver la sémantique du programme !

Ordre et accès mémoire

```
int x = 10, *p;
p = f(&x);
*p = 0;       // (a)
x = x + 1;    // (b)
```

- Puis-je changer l'ordre de (a) et (b) ?
- Ca dépend de la valeur retour de `f` ?
- Difficile de déterminer la valeur des pointeurs.

Règle

- les écritures doivent être gardés dans l'ordre
- les lectures doivent rester ordonnées / écritures
- les lectures n'ont pas besoin d'être ordonnées

Relation de précédence

$$\forall q_a, q_b \in Q^*, q_a \prec q_b \Leftrightarrow a < b \ \wedge$$

$$\begin{cases} & def(q_a) \cap use(q_b) \neq \emptyset \\ \vee & use(q_a) \cap def(q_b) \neq \emptyset \\ \vee & q_b = M_\tau[v_i] \leftarrow v_j \ \wedge \ q_a = M_{\tau'}[v_k] \leftarrow v_l \\ \vee & q_b = M_\tau[v_i] \leftarrow v_j \ \wedge \ q_a = v_k \leftarrow M_{\tau'}[v_l] \end{cases}$$

Instructions de branchements

- label l, goto l, if $v_i\ \omega_c\ v_j$ goto $l \Rightarrow$ ne pas réordonnancer !
- Limites de réordonnancement ?

Definition

Bloc de base (BB) = séquence de quadruplets dont

- seul le premier peut être une étiquette
- seul le dernier peut être un branchement

Graphe de flot de contrôle

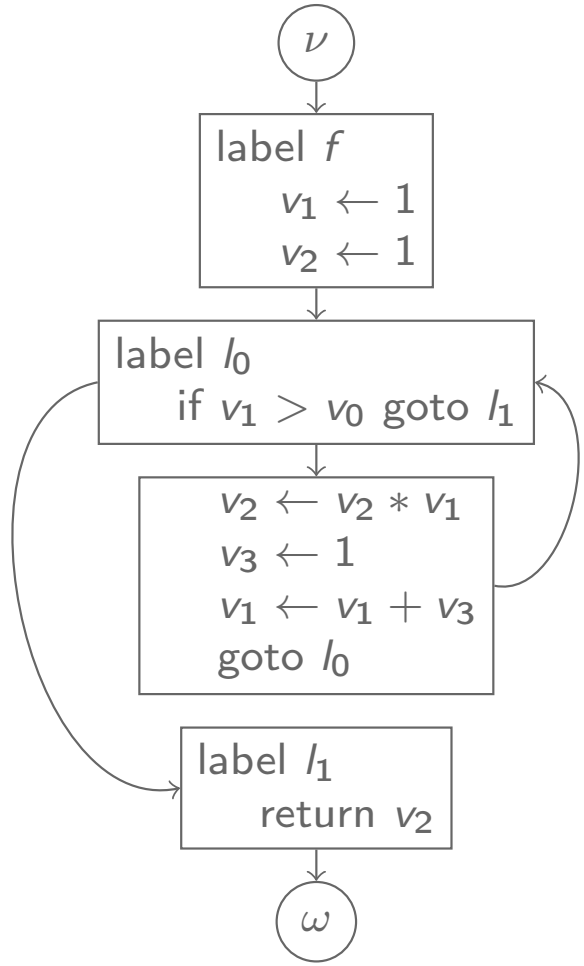
Definition

graphe de flot de contrôle (CFG) $G = \langle V, E, \nu, \omega \rangle$ tel que :

- V – ensemble des blocs de base
- $E \subseteq V \times V$ – ensemble des arcs $v \to w$, flot de contrôle
- $\nu, \omega \in V$ – pseudo-BB unique point d'entrée / de sortie de G

Exemple de CFG

```
label f
    v1 ← 1
    v2 ← 1
label l0
    if v1 > v0 goto l1
    v2 ← v2 * v1
    v3 ← 1
    v1 ← v1 + v3
    goto l0
label l1
    return v2
```



Relation de dépendance

$$\forall q_a, q_b \in Q^*, q_a \prec q_b \Leftrightarrow a < b \ \wedge$$

$$\begin{cases} & def(q_a) \cap use(q_b) \neq \emptyset \\ \vee & use(q_a) \cap def(q_b) \neq \emptyset \\ \vee & q_b = M_\tau[v_i] \leftarrow v_j \ \wedge \ q_a = M_{\tau'}[v_k] \leftarrow v_l \\ \vee & q_b = M_\tau[v_i] \leftarrow v_j \ \wedge \ v_k \leftarrow M_{\tau'}[v_l] \\ \vee & q_a = \text{label } l \\ \vee & q_b = \text{goto } l \\ \vee & q_b = \text{if } v_i \ \omega_c \ v_j \text{ goto } l \end{cases}$$

Exercice 1

Re-construisez le CFG du programme ci-dessous et, en préservant la précédence, faite la traduction en assembleur ARM.

label f

$v_0 \leftarrow 0$

$v_1 \leftarrow 0$

label l_0

$v_2 \leftarrow 100$

if $v_1 \geq v_2$ goto l_2

$v_3 \leftarrow 0x3000$

$v_4 \leftarrow 4$

$v_5 \leftarrow v_1 \times v_4$

$v_6 \leftarrow v_3 + v_5$

$v_7 \leftarrow M_{int}[v_6]$

$v_8 \leftarrow 0$

if $v_7 < v_8$ goto l_3

$v_9 \leftarrow 0x3000$

$v_{10} \leftarrow 4$

$v_{11} \leftarrow v_1 \times v_{10}$

$v_{12} \leftarrow v_9 + v_{11}$

$v_{13} \leftarrow M_{int}[v_{12}]$

$v_0 \leftarrow v_0 + v_{13}$

label l_3

$v_{14} \leftarrow 1$

$v_1 \leftarrow v_1 + v_{14}$

goto l_0

label l_2

return v_0

Plan

Utilisation d'un DAG

Definition

Un graphe acyclique dirigé (DAG) est un graphe $G_{DAG} = \langle V_{DAG}, E_{DAG} \rangle$ tel que :

$V_{DAG} = \mathbb{Z}_n \cup \mathbb{F}_n\ \Omega_1 \cup \Omega_2 \cup \mathcal{V} \cup \langle M_\tau\ [\ ] \rangle \cup \langle M_\tau\ [\ ] \leftarrow \rangle$ – constantes, opérateurs, accès mémoire

$E_{DAG} \subseteq V_{DAG} \times V_{DAG}$ – relation de précédence $\prec$

$\forall e \in E_{DAG}, weak(e) =$

$\top$ dépendance par opérande (arc plein)

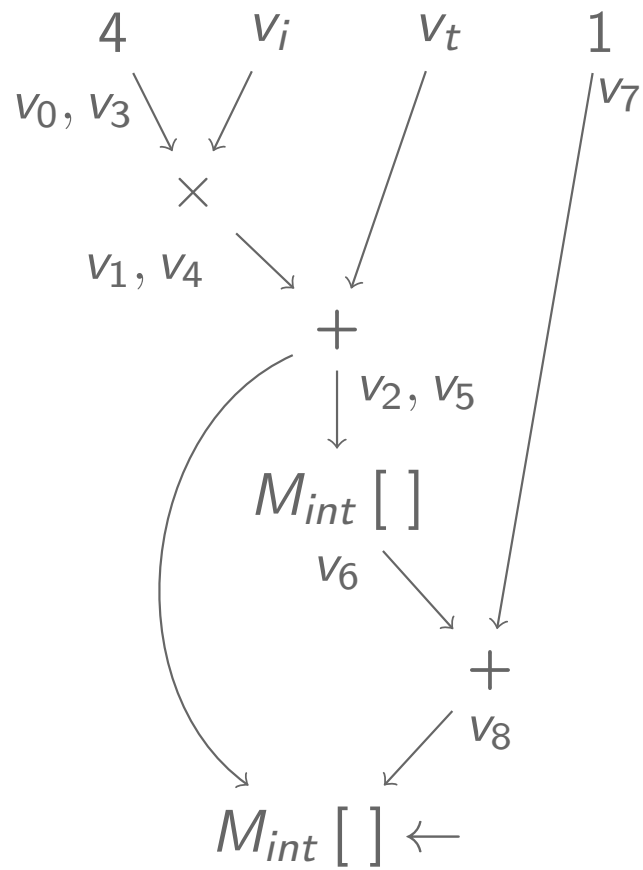
$\bot$ dépendance d'ordre simple (arc pointillé)

- 1 DAG par BB
- résum" les calculs du DAG
- $\prec$ visible !

Exemple de DAG

```
t[i] = t[i] + 1;
```

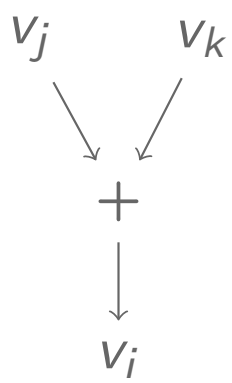
$$
\begin{aligned}
v_0 &\leftarrow 4 \\
v_1 &\leftarrow v_i \times v_0 \\
v_2 &\leftarrow v_t + v_1 \\
v_3 &\leftarrow 4 \\
v_4 &\leftarrow v_i \times v_3 \\
v_5 &\leftarrow v_t + v_4 \\
v_6 &\leftarrow M_{int}[v_5] \\
v_7 &\leftarrow 1 \\
v_8 &\leftarrow v_6 + v_7 \\
M_{int}[v_2] &\leftarrow_8
\end{aligned}
$$



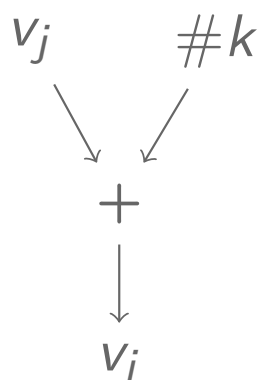
Sélection des instructions

1 instruction machine =

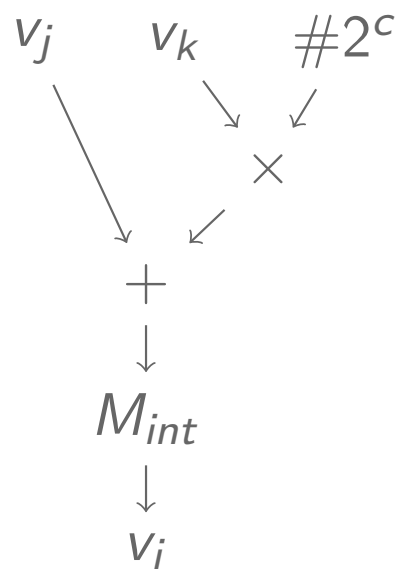
- 1 modèle d'instruction = sous-DAG
- registres ou des constantes en entrée
- registres en sortie



ADD v_i, v_j, v_k

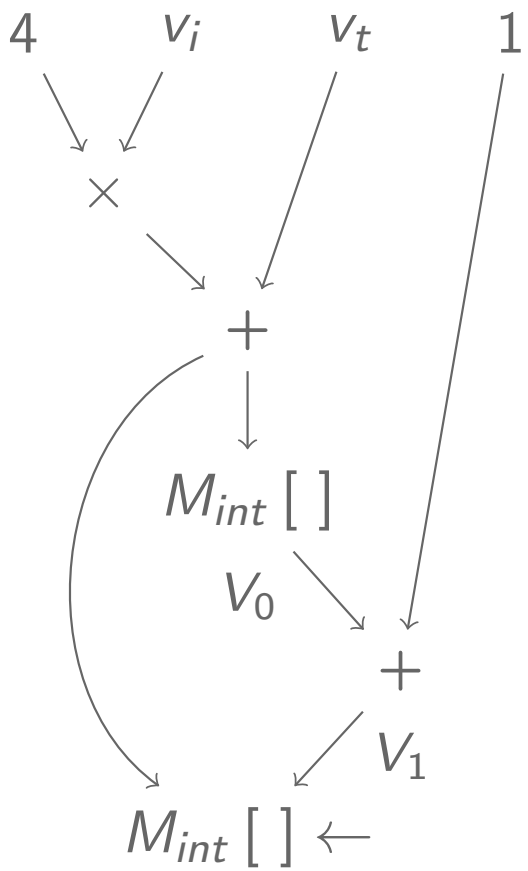


ADD $v_i, v_j, \#k$



LDR $v_i, [v_j, v_k, \#c]$

Reconnaissance des modèles



1. LDR $V_0, [v_t, v_i, LSL\#2]$
   $\{ v_i \mapsto V_0,\ v_j \mapsto v_t,\ v_k \mapsto v_i$ et
   $c \mapsto 2 \}$
2. ADD $V_1, V_0, \#1$
   $\{ v_i \mapsto V_1,\ v_j \mapsto V_0,\ k \mapsto 1 \}$
3. STR $V_1, [v_t, v_i, LSL\#2]$
   $\{ v_i \mapsto V_1,\ v_j \mapsto v_t,\ v_k \mapsto v_i$ et
   $c \mapsto 2 \}$

Reconnaissance

Règles :

- en fin, couverture totale des sommets
- ordre $\prec$ maintenu dans les instructions machines
- il doit existe un modèle pour chaque forme du DAG

Approches :

- force brute, ILP, SAT, ...
- assez coûteuse car appliqué à chaque BB

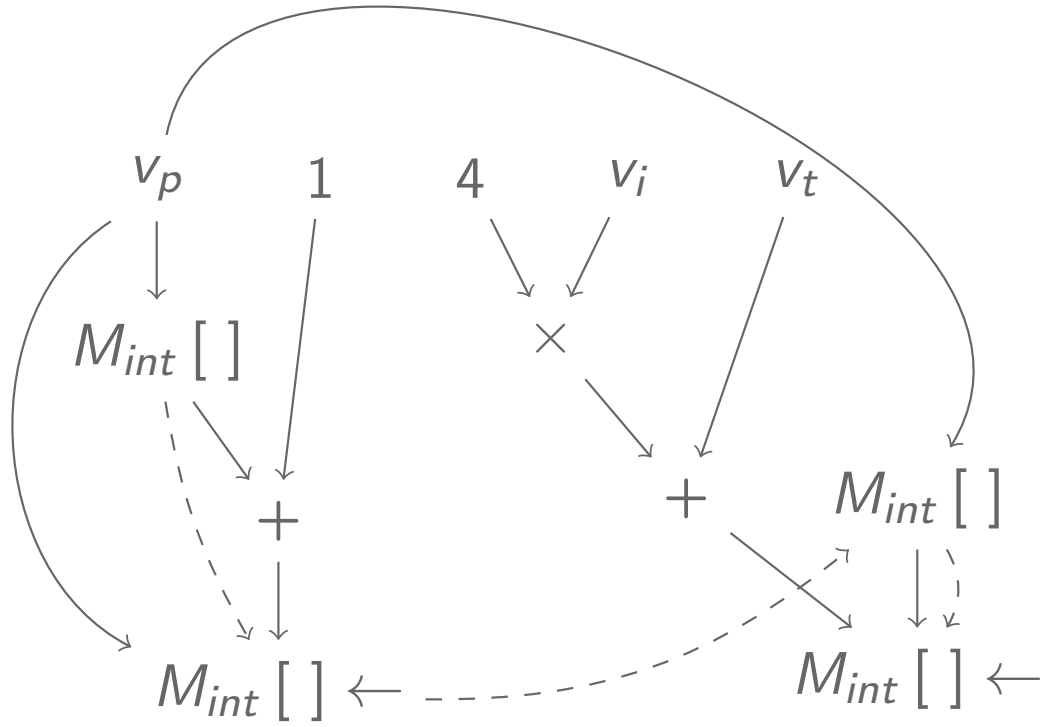
Approche par heuristique :

- classés les modèles par taille
- traversée du DAG par ordre topologique
- utiliser le premier modèle qui correspond

DAG et accès mémoire

```
*p = *p + 1;
t[i] = *p;
```

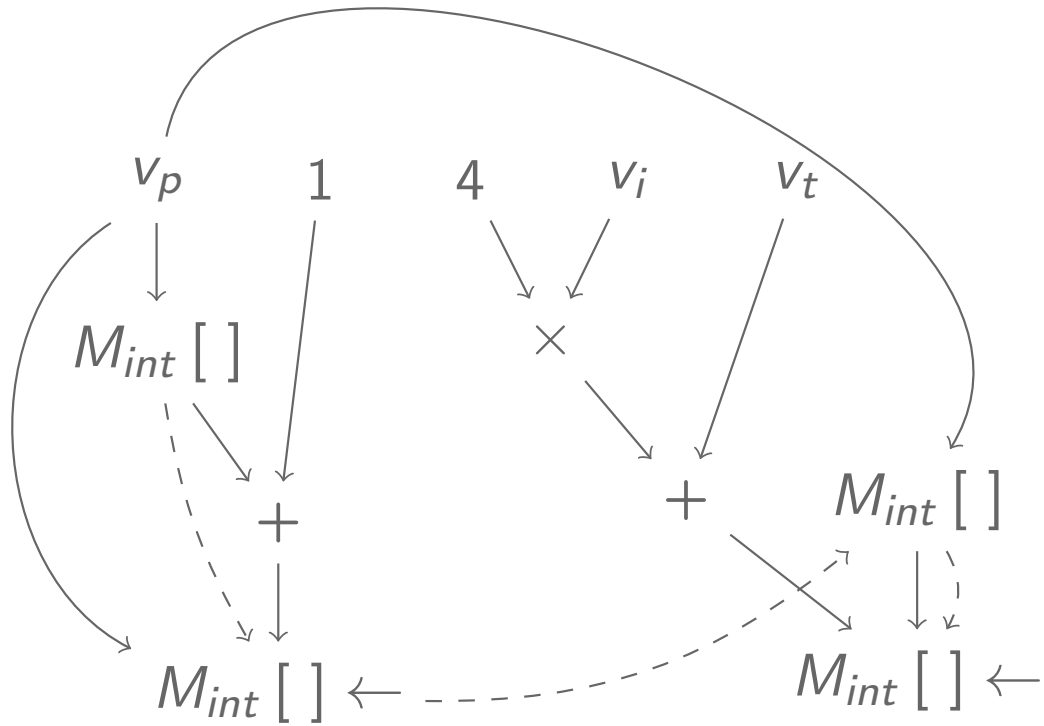
$$
\begin{aligned}
v_0 &\leftarrow M_{int}[v_p] \\
v_1 &\leftarrow 1 \\
v_2 &\leftarrow v_0 + v_1 \\
M_{int}[v_p] &\leftarrow v_2 \\
v_3 &\leftarrow 4 \\
v_4 &\leftarrow v_i \times v_3 \\
v_5 &\leftarrow v_t + v_4 \\
v_6 &\leftarrow M_{int}[v_p] \\
M_{int}[v_5] &\leftarrow v_6
\end{aligned}
$$



DAG et accès mémoire

```
*p = *p + 1;
t[i] = *p;
```

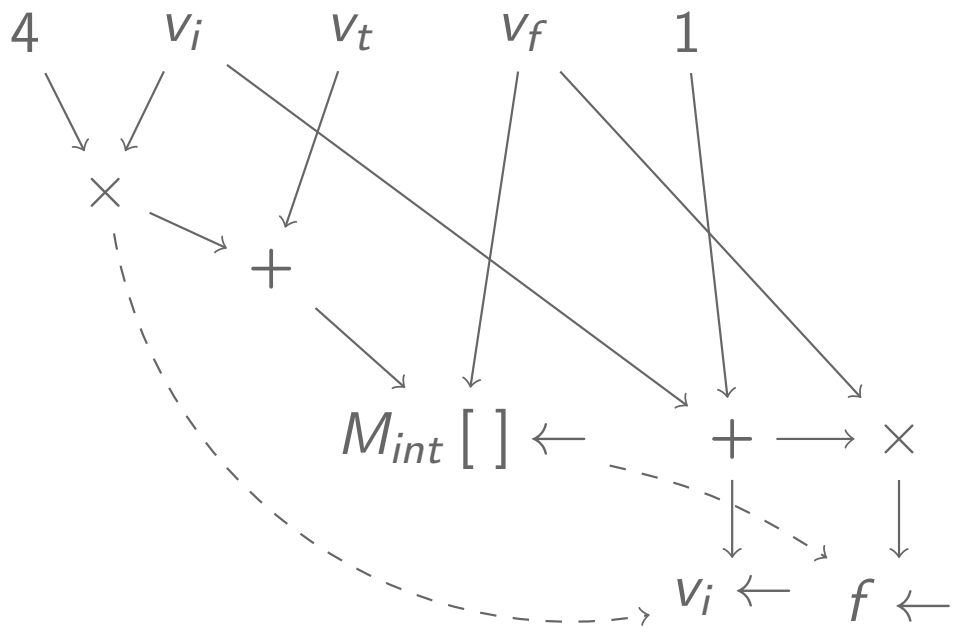
LDR $V_0, [v_p]$
ADD $V_1, V_0 \# 1$
STR $V_1, [v_p]$
LDR $V_0, [v_p]$
STR $V_1, [v_t, v_i, LSL \# 2]$



DAG et affectation de variable

```
t[i] = f;
i = i + 1;
f = f * i;
```

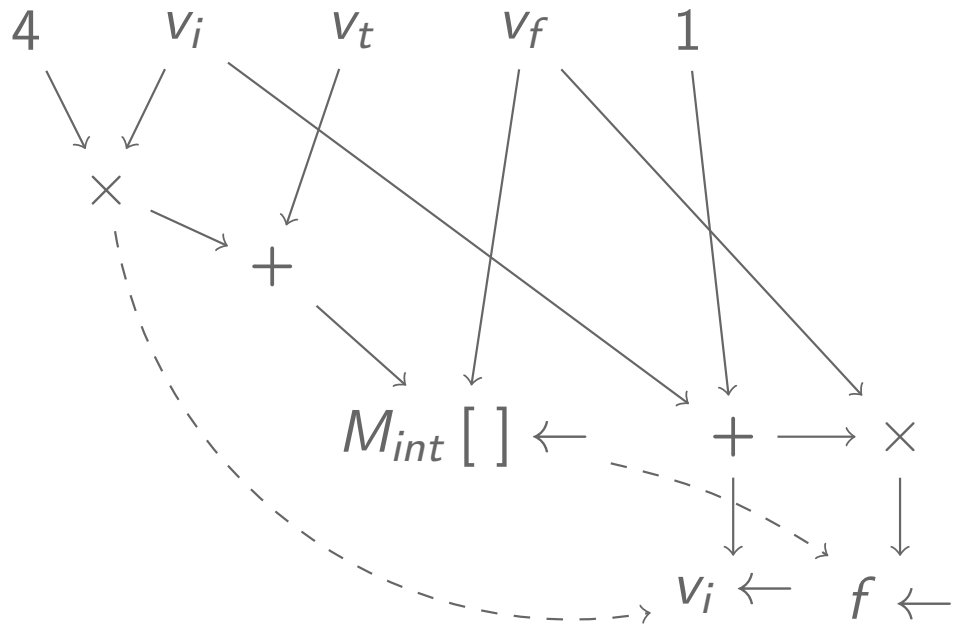
$v_0 \leftarrow 4$
$v_1 \leftarrow v_i \times v_0$
$v_2 \leftarrow v_t + v_1$
$M_{int}[v_2] \leftarrow v_f$
$v_3 \leftarrow 1$
$v_4 \leftarrow v_i + v_3$
$v_i \leftarrow v_4$
$v_5 \leftarrow v_f \times v_i$
$v_f \leftarrow v_5$



DAG et affectation de variable

```
t[i] = f;
i = i + 1;
f = f * i;
```

STR $v_f, [v_t, v_i, \#4]$
ADD $v_i, v_i, \#1$
MUL v_f, v_i, v_f



Approche DAG

- sélection optimale (ou presque des instructions)
- nombre de registres nécessaire diminué
- factorisation des calculs dans le BB
- adapter un jeu d'instruction $\Leftrightarrow$ écrire le modèle de chaque instruction

Approche utilisée dans de nombreux compilateurs (GCC, LLVM, etc).
Défaut : ne suffit pas pour traiter certaines instructions complexes comme les instructions vectorielles.

Exercice 2

1. Traduisez en quadruplets le code C ci-dessous :

```
t[i] = t1[i] + t2[i];
i = i + 1;
```

2. En utilisant un DAG effectuez la phase de sélection des instructions.

Exercice de synthèse

```
void invert(int n, int t[]) {
  i = 0;
  while(i < n) {
    t[i] = ((t[i] >> 16) & 0xffff) | (t[i] << 16);
    i++;
  }
}
```

1. Donner l'AST typé de cette fonction.
2. Donner la traduction en quadruplets en considérant que i, n et t sont placés dans les registres virtuels v_i, v_n et v_t.
3. En utilisant un DAG, procéder à la sélection des instruction pour le BB de la boucle.

Plan

Registres matériels

$\mathcal{V}$ ensemble infini de registres virtuels

$\mathcal{R}$ nombre limité de registres matériels

$R_{temp} \in \mathcal{R}$ registre temporaire réservé

Avec l'ARM :

PC	R_{15}
LR	R_{14}
SP	R_{13}
FP	R_{12}
R_{temp}	R_{11}
$\mathcal{R}$	$R_0..R_{10}$

Fonctions utiles

- $use : \mathbb{M} \mapsto 2^{\mathcal{V}}$ – registre utilisés
- $def : \mathbb{M} \mapsto 2^{\mathcal{V}}$ – registre définies
- $m\,[v \mapsto R]$ – remplacement dans m de v par R
- $restore(v, R)$ – code pour charger v de la mémoire dans le registre matériel R
- $save(v, R)$ – code pour sauvegarder en mémoire le registre virtuel v stocké dans le registre matériel R

Approche locale à un BB

Caractéristiques :

- lors de la traduction, variable locale $\Leftrightarrow$ registre virtuel
- avant un BB, les données sont stockées en mémoire
- après un BB, les variables locales modifiées $\Rightarrow$ stockées en mémoire

État utilisé pour l'analyse :

$F \subseteq \mathcal{R}$ registres matériels disponibles

$M \subseteq \mathcal{V}$ registres modifiés

$\mathcal{X} \subseteq \mathcal{V}$ registres contenant des variables

$\sigma : \mathcal{V} \mapsto \mathcal{R} \cup \{\bot\}$ allocation courante

Allocation d'un registre

v registre virtuel à allouer dans un registre réel

m_i instruction courant

$alloc(v, m_i) =$

if $\sigma[v] \neq \bot$ **then**
  return $\sigma[v]$
else if $F = \emptyset$ **then**
  return $spill(v, m_i)$
else
  $\underline{\text{let}}\ R_i \in F\ \underline{\text{in}}$
  $F \leftarrow F \setminus \{R_i\}$
  $\sigma \leftarrow \sigma[v \rightarrow R_i]$
  if $v \in use(m_i)$ **then**
    $restore(v, R)$
  end if
  return R_i
end if

Reconstruction du BB

B BB à allouer

B' BB après allocation

$B' \leftarrow [\,]$; $F \leftarrow \mathcal{R}$
$\sigma \leftarrow \lambda v.\bot$; $M \leftarrow \emptyset$
for all $m_i \in B$ **do**
  for all $v \in use(m)$ **do**
    $m_i \leftarrow m_i\,[v \mapsto alloc(v, m_i)]$
  end for
  $free(i)$
  $M \leftarrow M \cup def(m_i)$
  for all $v \in def(m)$ **do**
    $m_i \leftarrow m_i\,[v \mapsto alloc(v, m_i)]$
  end for
  $B' \leftarrow B'$ @ $[m_i]$
end for
for all $v \in M$ **do**
  $save(v, \sigma[v])$
end for

Exemple

	$F = \{R_{0-10}\}$, $\sigma = \emptyset$
	LDR $R_0, [SP, \#k_i]$
ADD $v_0, v_i, \#1$	ADD $R_1, R_0, \#1$
	$F = \{R_{2-10}\}$, $\sigma = \{v_i \mapsto R_0, v_0 \mapsto R_1\}$
	LDR $R_2, [SP, k_t]$
LDR $v_1, [v_t, v_0, \text{LSL}\#2]$	LDR $R_3, [R_2, R_1, \text{LSL}\#2]$
	$F = \{R_{4-10}\}$, $\sigma = \{..., v_t \mapsto R_2, v_1 \mapsto R_3\}$
SUB $v_2, v_1, \#1$	SUB $R_4, R_3, \#1$
	$F = \{R_{5-10}\}$, $\sigma = \{..., v_2 \mapsto R_4\}$
STR $v_2, [v_t, v_0, \text{LSL}\#2]$	STR $R_4, [R_2, R_1, \text{LSL}\#2]$
	$F = \{R_{5-10}\}$, $\sigma = \{v_i \mapsto R_0, v_0 \mapsto R_1,$
	$v_t \mapsto R_2, v_1 \mapsto R_3, v_2 \mapsto R_4\}$

Ré-utilisation des registres

Problème : des registres devenus inutile non-libérés.

Definition

$v \in \mathcal{V}$ est vivant après une instruction m_i dans un BB b ssi $\exists m_j \in b \ \wedge \ i < j \ \wedge \ v \in use(m_j)$.

Soit D_i ensemble des registres mort après m_i.

$free(i) =$

$F \leftarrow F \cup D_i$
for all $v \in M \cap D_i$ **do**
  if $v \in \mathcal{X}$ **then**
    $save(v, \sigma[v])$;
  end if
  $\sigma \leftarrow \sigma[v \rightarrow \bot]$
end for
$M \leftarrow M \setminus D_i$

Exemple (avec *free*)

	$F = \{R_{0-10}\}$, $\sigma = \{\}$	
	LDR $R_0, [SP, \#k_i]$	
ADD $v_0, v_i, \#1$	ADD $R_0, R_0, \#1$	$v_i \mapsto R_0$ puis libéré
	$F = \{R_{1-10}\}$, $\sigma = \{v_0 \mapsto R_0\}$	
	LDR $R_1, [SP, k_t]$	
LDR $v_1, [v_t, v_0, \text{LSL}\#2]$	LDR $R_2, [R_1, R_0, \text{LSL}\#2]$	v_t vivant
	$F = \{R_{3-10}\}$, $\sigma = \{..., v_t \mapsto R_1, v_1 \mapsto R_2\}$	
SUB $v_2, v_1, \#1$	SUB $R_2, R_2, \#1$	$v_1 \mapsto R_2$ libéré puis utilisé
	$F = \{R_{3-10}\}$, $\sigma = \{..., v_2 \mapsto R_2\}$	
STR $v_2, [v_t, v_0, \text{LSL}\#2]$	STR $R_2, [R_1, R_0, \text{LSL}\#2]$	v_2, v_t, v_0 morts
	$F = \{R_{0-10}\}$, $\sigma = \{\}$	

Vidage de registre

- Plus aucun registre disponible ?
- Vider un registre – *spilling*
- Lequel : celui dont l'utilisation est la plus tardive, v'
- Vider v', réutiliser $\sigma\,[v']$ pour v

$spill(v, m_i) =$

$\underline{\text{let}}\ R_i = \sigma\,[v']\ \underline{\text{in}}$
if $v' \in M$ **then**
  $save(v', R_i)$
  $M \leftarrow M \setminus \{v'\}$
end if
if $v \in use(m_i)$ **then**
  $restore(v, R_i)$
end if
$\sigma \leftarrow \sigma\,[v \rightarrow R_i]$
return R_i

Exercice

1. En utilisant la méthode locale, réalisez l'allocation du BB ci-dessous avec $F = \{R_{0-10}\}$.
2. Refaite l'allocation des registres en considérant que $F = \{R_{0-2}\}$.

Le BB de base est :

LDR $v_0, [v_{t1}, v_i, LSL\#2]$
LDR $v_1, [v_{t2}, v_i, LSL\#2]$
MUL v_3, v_0, v_1
ADD $v_4, v_3, \#1$
STR $v_4, [v_t, v_i, LSL\#2]$

Méthode globale

```
int sum(int t[100]) {
  s = 0;
  i = 0;
  while(i < 100) {
    s = s + t[i];
    i = i +1;
  }
  return s;
}
```

Approche locale :

- chargement des registres utilisés
- sauvegarde des registres modifiés en fin de BB

Problème : certains registres pourraient être conservés entre les BB !

Allouer des registres globalement (a)

Plan :

- réserver N_{glob} registres pour une allocation globale
- appliquer la méthode locale pour le reste

Quels registres alloués globalement ?

- les plus coûteux à charger / sauvegarder !
- C_{load} – coût d'un chargement (en cycles)
- C_{store} – coût d'un stockage (en cycles)
- L_1, L_2, ..., L_{n_b} – nid de boucle contenant un BB b
- n_b – profondeur du nid de boucle contenant b
- N_1, N_2, ..., N_{n_b} – nombre d'itération de chaque boucle

Allouer des registres globalement (b)

Problème : N_i difficiles à déterminer

- pas forcément constants
- très difficiles à calculer

Approximation : $N_i \mapsto 10$ ou 16 (heuristique)

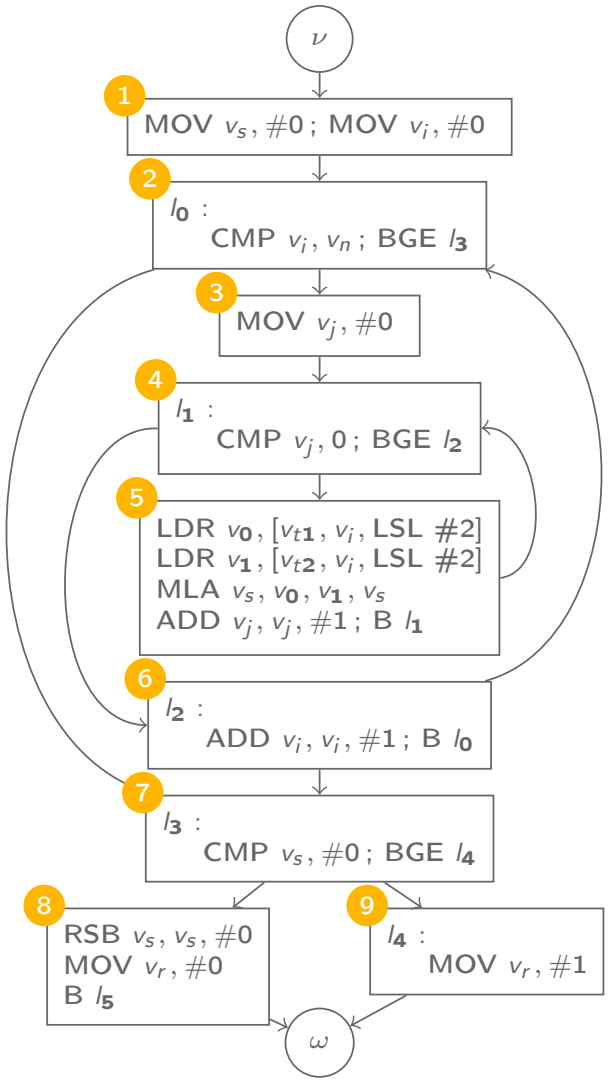
$$N_b = \Pi_{i \in [1, L_b]} N_i = 10^{L_b} \text{ ou } 16^{L_b}$$

Méthode de calcul :

- $s^b_{use} \in \{0, 1\}$ – registre utilisé dans b
- $s^b_{def} \in \{0, 1\}$ – registre défini dans b
- 1 load / 1 store max par BB (hors vidage)

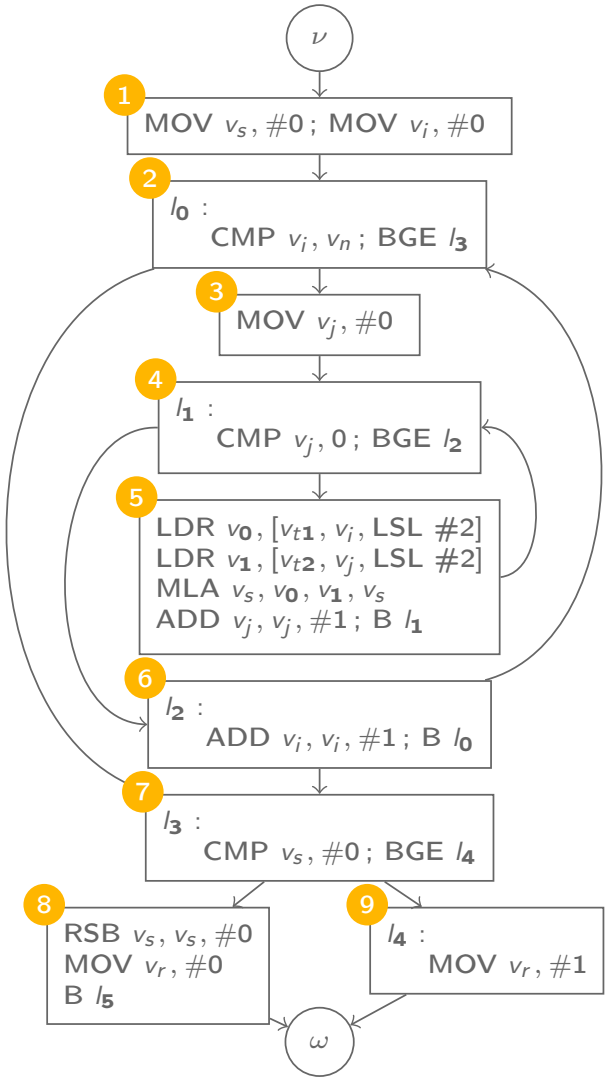
$$\forall v \in \mathcal{V}, C_v = \sum_{b \in V} s^b_{use} C_{load} N_b + s^b_{def} C_{store} N_b$$

Exemple (a)




```
s = 0;
for(i = 0; i < n; i++)
  for(j = 0; j < n; j++)
    s += t1[i] * t2[i];
if(s < 0) {
  s = -s;
  r = 0;
}
else
  r = 1;
```

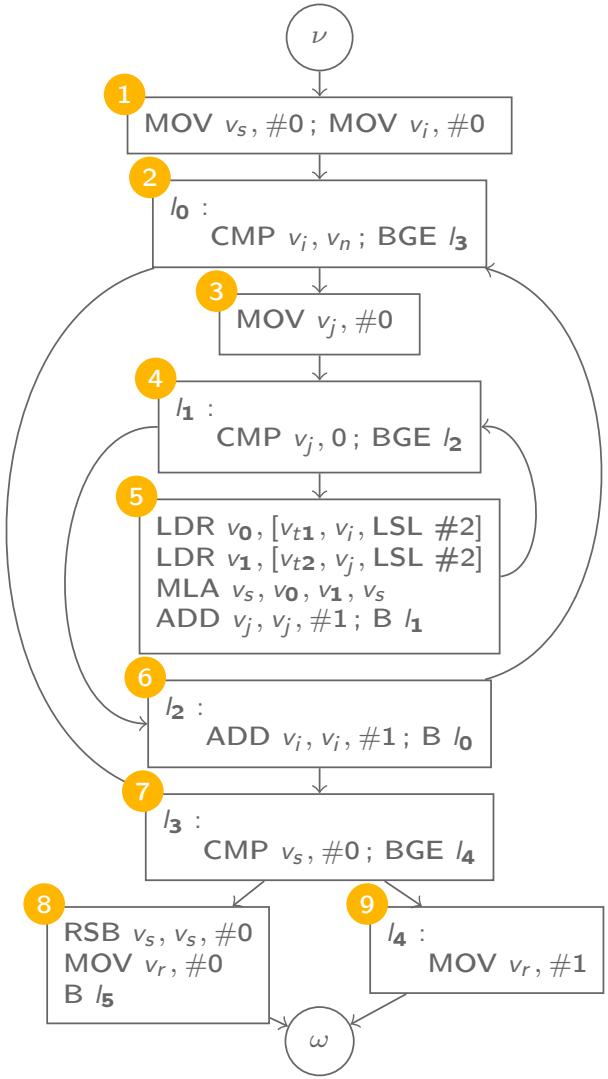
Exemple (b)



$C_{load} = 10,\ C_{store} = 5$:

$$
\begin{aligned}
v_s &= N_1 C_{store} + N_5 C_{load} + N_5 C_{store} \\
&\quad + N_7 C_{load} + N_8 C_{store} + N_8 C_{load} \\
&= 5 + 1000 + 500 + 10 + 5 + 10 \\
&= 1530 \\
v_i &= N_1 C_{store} + N_2 C_{load} + N_5 C_{load} \\
&\quad + N_6 C_{store} + N_6 C_{load} \\
&= 5 + 100 + 1000 + 50 + 100 = 1255 \\
v_j &= 50 + 1000 + 1000 + 500 \\
&= 2550 \\
v_{t1} &= 1000 \\
v_{t2} &= 1000 \\
v_r &= 10 \\
v_0 &= 1500 \\
v_1 &= 1500
\end{aligned}
$$

Exemple (c)



$C_{load} = 10,\ C_{store} = 5$:

$$
\begin{aligned}
v_s &= 1530 \\
v_i &= 1255 \\
v_j &= 2550 \\
v_{t1} &= 1000 \\
v_{t2} &= 1000 \\
v_r &= 10 \\
v_0 &= 1500 \\
v_1 &= 1500
\end{aligned}
$$

$N_{glob} = 5 \Rightarrow v_j,\ v_s,\ v_0,\ v_1,\ v_i$ alloués à $R_0..R_4$.

$N_{glob} = 3 \Rightarrow v_j,\ v_s,\ v_0$ alloués à $R_0..R_2$.

Exercice

Construire le CFG du programme ci-dessous et sélectionnez avec la méthode globale les registres à allouer à des registres matériel avec $N_{glob} = 3$.

$$
\begin{array}{ll}
 & \text{MOV } v_i, \#0 \\
l_0 : & \\
 & \text{CMP } v_i, \#128 \\
 & \text{BGE } l_1 \\
 & \text{LDR } v_x, [v_t, v_i, \text{LSL } \#2] \\
 & \text{RSB } v_0, v_i, \#256 \\
 & \text{LDR } v_1, [v_t, v_0, \text{LSL } \#2] \\
 & \text{STR } v_1, [v_t, v_i, \text{LSL } \#2] \\
 & \text{STR } v_x, [v_t, v_0, \text{LSL } \#2] \\
 & \text{ADD } v_i, v_i, \#1 \\
 & \text{B } l_0 \\
l_1 : &
\end{array}
$$

Trouver les boucles : relation de domination

Definition

Soit $G = (V, E, \nu, \omega)$, un chemin est
$[b_0 \to b_1, b_1 \to b_2, ..., b_{n-1} \to b_n]$ avec
$\forall i \in [0, n-1], b_i \to b_{i+1} \in E$.
Soit $b \to^* b'$, $b, b' \in V$, les chemins de b à b'.

Definition

$\forall b, b' \in V$, b domine b', écrit, b *dom* b' ssi $\forall p \in \nu \to^* b'$,
$\exists b \to b'' \in p$.

Définition d'une boucle

Definition

$h \in V$ est une tête de boucle, $header(h)$, ssi
$\exists b \rightarrow h \in E \ \wedge \ h \ dom \ b$.
$b \rightarrow h$ appelé *arc retour*.

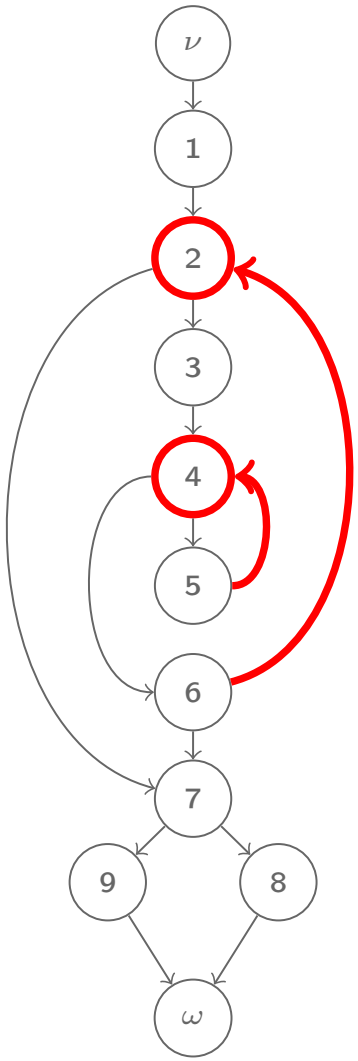
Definition

Une boucle L_h avec pour tête h est l'ensemble
$\{b \in V \mid h \ dom \ b \ \wedge \ \exists p \in h \rightarrow^* h \ \wedge \ b \in p\}$.

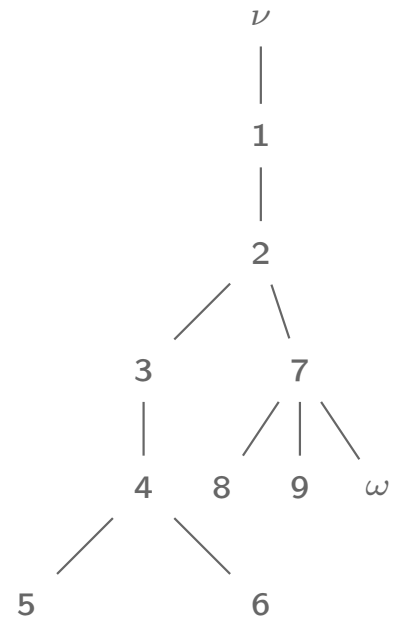
Definition

Une boucle L_h est imbriquée dans une boucle $L_{h'}$ ssi $h' \ dom \ h$.

Exemple (suite)



Arbre de domination :



Arc retour :

- $6 \rightarrow 2$ car 2 *dom* 6 donc *header*(2)
- $5 \rightarrow 4$ car 4 *dom* 5 donc *header*(4)

Coloriage de graphe

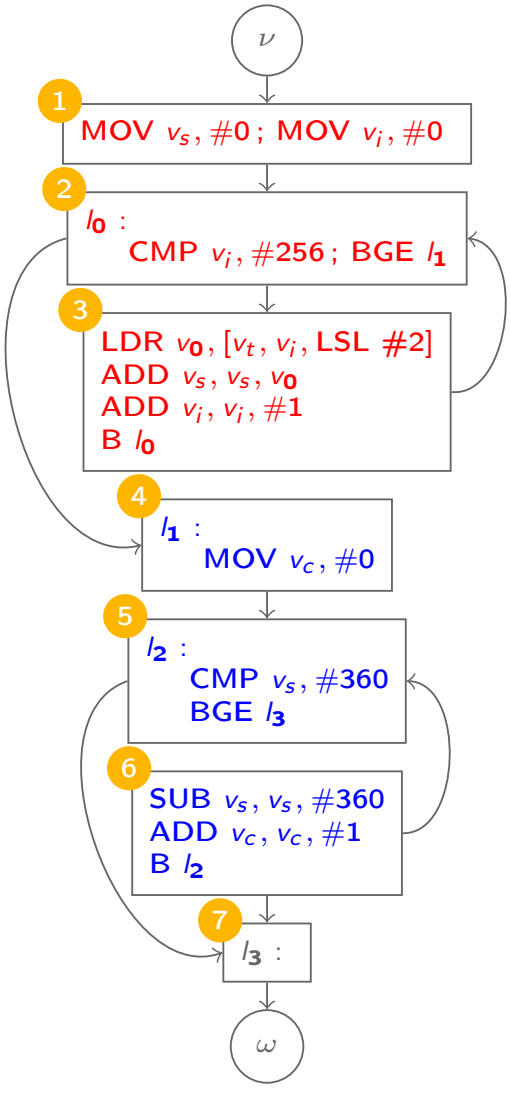
```
s = 0;
for(i = 0; i < 256; i++)
   s += t[i];
c = 0;
while(s > 260) {
   s -= 360;
   c =c + 1;
}
```

Problème :

- i alloué dans un registre matériel
- plus utilisé dans la seconde boucle !

Solution :
utiliser la propriété de vivacité !

Interférence de vivacité



- rouge – durée de vie de i
- bleu – durée de vie de c

pas d'intersection $\Rightarrow$
allocation dans le même registre matériel !

Graphe d'interférence

Definition

Un graphe d'interférence G_{int} est un graphe non-dirigé $G_{int} = \langle V_{int}, E_{int} \rangle$ avec $V_{int} \subseteq \mathcal{V}$ et $E_{int} \subseteq V_{int} \times V_{int}$ tel que : $(b, b') \in E_{int}$ ssi il existe un $b \in V$ et $i \in b$ où v et v' sont vivantes.

Allocation de registre $\simeq$ problème de coloriage de graphe (1 couleur = 1 registre) mais :

- problème compliqué
- échec de coloriage $\Rightarrow$ seulement du vidage
- quel registre vider ?

Procédure

1. construire G_{int} à partir de l'information de vivacité des variables
2. à partir G_{int}, construire une pile de registres S (*les registres les plus près du sommet seront alloués en priorité*)
3. utiliser S pour colorier G_{int} et éventuellement identifier des registres à viser

Construction de la pile

Lemme : $\forall b \in V_{int}$, si $deg(b) < |\mathcal{R}|$ alors b sera toujours coloriable.

$S \leftarrow [\,]$
while $V_{int} \neq \emptyset$ **do**
  supprimer de V_{int} et empiler les sommets b tel que $deg(b) < |\mathcal{R}|$
  sélectionner le sommet v de poids minimum, W_v, le supprimer de V_{int} et l'empiler
end while

Heuristique : $W_v = \frac{C_v}{deg(v)}$

- allouer les variables les plus coûteuses à accéder $\rightarrow C_v$
- supprimer les variables les plus liées $\rightarrow deg(v)$

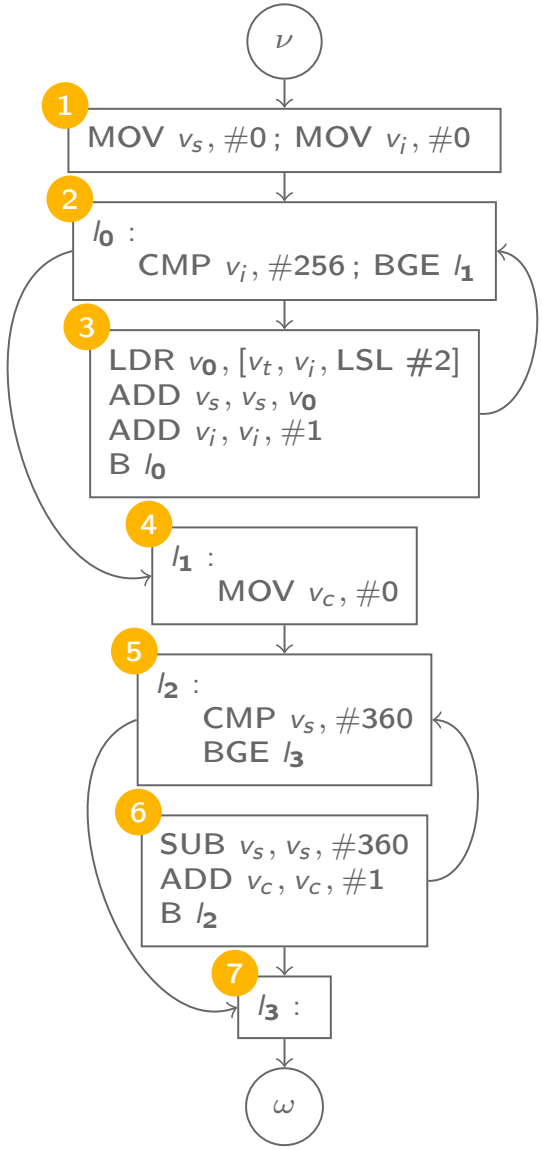
Coloriage des sommets

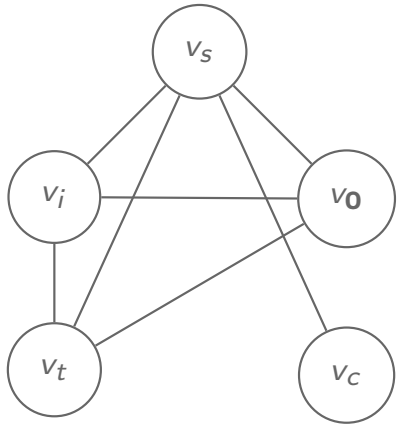
$\forall b \in V_{int}, col(b) \leftarrow \perp$ – couleur de chaque registre

for all $v \in S$ **do**
  dépiler v de S
  trouver $c \in \mathcal{R}$ tel qu'aucun sommet contiguë n'ait la même couleur
  if c ne peut être trouvé **then**
    $col(v) \leftarrow \perp$
  else
    $col(v) \leftarrow c$
  end if
end for

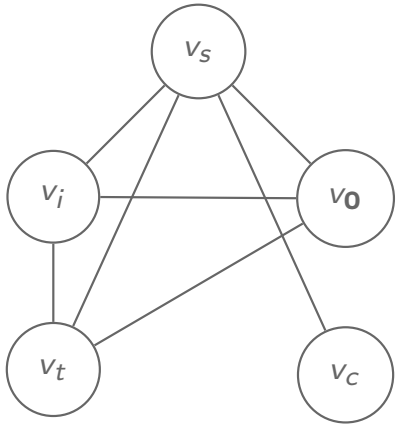
Les registres v dont $col(v) = \perp$ ne sont pas alloués et devront être chargés / stockés avec R_{temp} à chaque utilisation.

Exemple : construction de G_{int}





Construction de la pile



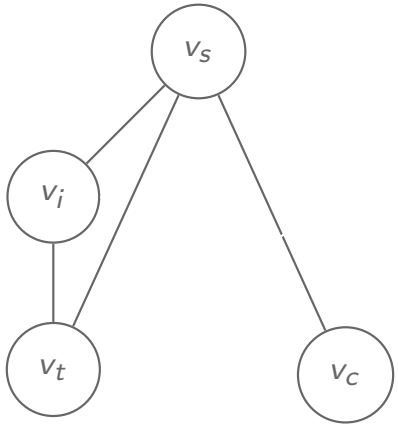
$$|\mathcal{R}| = 4$$
$$C_{load} = 10$$
$$C_{store} = 5$$

v	C_v	$deg(v)$	W_c
v_s	405	4	101
v_i	355	3	118
v_0	150	3	50
v_t	100	3	33
v_c	155	1	155

$$S = [v_c, v_i, v_s, v_t, v_0]$$

Convention : en cas de choix, empiler le v_i, $i \in \mathbb{N}$, de plus grand i, ou alors le v_a, $a \in \{a, b, ..., z\}$, de plus grand a.

Construction de la pile



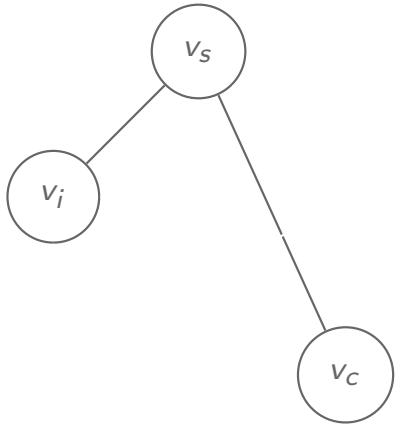
$$|\mathcal{R}| = 4$$
$$C_{load} = 10$$
$$C_{store} = 5$$

v	C_v	$deg(v)$	W_c
v_s	405	4	101
v_i	355	3	118
v_0	150	3	50
v_t	100	3	33
v_c	155	1	155

$$S = [v_c, v_i, v_s, v_t, v_0]$$

Convention : en cas de choix, empiler le v_i, $i \in \mathbb{N}$, de plus grand i, ou alors le v_a, $a \in \{a, b, ..., z\}$, de plus grand a.

Construction de la pile



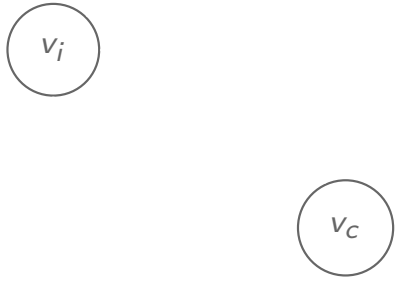
v	C_v	$deg(v)$	W_c
v_s	405	4	101
v_i	355	3	118
v_0	150	3	50
v_t	100	3	33
v_c	155	1	155

$$|\mathcal{R}| = 4$$
$$C_{load} = 10$$
$$C_{store} = 5$$

$$S = [v_c, v_i, v_s, v_t, v_0]$$

Convention : en cas de choix, empiler le v_i, $i \in \mathbb{N}$, de plus grand i, ou alors le v_a, $a \in \{a, b, ..., z\}$, de plus grand a.

Construction de la pile



$$|\mathcal{R}| = 4$$
$$C_{load} = 10$$
$$C_{store} = 5$$

v	C_v	$deg(v)$	W_c
v_s	405	4	101
v_i	355	3	118
v_0	150	3	50
v_t	100	3	33
v_c	155	1	155

$$S = [v_c, v_i, v_s, v_t, v_0]$$

Convention : en cas de choix, empiler le v_i, $i \in \mathbb{N}$, de plus grand i, ou alors le v_a, $a \in \{a, b, ..., z\}$, de plus grand a.

Construction de la pile

v	C_v	$deg(v)$	W_c
v_s	405	4	101
v_i	355	3	118
v_0	150	3	50
v_t	100	3	33
v_c	155	1	155

$$S = [v_c, v_i, v_s, v_t, v_0]$$

$$|\mathcal{R}| = 4$$
$$C_{load} = 10$$
$$C_{store} = 5$$

Convention : en cas de choix, empiler le v_i, $i \in \mathbb{N}$, de plus grand i, ou alors le v_a, $a \in \{a, b, ..., z\}$, de plus grand a.

Construction de la pile

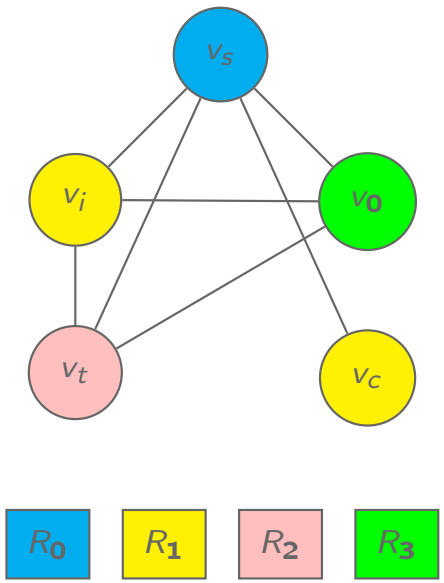
v	C_v	$deg(v)$	W_c
v_s	405	4	101
v_i	355	3	118
v_0	150	3	50
v_t	100	3	33
v_c	155	1	155

$$S = [v_c, v_i, v_s, v_t, v_0]$$

$$|\mathcal{R}| = 4$$
$$C_{load} = 10$$
$$C_{store} = 5$$

Convention : en cas de choix, empiler le v_i, $i \in \mathbb{N}$, de plus grand i, ou alors le v_a, $a \in \{a, b, ..., z\}$, de plus grand a.

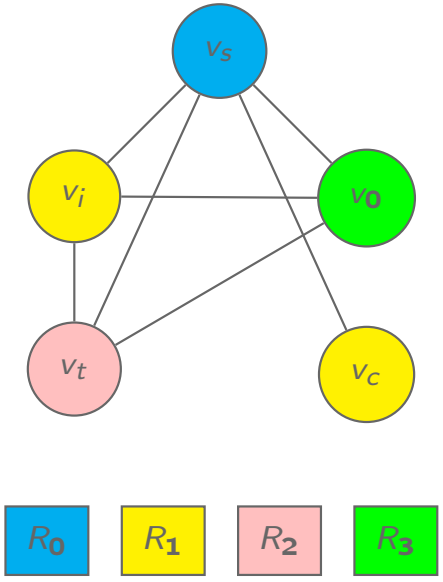
Coloriage

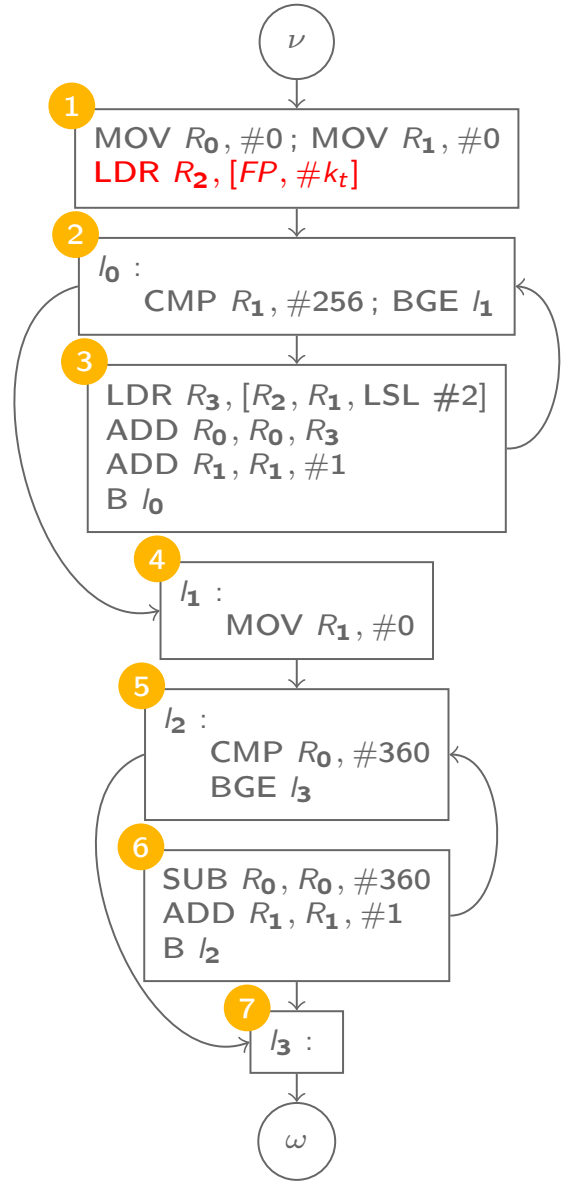


$S = [v_s, v_c, v_i, v_t, v_0]$

1. dépiler $v_s \Rightarrow col(v_s) = 0$
2. dépiler $v_c \Rightarrow col(v_c) = 1$
3. dépiler $v_i \Rightarrow col(v_i) = 1$
4. dépiler $v_t \Rightarrow col(v_t) = 2$
5. dépiler $v_0 \Rightarrow col(v_0) = 3$

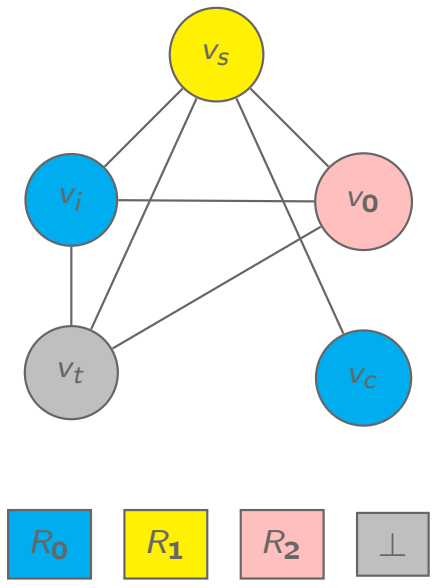
Renommage des registres





Note : t paramètre = lecture
$\Rightarrow$ il faut le charger dans R_2 !

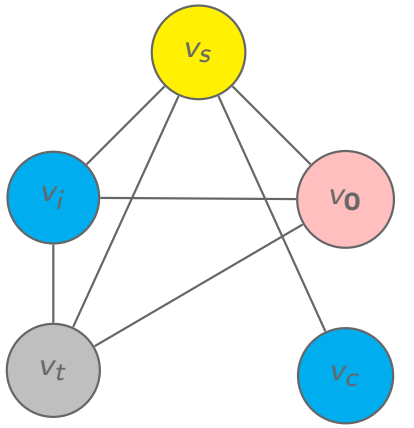
Exemple : avec $|\mathcal{R}| = 3$

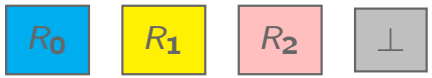


$S = [v_i, v_s, v_0, v_t, v_c]$

1. dépiler $v_i \Rightarrow col(v_i) = R_0$
2. dépiler v_s
   $\Rightarrow col(v_s) = R_1$
3. dépiler v_0
   $\Rightarrow col(v_0) = R_2$
4. dépiler $v_t \Rightarrow col(v_t) = \perp$
5. dépiler v_c
   $\Rightarrow col(v_c) = R_0$

Renommage des registres avec $|\mathcal{R}| = 3$



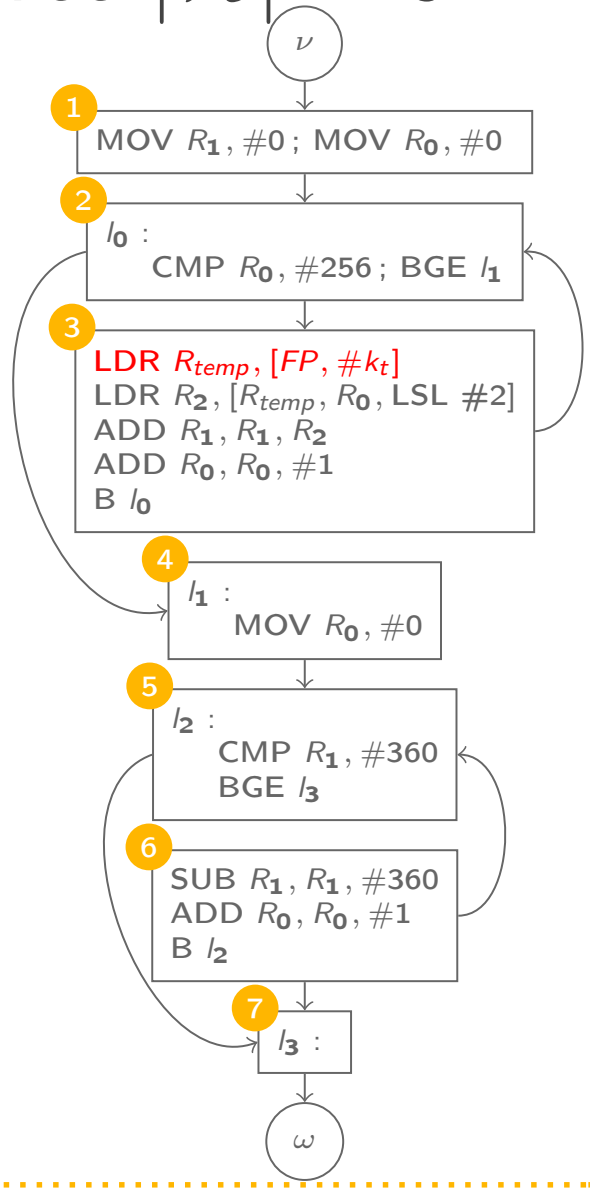


allocation locale :

$F = \{R_{temp}, R_{return}\}$,

$\sigma = \{v_s \mapsto R_0, v_c \mapsto R_1,$

$v_i \mapsto R_1, v_t \mapsto R_2\}$



Coloriage et appel de sous-programme

- possibilité d'échanger toute couleur
- correspondance avec les paramètres / résultat de l'appelant
- correspondance avec les paramètres / résultat
- sinon, stockage dans la pile puis rechargement au besoin

Exercice

Réalisez l'allocation des registres par coloriage pour le code et les configurations ci-dessous :

1. $|\mathcal{R}| = 4$
2. $|\mathcal{R}| = 3$

```
AND v0, vx, #0xff
MOV v1, v0, LSL#24
AND v2, vx, #0xff00
MOV v3, v2, LSL#8
ORR v4, v1, v3
AND v5, vx, #0xff0000
MOV v6, v5, LSR#8
ORR v7, v4, v6
AND v8, vx, #0xff000000
MOV v9, v8, LSR#24
ORR vy, v7, v9
    MOV vn, #0
    MOV vs, #1
l0 :
    AND v10, vy, #0x80000000
    CMP v10, #0
    BNE l1
    MOV vy, vy, LSL#1
    ADD vn, vn, #1
    MOV vs, vs, LSL#1
    B l0
l1 :
```