Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Electronique et d'Informatique
Département d'Informatique

Master 1 :IL Année Universitaire : 2023/2024
Module Compilation

Projet de compilation du langage MinIL
Avec les Outils FLEX et BISON

1. Introduction :

Le but de ce projet est de réaliser un mini-compilateur passant par les différentes phases de la compilation à savoir l'analyse lexicale en utilisant l'outil FLEX et l'analyse syntaxico-sémantique en utilisant l'outil BISON. La génération du code intermédiaire, l'optimisation ainsi que la génération du code machine se feront en langage C.

Les traitements parallèles concernant la gestion de la table des symboles ainsi que le traitement des différentes erreurs doivent être également réalisés lors des phases d'analyse du processus de compilation.

2. Description du Langage MinIL:

La structure générale d'un programme est la suivante :

VAR_GLOBAL {
}
DECLARATION {
}
INSTRUCTION {
}

2.1. Commentaire :

Un commentaire est ignoré par le compilateur. Il ne peut être que sur une seule ligne.

Exp : **%%** commentaire

2.2. Déclarations :

Le bloc déclaration comporte des variables simples, des tableaux, des structures de données et des constantes.

2.2.1. Déclaration des variables de type simple:

La déclaration d'une variable a la forme suivante :

TYPE *liste_variables ;*

- ***liste_variables*** est un ensemble d'identificateurs séparés par une virgule.

Exemple :

%%Déclaration d'une variable simple

TYPE *nom_variable1* **;**

%%Déclaration de 2 variables

TYPE *nom_variable2 , nom_variable3 ;*

2.2.2. déclaration des tableaux

La déclaration d'un tableau a la forme suivante :

TYPE nom [taille] ;

2.2.3. TYPE :

Le type peut être : INTEGER, FLOAT.

- **INTEGER :** Une constante entière est une suite de chiffres. Elle peut être signée ou non signée tel que sa valeur est entre **-32768** et **32767**. Si la constante entière est signée, elle doit être mise entre parenthèses.
- **FLOAT :** Une constante réelle est une suite de chiffres contenant le point décimal. Elle peut être signée ou non signée. Si la constante réelle est signée, elle doit être mise entre parenthèses.

2.2.4. Déclaration des Constantes:

La déclaration d'une constante se fait comme suit :

CONST *idf = valeur ;*

- **CONST :** une constante peut prendre une valeur unique correspondant à un des types cités précédemment et qui resté inchangée tout au long du programme.

	Exemple
INTEGER	**Déclaration: INTEGER** a ; **Valeurs de a possible:** 21, (-6) …
FLOAT	**Déclaration: FLOAT** a ; **Valeurs de a possible:** 88.5,(-0.24), (+3.0) …

CONST	**Déclaration: CONST** a =valeur ; **Valeurs de a possibles:** 5, (-6), 0.3 …

2.2.5. Identificateur :

Un identificateur est une suite alpha-numérique qui commence par une lettre majuscule ou minuscule suivie d'une suite de chiffres et lettres minuscules. Un IDF ne doit pas contenir plus de 8 caractères.

Les noms du programme principal, des variables, des structures et des constantes sont des identificateurs.

2.3. Opérateurs arithmétique, logique et de comparaison

- **Opérateurs arithmétique :**
  - **+, - , *, /**

- **Opérateurs logique :**
  - **(expression1 && expression2) :** le et logique.
  - **(expression1 || expression2) :** le ou logique.
  - **(! expression) :** la négation.

- **Opérateurs de comparaison**
  - expression1> expression2
  - expression 1< expression2
  - expression1 >= expression2
  - expression1 <= expression2
  - expression1 == expression2
  - expression1 != expression2

2.4. Les conditions

Une condition est une expression qui renvoie '1' ou '0'. Elle peut prendre la forme d'une expression logique ou de comparaison.

2.5. Instructions :

Affectation	
Description	Exemple
Idf =expression ;	A=(X+7+B)/ (5,3-(-2)) ; A= 0**;**

Condition IF (Si … Alors … Sinon … Fin Si**)**

Description	Exemple

IF(condition) {
instruction 1
instruction2
}
ELSE {
instruction 3
instruction4
}
Note :
Le premier bloc est exécuté ssi la condition est vérifiée.
Sinon le bloc « ELSE » sera exécuté s'il existe.
On peut avoir des conditions imbriquées.
	IF (Aa > Bb){
	Cc=E+2.6;
	}
	ELSE{
	Cc=0;
	}

Boucle (Faire si … faire fait**)**

Description	Exemple

FOR (idf: Initialisation : Pas: Condition_d'arrêt) {
instruction 1
instruction2
}
 	
Note : le bloc d'instructions est exécuté ssi la condition est vérifiée.
On peut avoir des boucles imbriquées.
	FOR (i : 0: 2 :n) {
	i=i+2;
	}

Entrées / Sorties

Description	Exemple
Entrée : **READ (nom_variable);** Sortie : **WRITE ("…", nom_variable, "…");**	… WRITE ("Donner la valeur de A :"); READ (A); WRITE ("La Valeur de A est ", A, "."); …

Boucle (Tantque … faire …fait**)**

Description	Exemple

WHILE (Condition) {
instruction 1
instruction2
}
 	
Note : le bloc d'instructions est exécuté ssi la condition est vérifiée.
On peut avoir des boucles imbriquées.
	WHILE (i <n) {
	i=i+2;
	}

Remarque :

1- **Toute instruction doit se terminer par un point-virgule.**

- ***Associativité et priorité des opérateurs :***

Les associativités et les priorités des opérateurs sont données par la table suivante par ordre croissant :

Opérateur		Associativité
	!	/
Opérateurs Logiques	||	Gauche
	&	Gauche
Opérateurs de comparaison	> >= == != <= <	Gauche
Opérateurs Arithmétiques	+ -	Gauche
	* /	Gauche

3. Analyse Lexicale avec l'outil FLEX :

Son but est d'associer à chaque mot du programme source la catégorie lexicale à laquelle il appartient et de générer la table des symboles. Pour cela, il est demandé de définir les différentes entités lexicales à l'aide d'expressions régulières et de générer le programme FLEX correspondant.

4. Analyse syntaxico-sémantique avec l'outil BISON :

Pour implémenter l'analyseur syntaxico-sémantique, il va falloir écrire la grammaire qui génère le langage défini au-dessus. La grammaire associée doit être LALR. En effet l'outil BISON est un analyseur ascendant qui opère sur des grammaires LALR. Il faudra spécifier dans le fichier BISON les différentes règles de la grammaire ainsi que les règles de priorités pour les opérateurs afin de résoudre les conflits. Les routines sémantiques doivent être associées aux règles dans le fichier BISON.

5. Gestion de la table de symboles :

La table de symboles doit être créée lors de la phase de l'analyse lexicale. Elle doit regrouper l'ensemble des variables et constantes définies par le programmeur avec toutes les informations nécessaires pour le processus de compilation. Cette table sera implémentée sous forme d'une table de hachage. Elle sera mise à jour au fur et à mesure de l'avancement de la compilation. Il est demandé de prévoir des procédures pour permettre de **rechercher** et d'**insérer** des éléments dans la table des symboles. Les variables structurées de type tableau doivent aussi figurer dans la table de symboles.

6. Génération du code intermédiaire

Le code intermédiaire doit être générer sous forme de quadruplets.

7. Optimisation

On considère quatre types de transformations successives appliquées au code intermédiaire:

• **Propagation de copie** (e.g. remplacer t1=t2 ; t3=4*t1 par t1=t2 ; t3=4*t2).

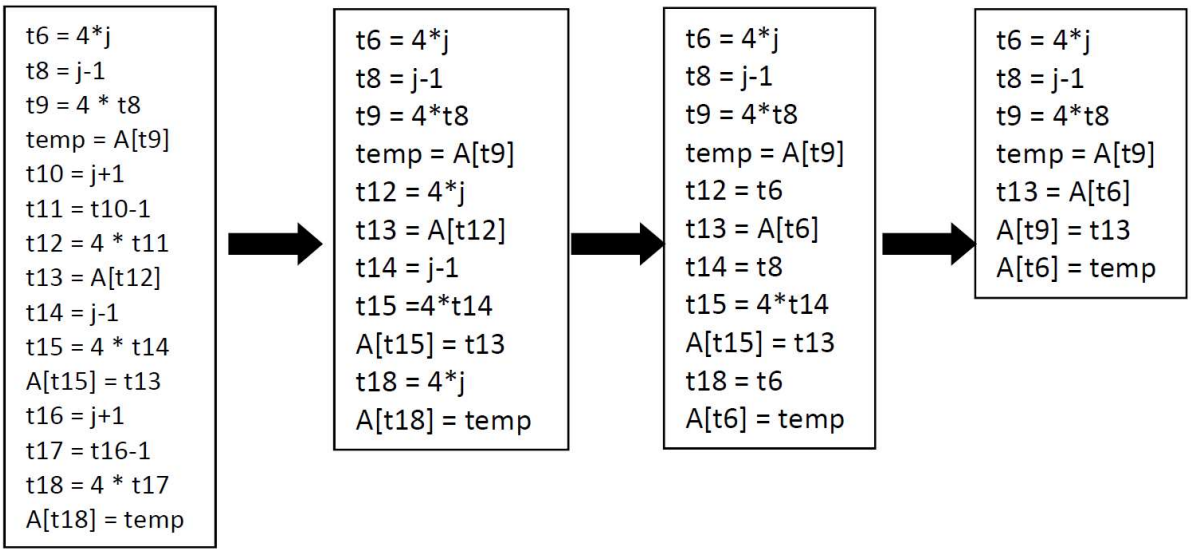
• **Propagation d'expression** (e.g. remplacer t1=expr; t3=4*t1 par t1=expr; t3=4*expr).

• **Élimination d'expressions redondantes** (communes) (e.g. remplacer t6=4*j ; t12=4*j par t6=4*j ; t12=t6).

• **Simplification algébrique** (e.g. remplacer t1+1-1 par t1).

• **Élimination de code inutile** (code mort).

Exemple :



8. Génération du code machine

Le code machine doit être générer en **assembleur 8086**.

9. Traitement des erreurs :

Il est demandé d'afficher les messages d'erreurs adéquats à chaque étape du processus de compilation. Ainsi, lorsqu'une erreur lexicale ou syntaxique est détectée par votre compilateur, elle doit être signalée le plus précisément possible, par sa nature et sa localisation dans le fichier source. On adoptera le format suivant pour cette signalisation :

Type_ de_ l'erreur, line 4, colonne 56, entité qui a générée l'erreur.