Codage assembleur

(Architectures matérielles et systèmes d'exploitation) - Première

Résumé de l'activité	On présente le rôle du langage d'assemblage et quelques instructions élémentaires dans le cours. L'élève interprète l'effet d'enchainements d'instructions puis écrit un petit programme.
Objectif(s)	• Comprendre le rôle l'un langage d'assemblage • Décomposer un programme simple en opérations élémentaires

Durée approximative de l'activité	• 1,5 à 2 heures
Participants	• Elèves de 1ère NSI – Une classe entière
Matériel nécessaire	• Documents fournis : cours (pages 4 à 8) et exercices (pages 10- 11) • Videoprojecteur pour les questions rapides des pages 3 et 9
Prérequis	• processeur, registres, avoir vu au moins un exemple de langage de haut niveau, même sans le maîtriser

Notions liées (Contenus du programme)	Modèle d'architecture séquentielle (Von Neumann)
Lien avec le programme scolaire (Capacité attendue)	Dérouler l'exécution d'une séquence d'instructions simples du type langage machine(première partie)

Déroulement :

1	**Introduction :**	
	Présentation	*Présentation de la nécessité d'un lien entre les langages de haut niveau et le traitement par le processeur.(cours I]page 4 : Les différents niveaux de langage)*

2	**Avant de commencer vraiment**	
	Le page 3 sert à réactiver rapidement les prérequis.	*Questions videoprojetées* *Correction*

3	**cours**	
	Pages 5 à 8 on présente les différentes instructions du langage	*Les élèves devraient être en mesure de remplir les cases écrites en bleu (p 7 et 8 sur le cours avant correction.).*

4	**On vérifie que les instructions de bases sont comprises**	
	Page 9	*Corrigé immédiat*

5	**exercices**	
	Page10-11	

Pour vérifier ses acquis :

1. **Le processeur est :**
   a. Un circuit intégré qui permet de mémoriser des informations.
   b. Un circuit intégré qui exécute des instructions machine.
   c. Un circuit intégré qui exécute des instructions de haut niveau.

2. **L'Unité Arithmétique et Logique (UAL)est :**
   a. Un circuit intégré qui permet de mémoriser des informations.
   b. Un circuit intégré qui exécute des instructions machine.
   c. Un circuit intégré qui permet de réaliser des opérations arithmétiques.

3. **Un registre est :**
   a. Un dispositif d'entrées-sorties.
   b. Un élément de mémorisation inclus dans la hiérarchie mémoire.

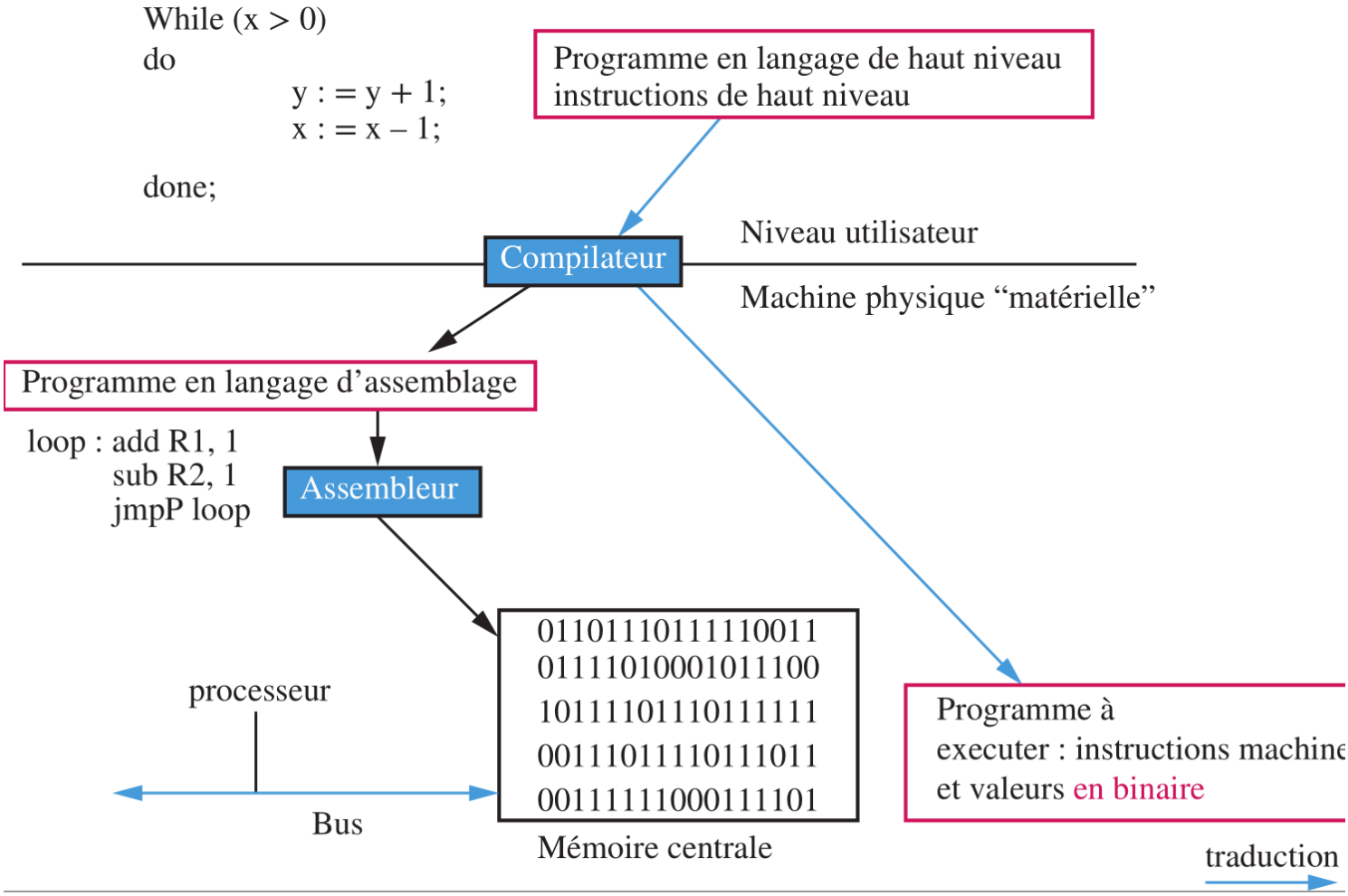
4. **Un registre dédié comme le compteur ordinal (CO) est :**
   a. Un registre qui mémorise les résultats intermédiaires au niveau du processeur.
   b. Un registre avec un rôle fixe : celui de contenir l'adresse de la prochaine instruction à exécuter.
   c. Un registre avec un rôle fixe : celui de contenir la prochaine instruction à exécuter.

5. **Un mot mémoire est :**
   a. Constitué de plusieurs octets
   b. Repéré par une adresse unique.
   c. Un registre dédié comme le CO

Cours :

I] Les différents niveaux de langage

- ***Langage de haut niveau :*** permet à l'utilisateur d'exprimer simplement un algorithme, à l'aide de structures de contrôle proches du raisonnement humain. Ce langage est indépendant de la machine physique, ce qui permet de programmer toujours de la même façon quels que soient l'architecture de l'ordinateur et son type de processeur.
- *Compilateur :* permet la traduction d'un langage de haut niveau vers son équivalent en langage d'assemblage.
- ***Langage d'assemblage :*** les instructions sont des mnémoniques alphanumériques plus facilement mémorisables et utilisables par un être humain. Permet de s'affranchir notamment des codes binaires et des calculs d'adresse.
- Assembleur : permet la traduction du langage d'assemblage vers le langage machine.



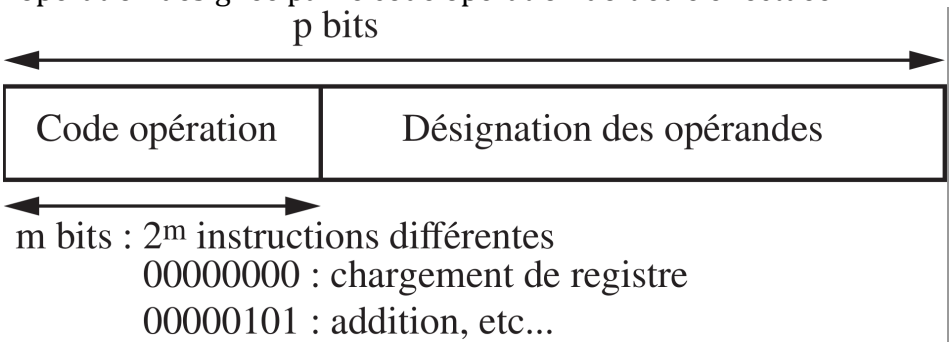
- ***Langage machine :*** au niveau matériel, le processeur a été défini avec un jeu d'instructions machine qui sont des chaînes binaires qui codent des actions élémentaires (addition, chargement de registre, manipulation des mots mémoire) entre les registres et la mémoire et mettent en jeu notamment les circuits de l'unité arithmétique et logique (UAL). Le processeur n'est capable d'exécuter que ces instructions machine.

II] Langage d'assemblage et langage machine

1) Langage machine *:*

Une instruction machine est une chaîne binaire de p bits permettant de spécifier au processeur un traitement élémentaire à réaliser. Elle est composée principalement de deux parties :

- Le champ code opération : il indique au processeur le type de traitement à effectuer (addition, lecture d'une case mémoire,…)
- La champ opérande : il permet d'indiquer la nature des données sur lesquelles l'opération désignée par le code opération doit être effectuée.

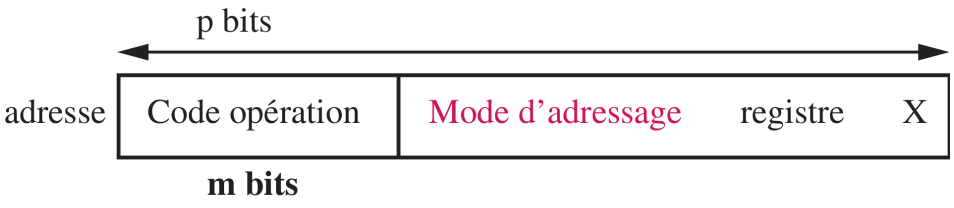


Les instructions du langage machine peuvent être rangées selon quatre catégories :

- ***arithmétiques et logiques :*** permettent de réaliser des calculs (addition, soustraction,…) et des opérations logiques (et, ou …). Elles mettent en jeu les circuits de l'UAL.
- ***Transferts de données :*** permettent de transférer une donnée depuis les registres processeurs vers la mémoire centrale et vice versa, ainsi qu'entre registres du processeur.
- ***Entrées-sorties :*** permettent au processeur d lire une donnée depuis un périphérique (clavier par exemple) ou d'écrire une donnée vers un périphérique (imprimante par exemple).
- ***Rupture de séquence d'exécution :*** permettent de rompre l'exécution séquentielle des instructions d'un programme.

2) Le langage d'assemblage

Une instruction du langage d'assemblage est l'équivalent du langage machine dans lequel chaque champ de l'instruction machine est remplacé par un mnémonique alphanumérique. L'instruction est alors composée de champs, séparés par un ou plusieurs espaces. On identifie un champ étiquette et un champ désignation des opérandes pouvant effectivement comporter plusieurs opérandes.



00001000 00000101 0000 0001 0000000000000000000
addition : LOADIm R1, #000
etiquette : codeopmode_adressage registre, X

a. Les étiquettes

Une **étiquette** ou label est une chaîne de caractères permettant de nommer une instruction ou une variable.
Elle correspond à une adresse dans le programme, soit à celle de l'instruction, soit à celle de la variable.

b. La désignation des opérandes.

L'opération à réaliser spécifiée par le code opération peut porter sur des opérandes de différentes natures :

Un opérande qui est une valeur numérique directement spécifiée dans l'instruction ;
Un opérande qui se trouve dans un mot de la mémoire centrale ;
Un opérande qui se trouve dans un registre du processeur.

Afin de spécifier la nature des opérandes, les instructions machines emploient la notion de mode d'adressage. Il en existe plusieurs :

Mode d'adressage immédiat
Dans ce mode l'opérande est une valeur qui est directement codée dans l'instruction.
Exemple :
LOADIm R1, #100 charge la valeur 100 dans le registre R1.
Le caractère # indique que 100 doit être interprété comme une valeur numérique.

Mode d'adressage adresse.
Dans ce mode, l'opérande est en mémoire centrale et désigné par son adresse.
Il a deux possibilités d'adressage :

le mode d'adressage direct : l'adresse spécifiée dans l'instruction est l'adresse d'un mot contenant l'opérande.

Le mode d'adressage indirect : l'adresse spécifiée dans l'instruction est l'adresse d'un mot contenant lui même l'adresse de l'opérande.

Exemple :

Adresse	Contenu
100	150
150	50

Adressage direct : LOADD R1, @100
Charge l'opérande qui se trouve dans le mot d'adresse 100 dans le registre R1, c'est à dire la valeur 150. Le caractère @ indique que 100 est une adresse.

Adressage indirect : LOADI R1,@100
Charge l'opérande qui se trouve dans le mot d'adresse 150 dans le registre R1, c'est à dire la valeur 50.

Remarque : il existe aussi un mode d'adressage registre.
Nous ne l'utiliserons pas ici. L'adressage registre implique que l'opérande est en mémoire centrale. Son adresse est calculée à partir du contenu d'un registre du processeur tel que le

compteur ordinal ou des registres d'adressage dédiés tels que les registres d'index nommés RIX.

c. Les différents types d'instructions.

Le code opération est une chaîne de caractères mnémoniques (Le code **mnémonique** est un ensemble de quelques lettres, permettant de désigner rapidement une action du code opération binaire.)
Les opérations peuvent être classées en différentes catégories :

- les opérations de chargement, stockage et échanges de registres
  - LOAD
  - STORE
  - SWP
  - MOV
- les opérations arithmétiques et logiques mettant en jeu les circuits de l'UAL :
  - ADD
  - MUL
  - OR
  - XOR
  - AND
  - ……
- les opérations de structure de séquences telles que les opérations de branchements
  - JUMP
  - JUMPO
  - JUMPS
  - JUMPZ
  - ……..

Exemples d'instructions :

Instructions	Signification
LOADIm R1, #100	Le registre R1 est chargé avec la valeur 100.
STORED R1, @100	Le contenu du registre R1 est écrit dans le mot d'adresse 100.
SWP R1, R2	Les contenus des registres R1 et R2 sont échangés.
MOV R1, R2	Le contenu du registre R1 est copié dans R2. Celui de R1 reste inchangé.
ADDD R2, R1, @100	Le contenu du registre R1 est additionné avec l'opérande contenu dans le mot d'adresse 100 et le résultat est stocké dans R2

d. Les instructions de branchement.

Par défaut, le processeur exécute les instructions les unes à la suite des autres, en séquence.
Lors de l'exécution « normale » d'un programme, le processeur lit l'adresse contenue dans le registre IP, incrémente celui-ci pour qu'il pointe vers l'instruction suivante, puis exécute l'instruction contenue à l'adresse qu'il vient de lire. Lorsqu'il rencontre une instruction de saut (ou branchement), celle-ci va lui faire modifier le contenu du registre IP pour qu'il pointe à l'adresse d'une autre instruction.

Les instructions de branchement ont pour but d'interrompre une séquence et de « sauter »à une instruction qui se trouve plus loin ou au contraire en avant .

Il existe deux types de branchement :

_ Les branchements **inconditionnels**, ils sont toujours réalisés car **sans condition**.

Ex : JUMP @108 au lieu d'exécuter l'instruction de la ligne suivante dans le code, le processeur va exécuter l'instruction d'adresse 108 dans le code.(de même, JUMP boucle permet d'exécuter l'instruction repérée par l'étiquette boucle)

_ Les branchements **conditionnels**, ils ne sont réalisés que si une condition est **vraie** au moment ou le branchement doit être réalisé. Si celle-ci est réalisée, le processeur saute à l'instruction demandée, dans le cas contraire il ignore cette instruction et passe automatiquement à l'instruction d'après, comme si cette instruction n'existait pas...

Les conditions pour chacune de ces instructions sont fonction de l'état des registres spécifiques appelés **indicateurs**. C'est le dernier calcul réalisé par l'UAL qui positionne ces indicateurs.

Type de saut	Condition
JMPP	Indicateur S à « positif »
JMPN	Indicateur S à « négatif »
JMPZ	Indicateur Z à « nul »
JMP0	Indicateur 0 à « vrai »

Exemple de programme avec instruction de branchement conditionnel :

adresse	Instruction assembleur	commentaire
0100	LOADD R1 , @ 1500	Charge dans R1 la valeur contenue à l'emplacement mémoire 1500
0104	LOADIm R2 , # 10	Charge la valeur 10 dans le registre R2
0108	ADD R3 , R1 , R2	Charge dans R3, R1+R2
0112	JMPP Pos	Saut indicateur S à positif, si condition vraie réalise l'instruction désignée par l'étiquette pos
0116	STORED R3 , @ 1000	La valeur de R3 est stockée à l'emplacement mémoire 1000
0120	STOP	Directive qui indique la fin d'un programme
	Pos : STORED R3 , @ 2000	La valeur de R3 est stockée à l'emplacement mémoire 2000

Deux cas sont à étudier :

_ **Cas1** : R1 contient la valeur 20, R3 contient donc la valeur 20+10 soit 30.

Le résultat généré par cette addition est positif, la condition sur JMPP est donc vraie et la valeur 30 est stockée à l'emplacement mémoire d'adresse 2000.

_ **Cas 2** : R1 contient la valeur -15, R3 contient donc la valeur -15+10 soit -5.
Le résultat généré par cette addition est négatif, la condition sur JMPP est donc fausse, la valeur -5 est stockée à l'emplacement mémoire d'adresse 1000.

Vérifions que nous avons compris :

1. L'instruction $ADDIm\ R1, R3, \#10$ réalise :
   a. $R1 \longleftarrow R3 + [10]$
   b. $R1 \longleftarrow R3 + 10$
   c. $R1 \longleftarrow [10]$

2. L'instruction $JMPP$:
   a. Effectue toujours un branchement.
   b. Effectue un branchement si le dernier calcul réalisé par l'UAL a donné un résultat positif.
   c. Effectue un branchement si le dernier calcul réalisé par l'UAL a donné un résultat négatif.

3. Si on considère un mode d'adressage direct, le champ X de l'instruction $LOADD\ R1, X$ code :
   a. Une valeur immédiate.
   b. L'adresse en mémoire centrale d'un mot contenant l'opérande.
   c. L'adresse en mémoire centrale d'un mot contenant lui-même l'adresse d'un mot contenant l'opérande.
   d. Un numéro de registre.

4. L'instruction $ADDIm\ R1, R2, X$ effectue une addition entre :
   a. $R2$ et un opérande contenu dans le mot d'adresse X.
   b. $R2$ et la valeur immédiate X.
   c. $R2$ et un opérande dont l'adresse est contenue dans le mot d'adresse X.

Exercices :

1. Ecrire l'instruction assembleur :
   a. On ajoute le contenu du registre R1 à une valeur immédiate 10 et on envoie le résultat dans le registre R3
   b. On écrit le contenu du registre R4 dans le mot d'adresse 300
   c. On charge le registre R4 avec un opérande dont l'adresse se trouve dans le mot d'adresse 100
2. La configuration de la mémoire centrale est donnée ci-dessous :

Adresse	Valeur
100	
104	112
108	100
112	108

   a. Quelle valeur est placée dans le registre R3 par $LOADI\ R3, @112$?
   b. Quelle valeur est placée dans le registre R3 par $LOADIm\ R3, \#112$?
   c. Quelle valeur est placée dans le registre R3 par $LOADD\ R3, @112$?

3. Décrire à chaque ligne ce que réalise le programme suivant

instruction	*action*
carre : DS1	*carre est une étiquette : on définit la variable carre en lui réservant un mot mémoire*
num : DS1,#1	*on définit la variable ……. en lui réservant un mot mémoire et on ……*
LOADD R1, num	
LOADIm R2, #5	
Boucle : MULD R3, R1, #1	
ADDIm R1, R1, #1	
STORED R1, carre	
ADDIm R2, R2, # -1	
JUMPZ fin	

JMP boucle	
STOP	

Remplir le tableau d'état des registres:

R1	R2	R3	R1	Carre	R2

4. Ecrire un programme en langage d'assemblage qui réalise le calcul suivant :

$$B = (A \times 5) + (6 + B)$$

où A et B sont deux variables correspondant chacune à un mot mémoire.

Facultatif :

5. Sans utiliser l'instruction de multiplication, écrivez un programme qui réalise le calcul

$$A = A + (n \times 5)$$

où A et n sont deux variables lues en mémoire centrale.
Consigne : utiliser au moins une fois un mode d'adressage indirect).

Adresse	Valeur
100	A
104	108
108	n