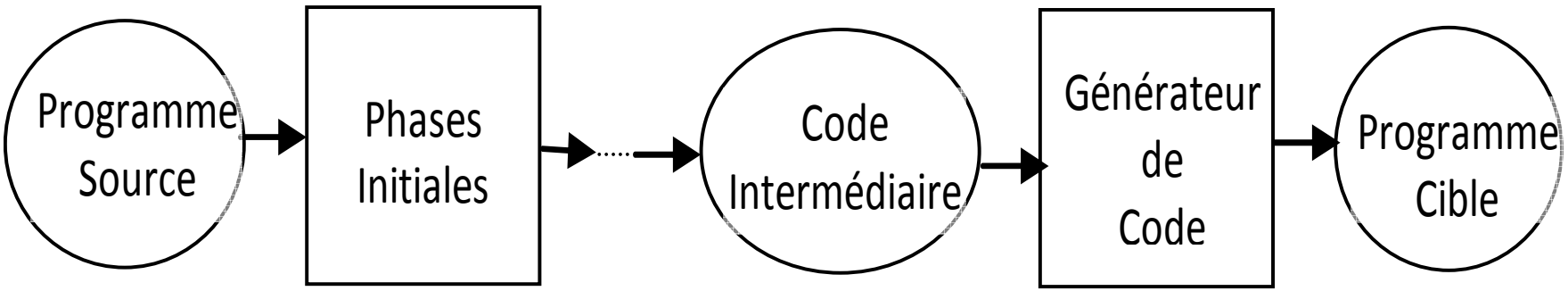
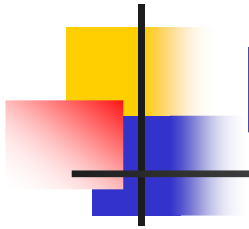
Production de Code

ESI – École nationale Supérieure en Informatique

Production de code

- Le code généré est machine dépendant i.e. le code généré est spécifique à un type de machine (type de processeur et jeu d'instruction associé).

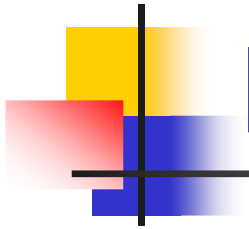




Production de code

- Le code machine compréhensible par la machine dit également code natif aura une forme ressemblant au code suivant exprimé en hexadécimal :

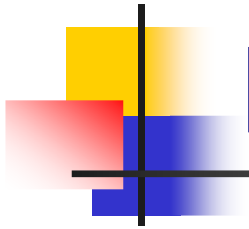
adresse	code
0046	8B45FC
0049	4863F0
004c	8B45FC
004f	4863D0
0052	8B45FC
0055	4898
0057	8B0485



Production de code

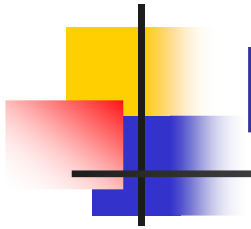
- Pour avoir une forme intelligible, nous exprimerons le code machine par des mnémoniques tel que fait dans le programme suivant :

```
MOV CX,34H
MOV [201H],CX
MOV DX,808H
MOV [202H],DX
MOV AX,[201H]
ADD AX,14H
MOV [200H],AX
```



Machine cible

- Connaître toutes les spécificités de la machine cible sur laquelle s'exécutera le code produit par le compilateur.
- En particulier, il faudra connaître le jeu d'instructions de la machine cible, le nombre de registres, la taille des mots mémoire et les modes d'adressage.



Machine cible

Nous nous intéresserons à une machine fictive mais qui s'inspire largement des processeurs Intel.

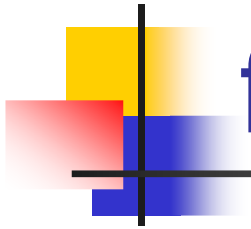
- Elle dispose de N registres numérotés : R_0, R_1, … , R_{n-1}
- Les mots ont une taille de 4 octets
- La machine est adressable par octet
- Les instructions ont le format suivant :

op source destination

dont la signification est :

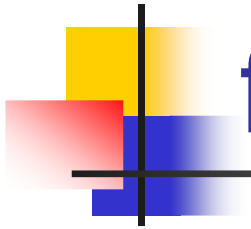
destination ←destination **op** source

Blocs de base et graphes de flot de contrôle

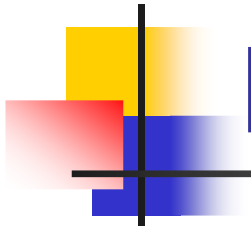


- Avant de produire le code machine proprement dit, le code intermédiaire est parcouru plusieurs fois.
- Ce code intermédiaire est tout d'abord organisé en un graphe dit "graphe de flot de contrôle". Les nœuds de ce graphe représentent les traitements et les arcs le flot de contrôle.

Blocs de base et graphes de flot de contrôle



- Le but de cette construction est d'optimiser le code produit i.e. produire un code machine de qualité utilisant au mieux les ressources de la machine (utiliser le plus possible des opérandes registres, éviter les modes d'adressage complexes, …).



Bloc de base

- Chaque nœud du graphe de flot de contrôle est appelé "bloc de base".
- Un bloc de base est en fait une séquence d'instructions consécutives dans laquelle le flot de contrôle est activé au début (première instruction) et se termine à la fin (dernière instruction) sans branchement ni arrêt.

Algorithme de partitionnement

1. Déterminer les instructions de tête de chaque bloc :

   i) La première instruction du programme est une instruction de tête ;

   ii) Toute instruction atteinte par branchement est une instruction de tête ;

   iii) Toute instruction qui suit un branchement est une instruction de tête.

2. Pour chaque instruction de tête :

   - Le bloc de base correspondant débute par cette instruction et est augmenté des instructions qui suivent dans le programme une à une jusqu'à atteindre une autre instruction de tête ou la fin du programme. Il est à noter que l'instruction de tête atteinte ne fait pas partie du bloc de base.

Algorithme de partitionnement Exemple

- Code source :

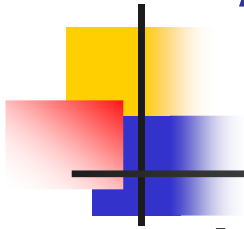
```
{
prod =0;
i:=1;
do
        {
        prod += a[i]*b[i];
        i:= i++;
        }
while (i<= 20)
}
```

Algorithme de partitionnement Exemple

- Code intermédiaire :

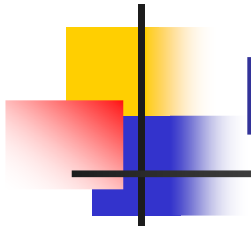
```
1.    prod := 0
2.    i := 1
3.    t1 := 4 * i
4.    t2 := a [t1]
5.    t3 := 4 * i
6.    t4 := b [t3]
7.    t5 := t2 * t4
8.    t6 := prod + t5
9.    prod := t6
10.   t7 := i + 1
11.   i := t7
12.   CMP i  20
13.   JLE (3)
```

Algorithme de partitionnement Exemple

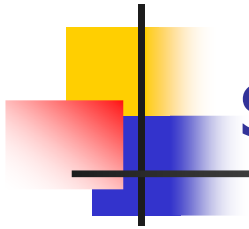


- Après application de l'algorithme de partition du programme précédent en bloc de base, on obtient :
  - Bloc de base #1 constitué des instructions (1) et (2) ;
  - Bloc de base #2 constitué du reste des instructions.

Transformations sur les blocs de Base

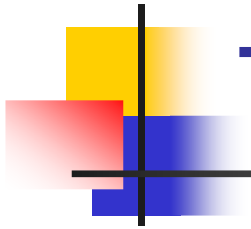


- Les transformations préservant la structure des programmes ;
- Les transformations algébriques.



Transformations préservant la structure des programmes

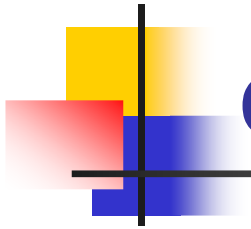
- Elimination des sous-expressions communes
- Elimination du code inutile
- Renommer des variables temporaires
- Echange d'instructions adjacentes



Transformations algébriques

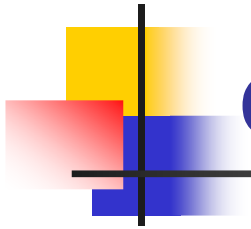
- Ces transformations modifient la structure des programmes sans modifier les résultats calculés.
- On peut citer comme types de transformations algébriques:
  - La simplification des expressions
  - L'utilisation d'opérateurs moins coûteux

Construction du graphe de flot de contrôle



- Le graphe de flot de contrôle est un graphe orienté dont les sommets ou nœuds sont les blocs de base construit précédemment.

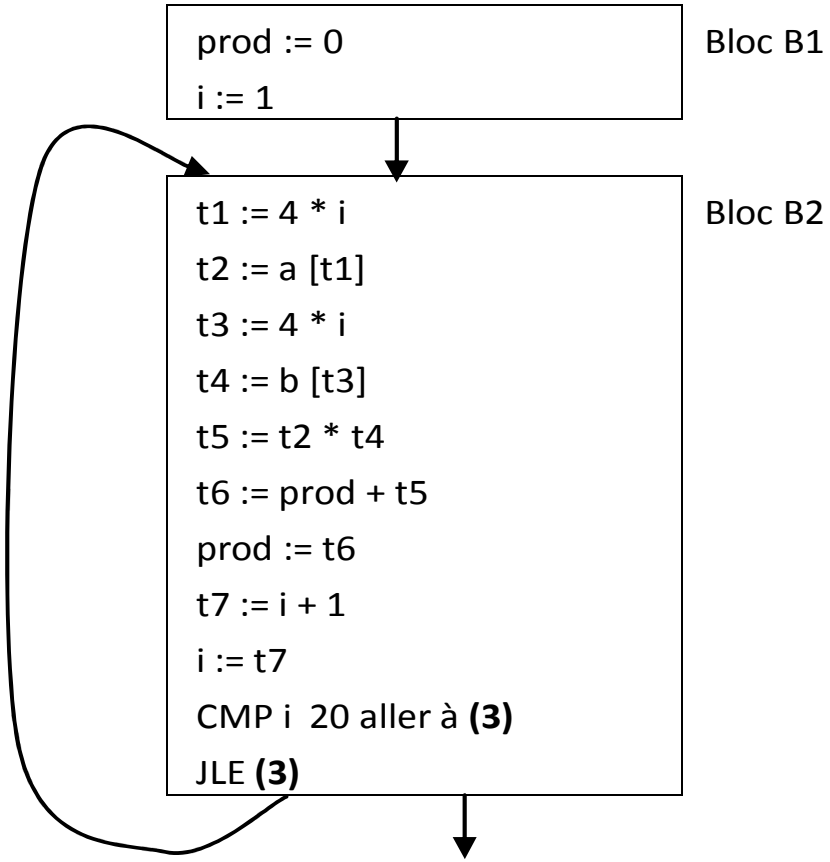
Construction du graphe de flot de contrôle



- Le nœud initial (ou d'entrée du programme) contient la première instruction ;
- Un arc partant d'un bloc B1 vers un bloc B2 si B2 peut suivre immédiatement B1 dans une exécution, i.e. :
  - La dernière instruction de B1 est une instruction de branchement vers la première instruction de B2 ou,
  - Le bloc B2 suit immédiatement le bloc B1 dans l'ordre du programme et B1 ne se termine pas par un branchement inconditionnel.

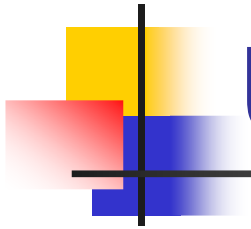
Construction du graphe de flot de contrôle

- Graphe de l'exemple précédent :



Construction du graphe de flot de contrôle

- **Définition des boucles :**
- Une boucle est un ensemble de nœuds d'un graphe de flot de contrôle tel que :
  - Tous les nœuds sont fortement connectés;
  - La collection de nœuds a une entrée unique



Un générateur de code simple

- produire du code machine pour un bloc de base du graphe de flot de contrôle à partir d'une représentation (code) intermédiaire sous forme de quadruplets.

Informations d'utilisation ultérieure

i) Analyser un bloc de base de la fin vers le début ;

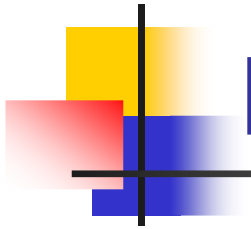
ii) Lorsqu'on atteint un quadruplet i : (A := B op C)

Faire

a) Attacher au quadruplet i les informations "actuelles" (courantes) de la table es symboles concernant les prochaines utilisations et l'activité des variables A, B et C ;

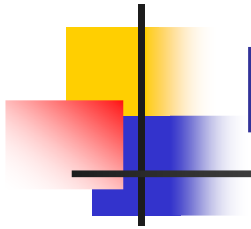
b) Dans la table des symboles, attacher à A l'information "pas actif" et "pas de prochaine utilisation" ;

c) Attacher à B et C l'information "actif" et prochaine utilisation de B et C sera faite au quadruplet N° i.



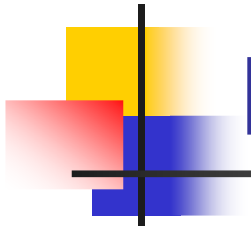
Descripteurs de registres

- Les descripteurs de registres gardent trace du contenu courant des registres durant la phase de génération de code.
- Les descripteurs seront consultés à chaque nouvelle allocation de registre pour produire une instruction machine.
- Initialement, le descripteur de registre indique que tous les registres de la machine sont vides.
- Au cours de la génération de code, chaque registre peut contenir zéro, un ou plusieurs noms à un instant donné.



Descripteurs d’adresses

- Pour chaque nom dans un bloc de base, le descripteur d'adresses garde trace de l'emplacement où on peut trouver la valeur courante d'un nom à l'exécution.
- Cet emplacement peut être un registre ou une adresse mémoire.
- Les informations des descripteurs d'adresses ou de registres peuvent être stockées dans la table des symboles.



Descripteurs - Exemple

- Instruction d'un programme cible :

```
d := (a – b) + (a -  c) + (a – c);
```

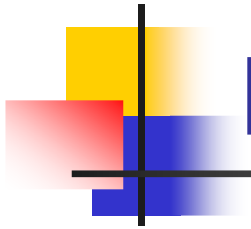
- Quadruplets correspondants :

```
t := a – b

u := a – c

v := t + u

d := v + u
```



Descripteurs - Exemple

Instruction	Code Produit	Descripteur de registres	Descripteur d'adresses
		Les registres sont vides	
t := a - b	Mov a, R0 Sub b, R0	R0 contient t	t est dans R0
u := a – c	Mov a, R1 Sub c, R1	R0 contient t R1 contient u	t est dans R0 u est dans R1
v := t + u	Add R1, R0	R0 contient v R1 contient u	v est dans R0 u est dans R1
d := v + u	Add R1, R0 Mov R0, d	R0 contient d	d est dans R0 d est dans R0 et dans la mémoire

Algorithme de production de code

Pour chaque quadruplet A:= B op C
Faire

a) Invoquer la fonction GETREG( ) pour déterminer l'emplacement L où le résultat B op C sera rangé (on tentera le plus possible de trouver un emplacement registre pour L mais L peut avoir un emplacement mémoire) ;

b) Consulter le descripteur d'adresse pour la variable B pour sélectionner B' l'emplacement courant (si plusieurs emplacements, choisir un emplacement registre) ;

Si B n'est pas dans L
Alors Générer Mov B', L

c) Générer l'instruction Op C', L où C' est un des emplacements courants de C
Mise à jour du descripteur d'adresse pour A pour indiquer que A est dans L ;
si L est un registre, M.A.J de son descripteur pour indiquer que L contient A;

d) Si les valeurs courantes de B et/ou C n'ont pas d'utilisation ultérieure, ne sont pas actives à la sortie du bloc et sont dans des registres alors mise à jour des descripteurs de registres pour indiquer qu'après exécution de A:= B op C, ces registrent ne contiennent plus B et/ou C.

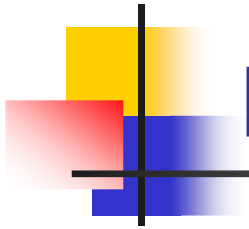
Algorithme de production de code Fonction GetReg

i) **Si** le nom B est dans un registre qui ne contient pas la valeur d'autres noms (exemple : pour X:=Y on aura deux noms dans un même registre) et B n'est pas active et n'a pas d'utilisation ultérieure après A := B op C
Alors
Retourner le Registre de B pour L ;
Mise à jour du descripteur d'adresse de B: "B n'est plus dans L" ;

ii) **Si i)** échoue
Alors Retourner un registre vide pour L s'il y a un registre vide disponible;

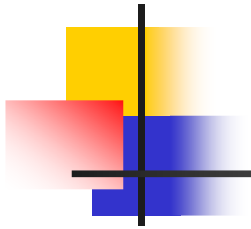
iii) **Si iii)** échoue
Alors
Si A a une utilisation future dans le bloc
ou Op est un opérateur qui requiert un registre
Alors
Sélectionner un registre déjà occupé R ;
Sauvegarder sa valeur en mémoire par l'instruction Mov R, M ;
Mise à jour du descripteur d'adresse pour M et retourner R ;

iv) **Si** A n'est pas utilisé plus loin dans le bloc
ou aucun registre occupé ne convient
Alors choisir pour L l'emplacement mémoire de A.



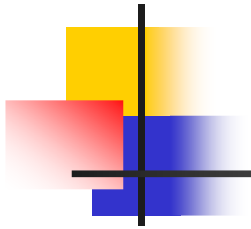
Production de code à partir de DAG

- DAG est l'acronyme de "Directed Acyclic Graph" (ou graphe orienté sans cycles).
- Dans le contexte de la production de code, le DAG sera utilisé pour essayer d'améliorer la qualité du code produit.



Construction du DAG

- les feuilles sont étiquetées avec des identificateurs uniques qui sont soit des noms de variables soit des constantes.
- Les nœuds internes sont les symboles d'opérateurs. Les nœuds internes portent également des étiquettes qui représentent les calculs intermédiaires effectués dans un bloc de base.



Construction du DAG

- Parcours des quadruplets du bloc de base.
- A chaque quadruplet créer un nœud "opération" avec deux fils dans le cas d'opérations binaires ou un nœud "opération" avec un fils dans le cas d'opérations unaires.
- Avant de créer les nœuds fils , on vérifie d'abord si un ou les deux opérandes de l'opération considérée ont été déjà calculés (nœuds existant déjà dans le DAG). Si c'est le cas, on ne créera pas de nœud pour ce type d'opérande, mais on dirigera l'arc du nœud opérateur vers le nœud déjà crée.

Exemple 1 de DAG

- Quadruplets d'un bloc de base :

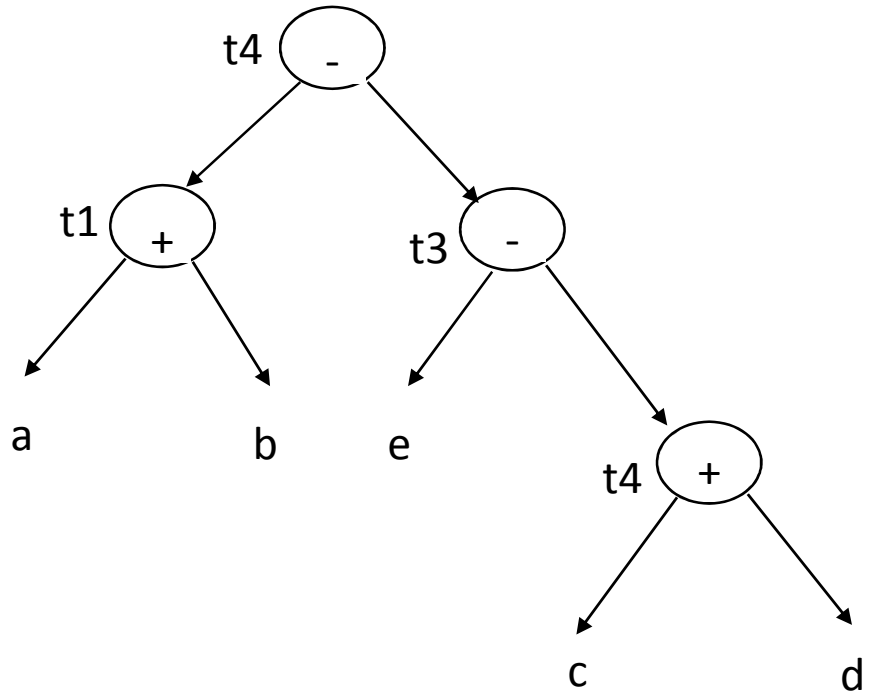
  t1 := a + b

  t2 := c + d

  t3 := e – t2

  t4 := t1 – t3

- DAG correspondant :

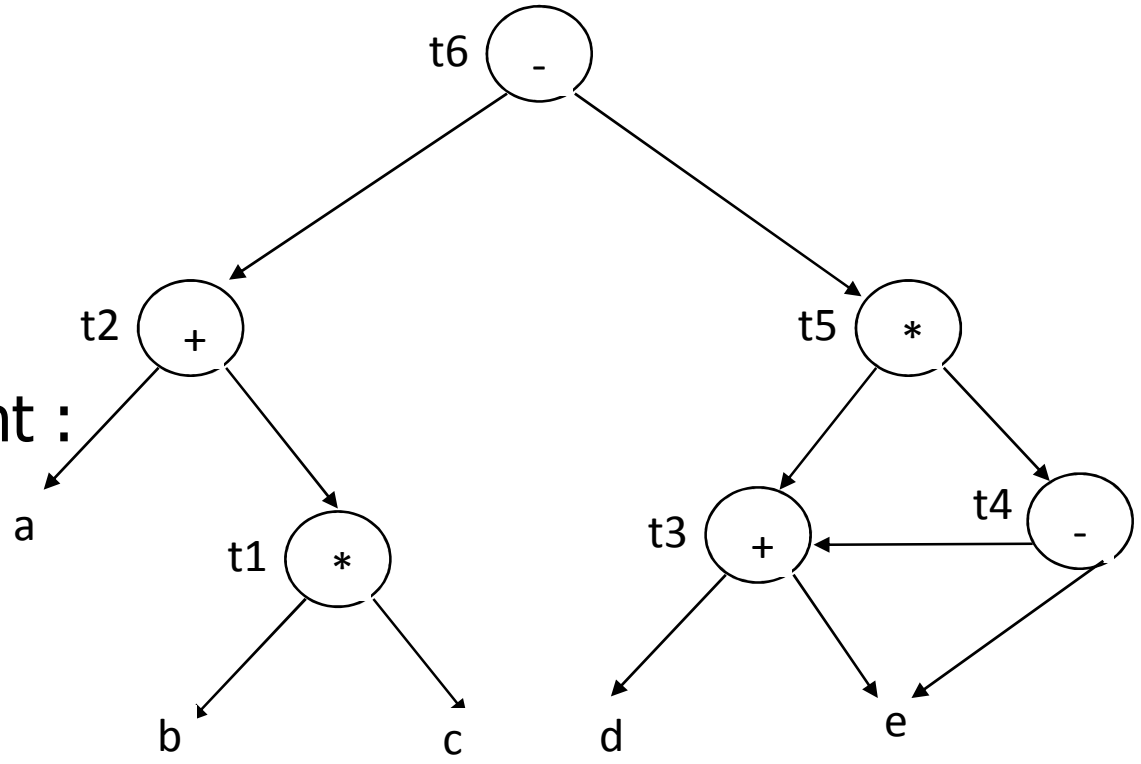


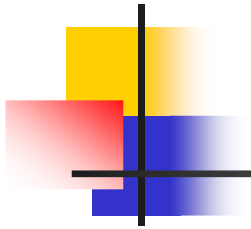
Exemple 2 de DAG

- Quadruplets d'un bloc de base :

  t1 = b * c
  t2 = a + t1
  t3 = d + e
  t4 = t3 – e
  t5 = t3 * t4
  t6 = t2 – t5

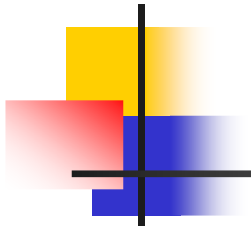
- DAG correspondant :





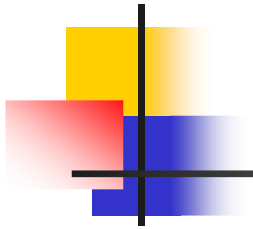
Heuristique d’ordonnancement

- Pour produire pour un bloc de base un meilleur code que celui produit par l'algorithme vu dans les sections précédentes
- Il est parfois judicieux de réordonner les quadruplets (sans modifier les résultats obtenus) de ce bloc en utilisant "les informations" qu'apporte un DAG.
- Une fois un meilleur ordre trouvé, l'algorithme de la section devra être appliqué.



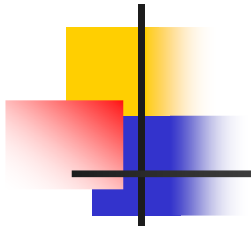
Heuristique d’ordonnancement

- une heuristique proposée par Aho, Sethi, Ullman
- L'idée de base de cette heuristique est que l'évaluation d'un nœud suit (autant que possible) l'évaluation de son fils gauche.
- D’après les auteurs, en général, avec ce ré-ordre un meilleur code que celui obtenu sans réordonner les quadruplets mais ce n'est pas toujours le cas.



Heuristique d'ordonnancement

Le DAG est parcouru "en profondeur".
Initialement tous les nœuds sont non marqués ;
Tant que des nœuds internes n'ont pas été marqués
Faire
Choisir un nœud N non marqué dont tous les parents ont été marqués ;
Marquer N ;
    Tant que le fils le plus à gauche de M e N n'est pas une feuille
        et a tous ses parents marqués
    Faire
        Marquer M;
        N:= M;
    Fait
Fait.



Heuristique d'ordonnancement

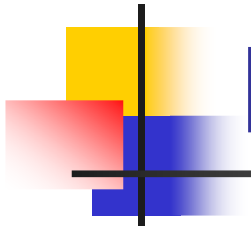
- Considérer les quadruplets suivants et supposons que la machine ne dispose que deux registres R0 et R1.

  t1 := a + b

  t2 := c + d

  t3 := e – t2

  t4 := t1 – t3



Heuristique d'ordonnancement

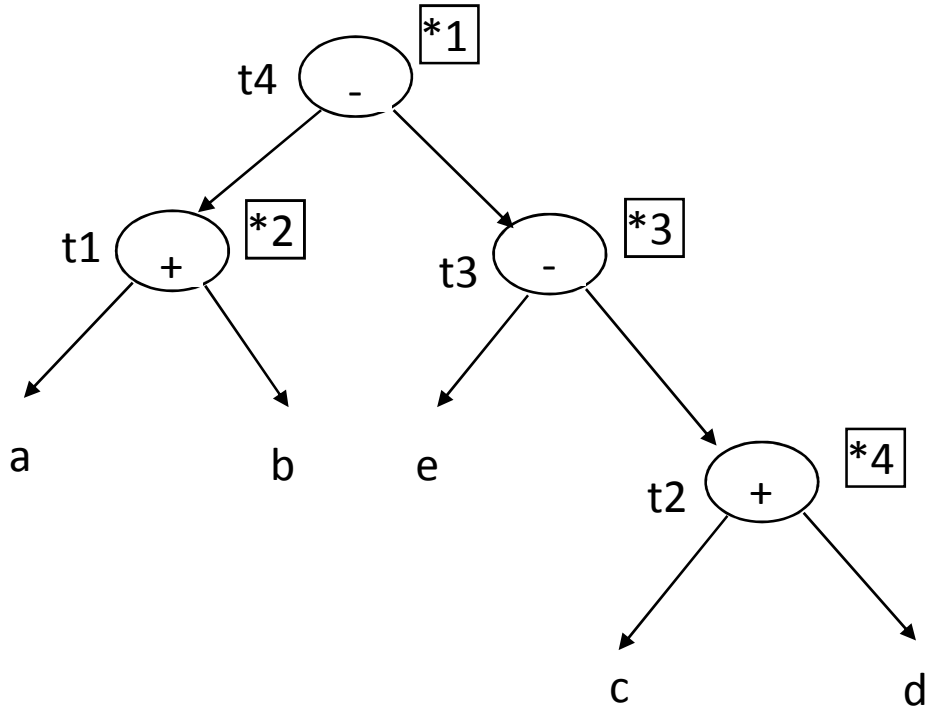
- L'application de l'algorithme simple de production de code donne le code suivant :

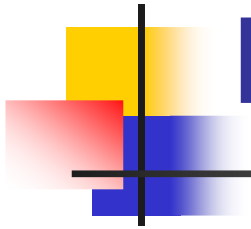
```
Mov a, R0
Add b, R0
Mov c, R1
Add d, R1
Mov R0, t1
Mov e, R0
Sub R1, R0
Mov t1, R1
Sub R0, R1
Mov R1, t4
```

Heuristique d'ordonnancement

- Application de l'heuristique sur le DAG ; le numéro de l'étoile donne l'ordre de marquage :
- Ordre d'évaluation est l'inverse de l'ordre de marquage :

Evaluer dans cet ordre : t2 t3 t1 t4

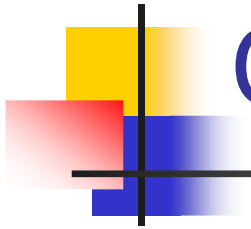




Heuristique d'ordonnancement

- Code produit après le ré-ordre :

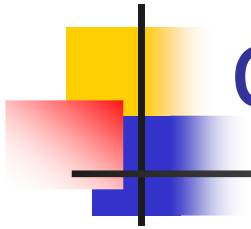
```
Mov c, R0
Add d, R0
Mov e, R1
Sub R0, R1
Mov a, R0
Add b, R0
Sub R1, R0
Mov R0, t4
```



Optimisation

- Assignation globale de registres. Le compilateur est responsable de l'allocation des registres.
- Déplacement du code invariant des boucles.
- Elimination du code inutile ou code mort : code qui n'est jamais exécuté.
- Simplification des expressions. Exemple : évaluer les quadruplets dont les arguments sont constants.
- Utilisation d'opérateurs moins coûteux.

Déplacement du code invariant des boucles



- Déplacer dans un nouveau bloc H, avant d'entrer dans la boucle L, toutes les instructions qui sont invariantes dans la boucle L.
- Algorithme à deux étapes :
  - Etape 1 : identifier les instructions invariantes.
  - Etape 2 : parmi ces instructions déplacer uniquement les instructions qui ne faussent pas l'exécution.

Déplacement du code invariant des boucles. Etape 1. Algorithme.

- Marquer comme "invariant" toute instruction dont les opérandes vérifient une des conditions suivantes :
  - Les opérandes sont constants
  - Toute définition des opérandes se fait en dehors de la boucle.
  - Les opérandes ont une seule définition à l'intérieur de la boucle et cette définition appartient aux instructions invariantes.

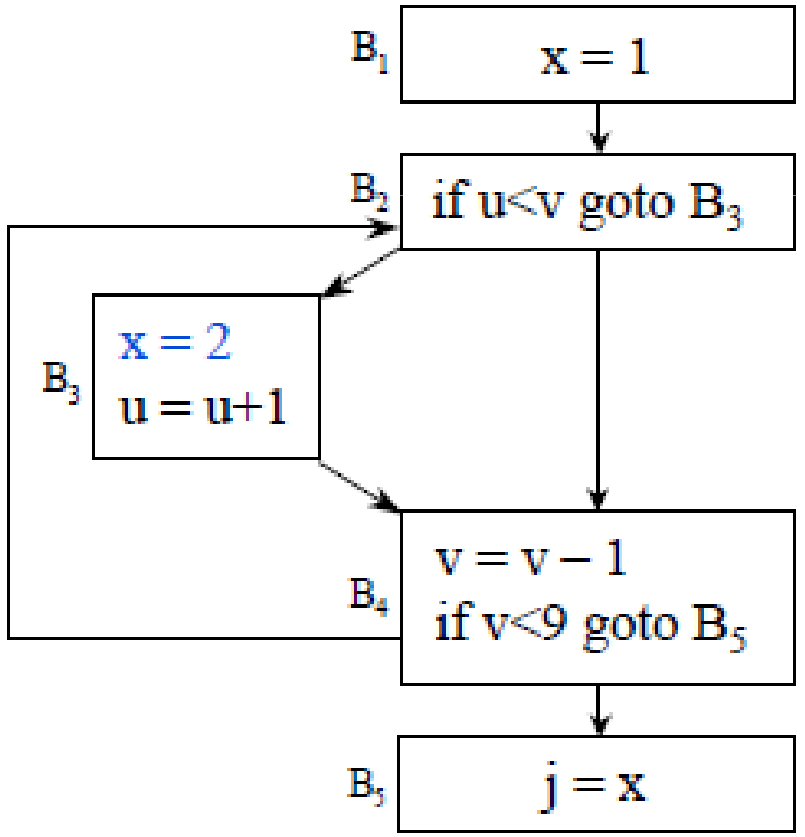
Déplacement du code invariant des boucles. Etape 2. Algorithme.

- Déplacer une instruction **I** telle que **x = y op z** marqué "invariant" si elle vérifie les trois conditions suivantes:
  1. Le bloc contenant cette instruction domine toutes les sorties de la boucle i.e. exécution systématique de cette instruction
  2. Aucune instruction de la boucle ne redéfinit x.
  3. Le bloc contenant cette instruction domine tous les usages de x i.e. tous les usages de x dans la boucle dépendent de **I**.

Déplacement du code invariant des boucles. Etape 1. Condition 1.

- Peut-on sortir l'instruction x=2 de la boucle ?

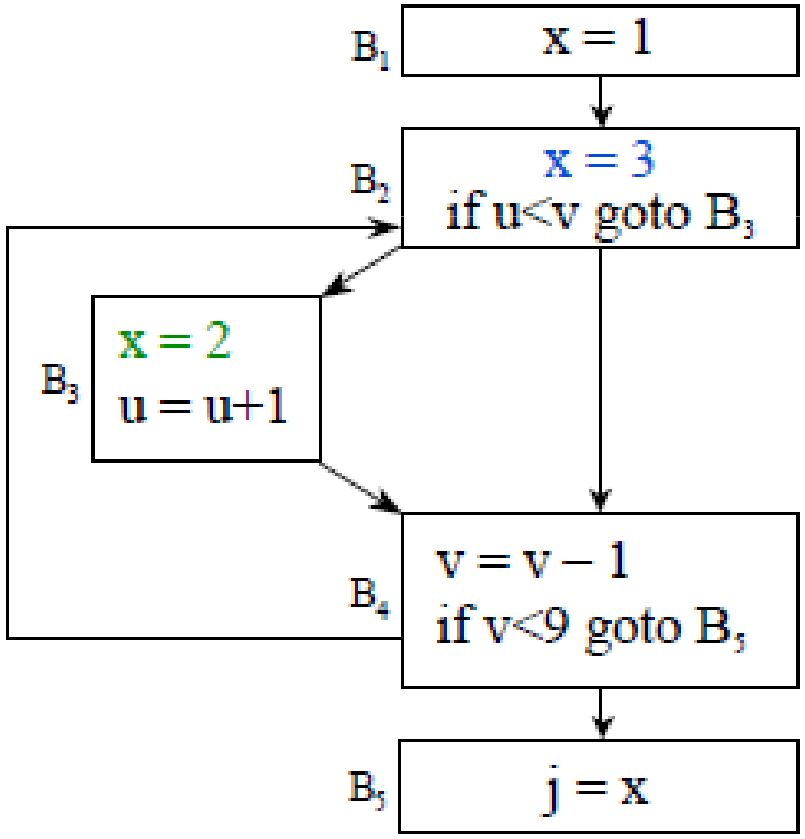
- Non car B2 ne domine pas B4.



Déplacement du code invariant des boucles. Etape 1. Condition 2.

- Peut-on sortir l'instruction x=3 de la boucle ?

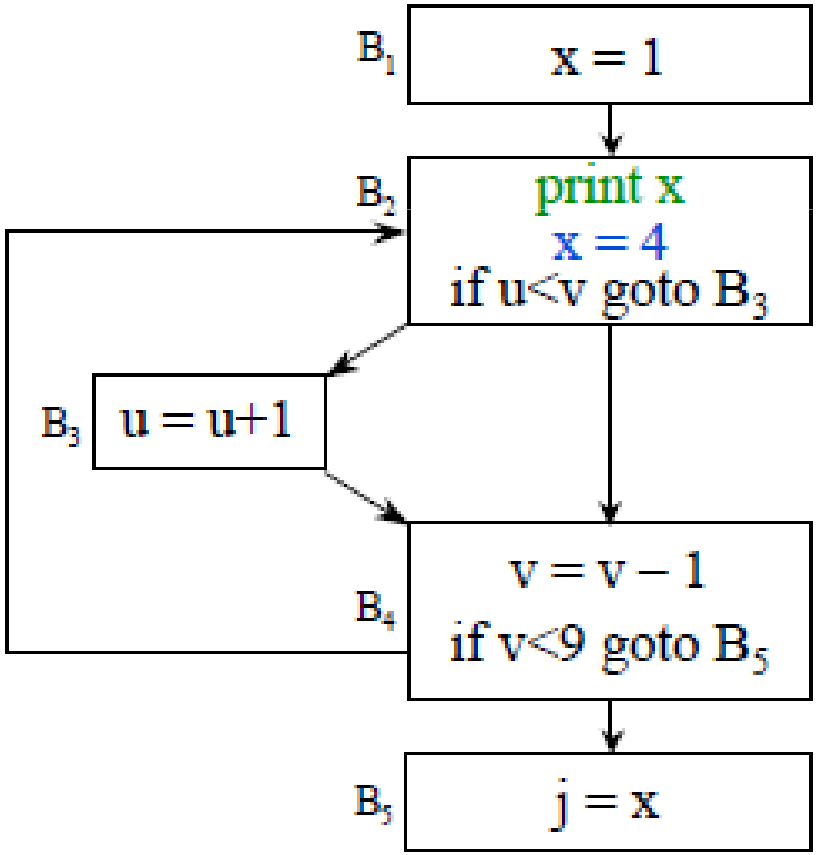
- Non.
- La condition 1 est vérifiée mais si on sort x=3, la valeur de x dans la boucle sera toujours 2 si x=2 est exécutée.

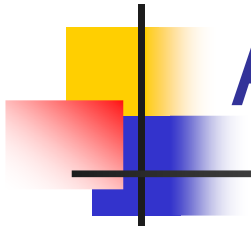


Déplacement du code invariant des boucles. Etape 1. Condition 3.

- Peut-on sortir l'instruction x=4 de la boucle ?

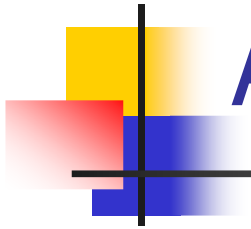
- Non.
- Les condition 1 et 2 sont vérifiées mais l'usage de x ne dépend pas uniquement de x=4.





Assignation globale de registres

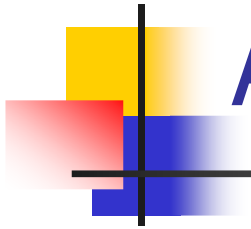
- Considérer le graphe du flot de contrôle
- Les variables "très utilisées" dans les boucles seront privilégiées lors de l'affectation des registres dans le but d'avoir un exécutable le plus rapide possible.



Assignation globale de registres

- Vivacité des variables :

  Pour un nom A en un point p (quadruplet), si la valeur de A en p peut être utilisé le long d'un chemin du graphe de flot de contrôle en partant de p alors la variable **A est vivante au point p** ; autrement la variable **A est dite non vivante**.



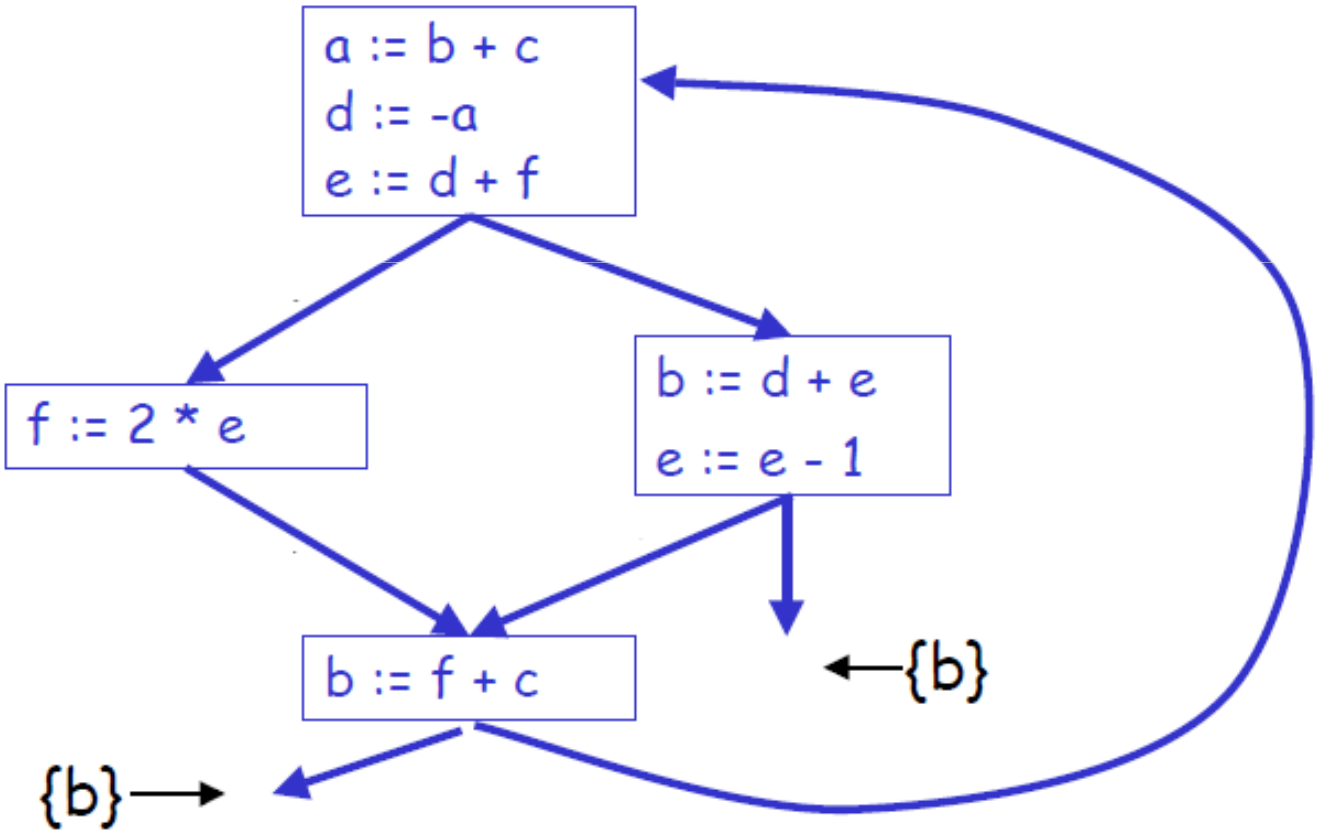
Assignation globale de registres

- Si deux variables ne sont pas vivantes en même temps en tout point du programme alors elles peuvent partager un même registre.
- **Question :**

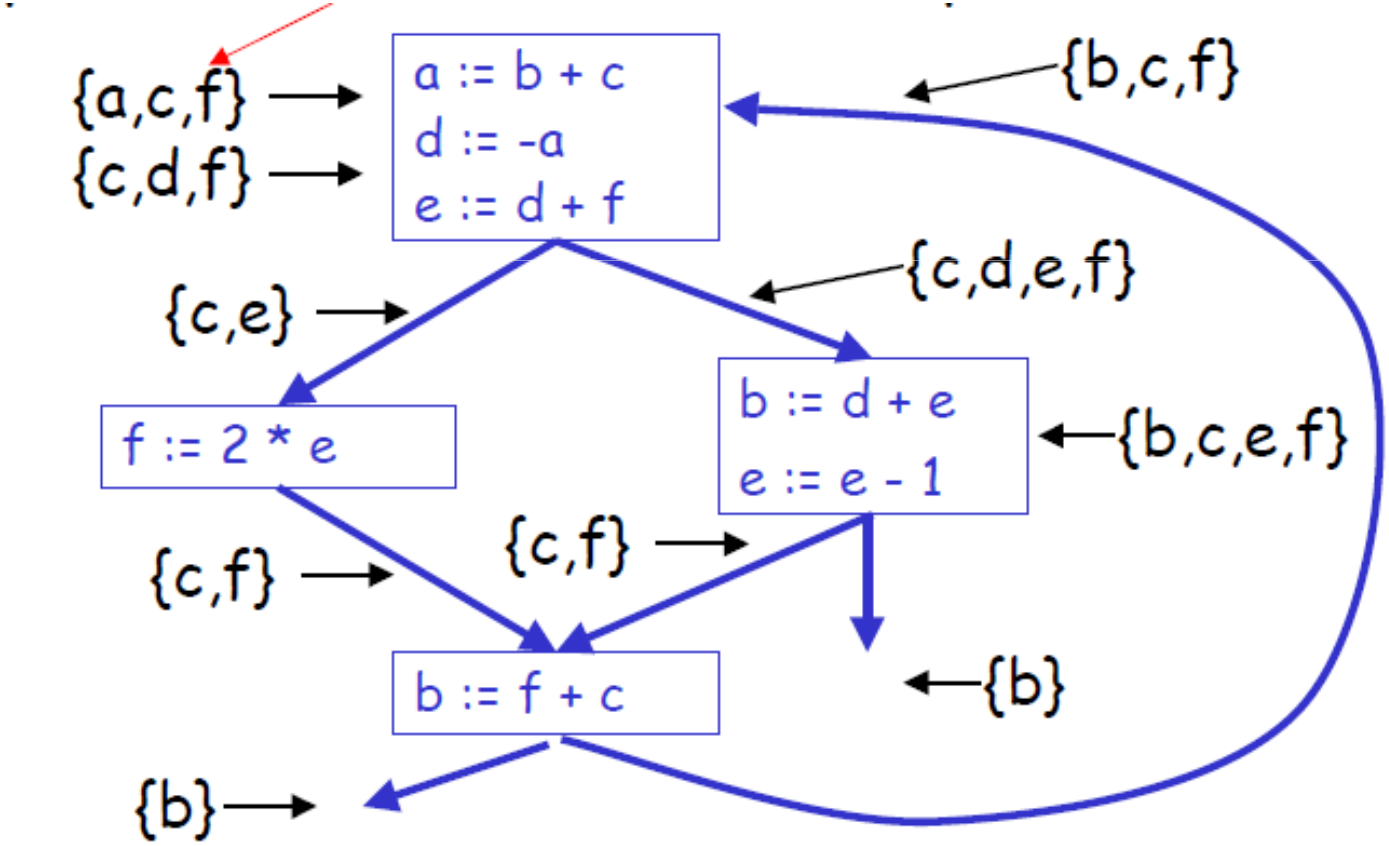
  Un nombre K de registres peut-il contenir toutes les variables du graphe de flot de contrôle ?
- **Comment procéder ?**

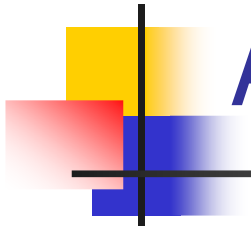
1. Variables vivantes en chaque point du programme.
2. Exploiter le Graphe d'Interférence des Registres.

Assignation globale de registres



Assignation globale de registres





Assignation globale de registres

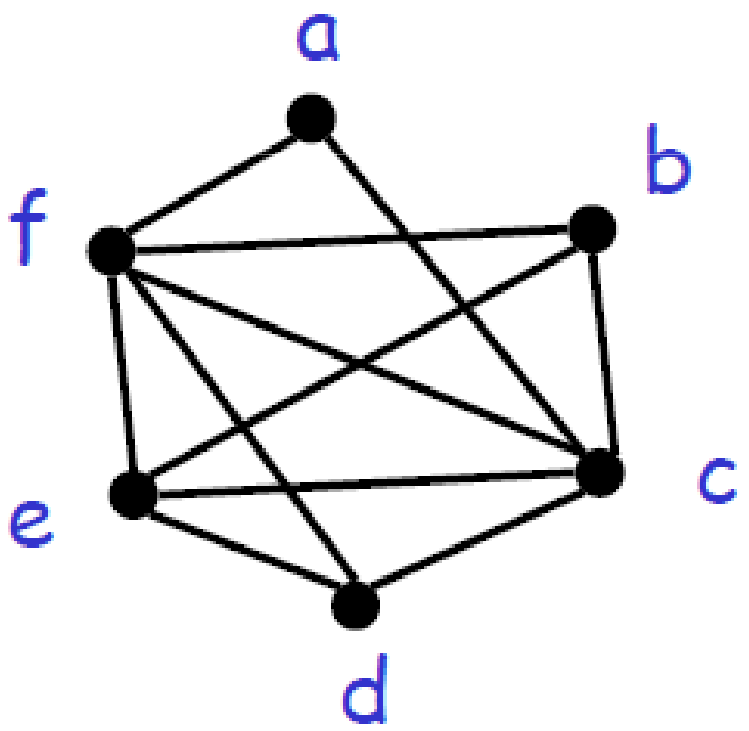
- **Graphe d'Interférence des Registres RIG :**

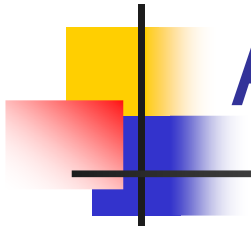
1. Un sommet pour chaque variable
2. Une arête entre a et b ssi elles sont vivantes en même temps en un point du programme

- **Deux variables peuvent être allouées dans un même registre s'il n y a pas d'arête entre eux.**

Assignation globale de registres

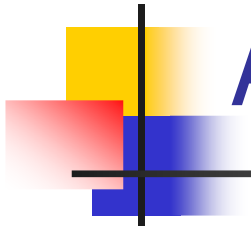
- RIG de l'exemple précédent :





Assignation globale de registres

- **Pour déterminer le nombre minimal de registres nécessaire pour contenir les variables revient à trouver le nombre chromatique du RIG.**
- **Soit K le nombre de registres de la machine.**
- **Si le RIG est K-colorable alors l'assignation des registres n'utilise pas plus de K registres.**



Assignation globale de registres

- Comment calculer le nombre chromatique ?
- Problème NP-difficile (NP-hard).
- Solution par l'usage d'heuristique.

Assignation globale de registres

- **Principe de l'heuristique :**

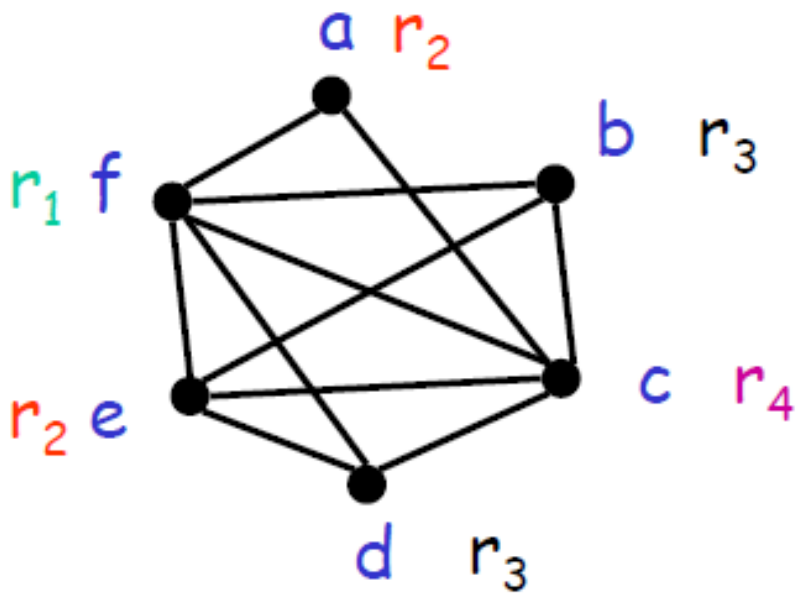
1. Sélectionner un nœud S ayant moins de K voisins dans le RIG
2. Eliminer S du RIG et empiler S

   Si le graphe résultant est K-colorable alors le graphe original est K-colorable
3. Répéter jusqu'à ne pas avoir sommet dans le graphe
4. Assigner des couleurs aux sommets dans la pile (à chaque étape assigner une couleur différente des couleurs de ses voisins déjà colorés.

Assignation globale de registres

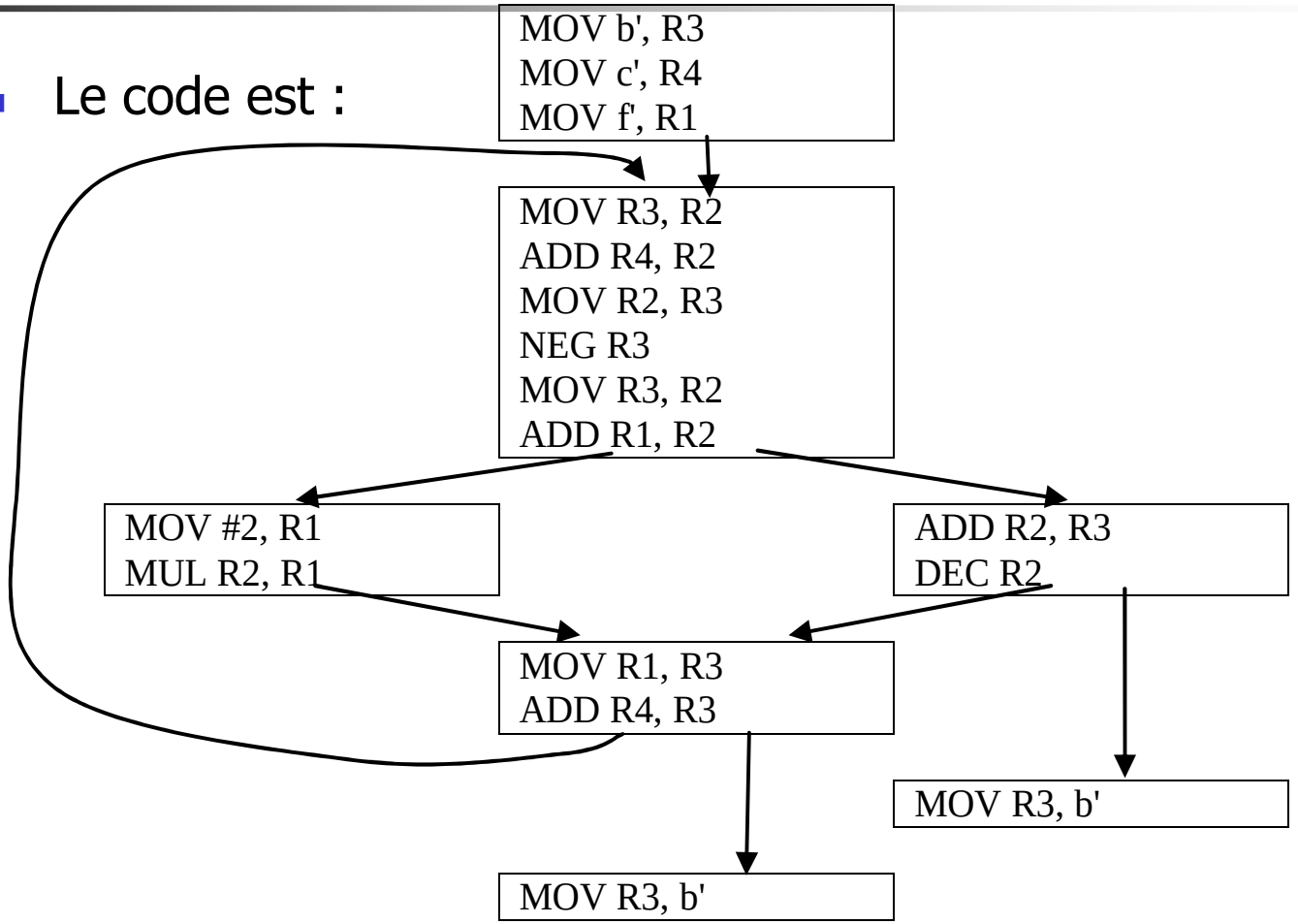
- **Exemple :**

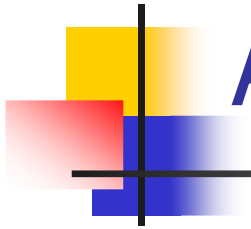
1. Enlever a puis d
2. Enlever c, b, e et f
3. Assigner des couleurs à f,e,b,c,d,a



Assignation globale de registres

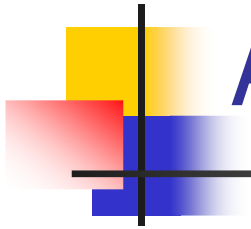
- Le code est :





Assignation globale de registres

- **Problème :**
- Si le nombre de registres disponibles ne suffit par pour contenir toutes les variables ?
- **Privilégier les variables ayant un usage "intensif".**



Assignation globale de registres

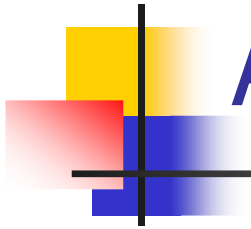
- Définitions de "l'usage" des variables
- Soit B un bloc de base, les ensembles suivants serviront à quantifier l'utilisation des variables du graphe du flot de contrôle.

  IN[B] : ensemble des noms vivants au point immédiatement avant d'entrer en B.

  OUT[B] : ensemble des noms vivants au point immédiatement après la sortie de B.

  DEF[B] : ensemble des noms assignés avant toute utilisation de ce nom en B.

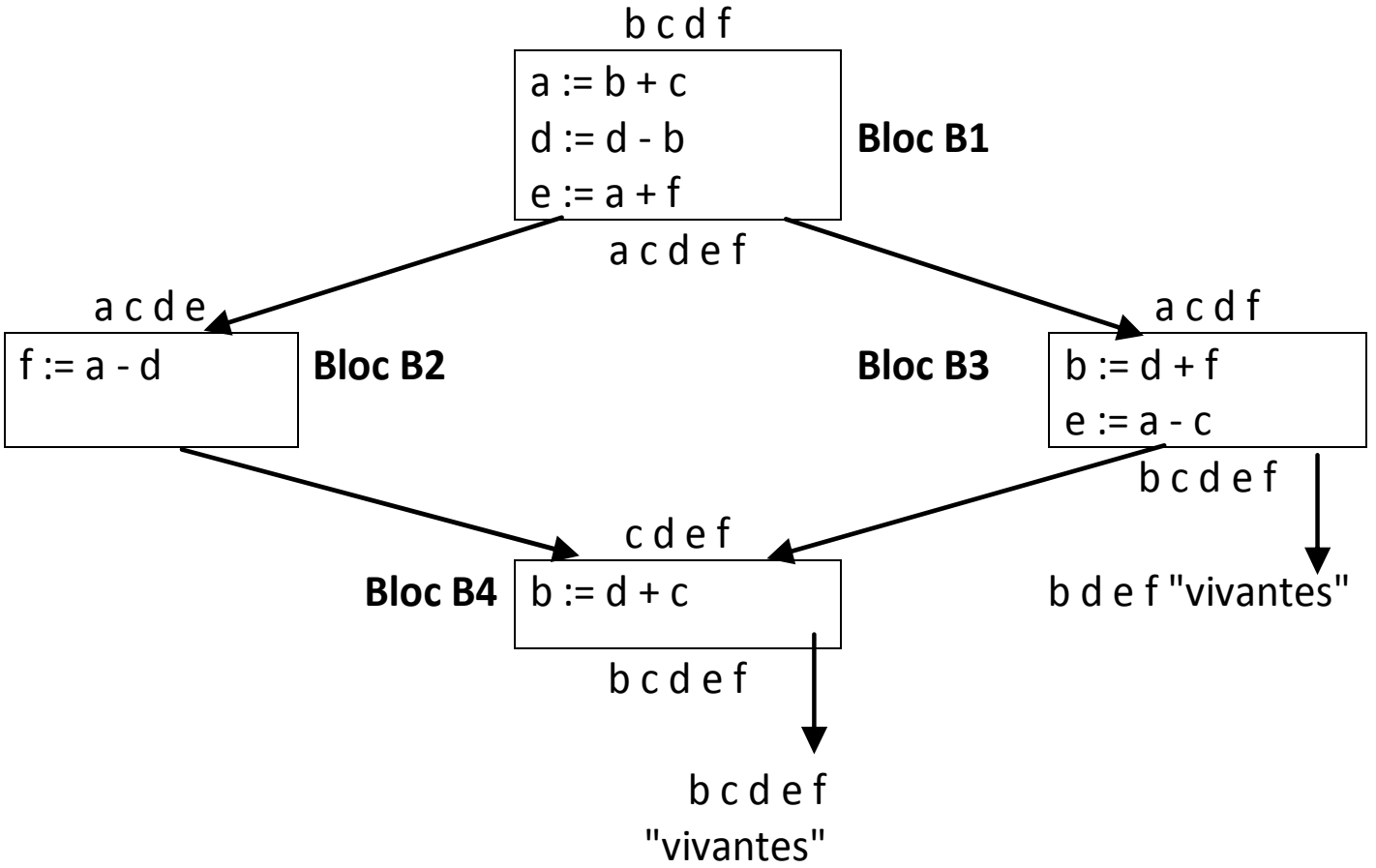
  USE[B] : ensemble des noms utilisés en B avant toute définition.



Assignation globale de registres

- Relations entre ces ensembles :
  - **IN[B] = OUT[B] - DEF[B]** $\cup$ **USE[B]**
  - **OUT[B] =** $\cup$ **IN[S]** où S est un successeur de B

Assignation globale de registres



Assignation globale de registres

- Compteur d'usage
- Le compteur d'usage d'une variable a dans une boucle L est donné par la formule suivante :

$$\sum_{\substack{\text{Blocs B} \\ \text{Dans L}}} (\text{USE}(a \text{ dans B}) + 2*\text{LIVE}(a \text{ dans B}))$$

- **USE[a,B]** : Nombre de fois où a est utilisé dans B avant toute définition de a.
- **LIVE[a,B] = 1** si a est vivante en sortie de B et a est assignée dans B; **(= 0** sinon)

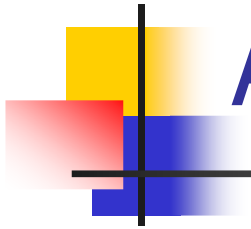
Assignation globale de registres

- ***Stratégie d'allocation***

  Affecter des registres aux variables ayant des compteurs d'usage les plus élevés.

- Compteurs d'usage des variables de l'exemple précédent :

Variable	a	b	c	d	e	f
Compteur d'usage	**4**	**6**	**3**	**6**	**4**	**4**



Assignation globale de registres

- Donner le code produit pour le graphe de flot de contrôle précédent sachant que les registres R0, R1 et R2 seront assignées aux variables (a, b et d) ayant un compteur d'usage élevé. La variable a été arbitrairement choisi par rapport à e t f.

Assignation globale de registres