

M2101: Assembleur

Architecture et Programmation des mécanismes de base d'un système informatique

Mars 2019 – Juin 2019

UE21 Architecture matérielle - Systèmes d'exploitation
- Réseaux



Sophie Laplace

S2 IUT Département Informatique

Introduction

- **Séquence pédagogique:**

- 8 cours: 2 x 1 semaine + 2 x 3 semaines
- 7 TD: 1 x 7 semaines
- 6 TP: 1 x 6 semaines
- 1 contrôle de 1H30

- **Objectif:**

Savoir développer des applications simples mettant en œuvre les mécanismes de bas niveau d'un système informatique

Introduction

- **Contenu:**
 - **Langages de programmation de bas niveau**
 - **Mécanismes de bas niveau d'un système informatique**
 - **Étude d'un système à microprocesseur ou microcontrôleur (réel ou simulé) avec ses composants (mémoires, interfaces, périphériques, etc.)**

Introduction

- Modalités de mise en oeuvre:
 - Utilisation du langage C et/ou d'un langage d'assemblage (**assembleur**)
 - Observation de l'exécution pas à pas d'un programme à l'aide d'un outil de **simulation**/déverminage d'un processeur simple
 - Développement de programmes simples permettant d'illustrer les principaux mécanismes de bas niveau d'un système informatique
 - Étude des mécanismes de gestion des interruptions

Plan du module

1. **Modèle de microprocesseur**
2. **Programmation en assembleur**
3. **Procédures et fonction**
4. **Programmation avancée**
5. **Interruptions**

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Modèle de microprocesseur

Matériel

- **Constitution d'un microordinateur**
 - Carte mère
 - Alimentation
 - Mémoires de masse
 - Périphériques
- **Constitution d'une carte mère:**
 - Micro-processeur
 - Mémoire (vive et disque dur)
 - Unité d'échange: Chipset (ensemble de circuits intégrés) gérant les transferts de données
 - Connecteurs d'E/S (série, parallèle...)
 - Connecteurs d'extension (cartes mémoires, vidéo...)
 - Connecteurs d'alimentation

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Modèle de microprocesseur

Définitions

**Microprocesseur ↔ Unité Centrale de Traitement
↔ CPU (Central Processing Unit)**

CPU= UC +UT+registres

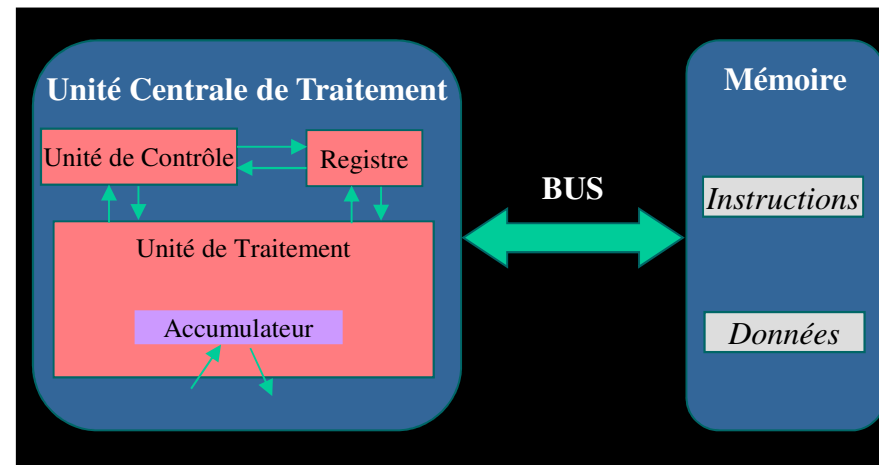
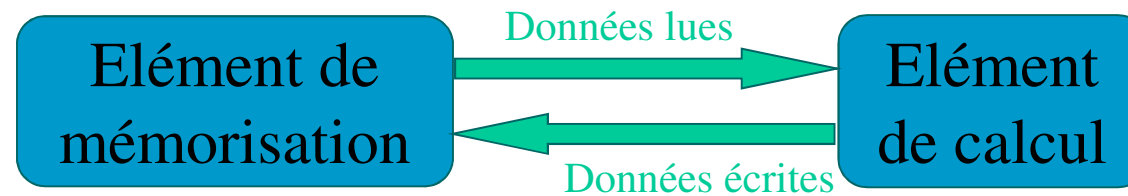
- **UC (Unité de contrôle):**
 - Séquenceur: description en séquence des opérations élémentaires qui permettent l'exécution d'une instruction
 - Ordres à tous les organes du microprocesseur
- **UT (Unité de traitement) ↔ UAL (Unité Arithmétique et Logique) ↔ Unité de calcul:**
 - Réalisation des calculs
 - Résultats dans l'accumulateur

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Modèle de microprocesseur

Modèle de Von Neumann

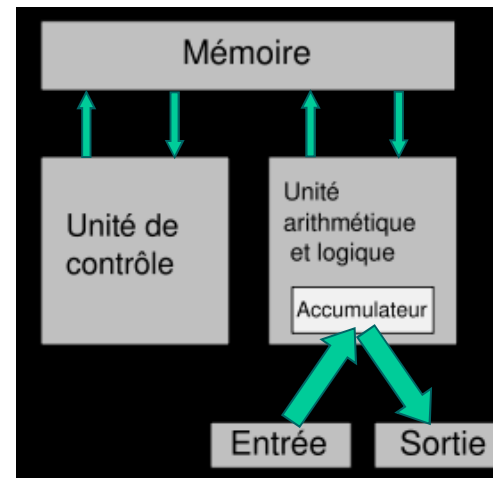


- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

Modèle de microprocesseur

Vue externe: par le programmeur

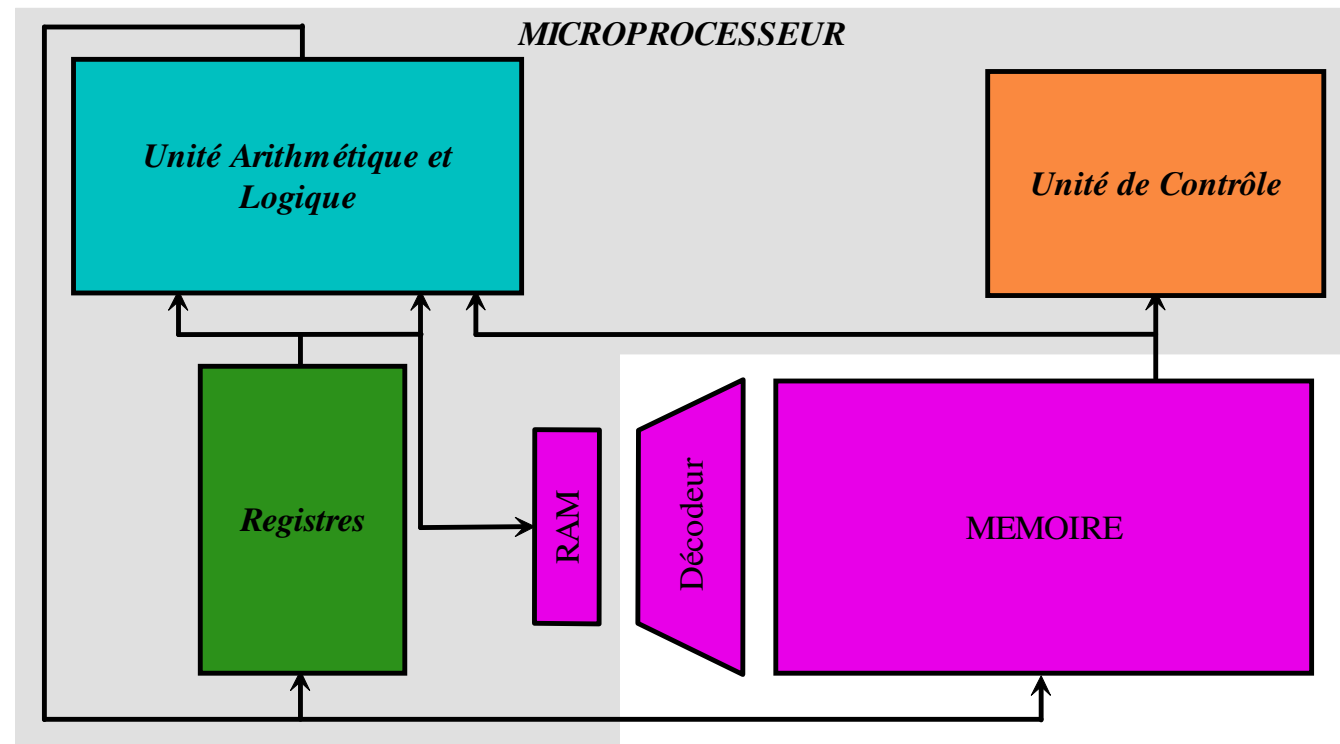
- Autre représentation



- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

Modèle de microprocesseur

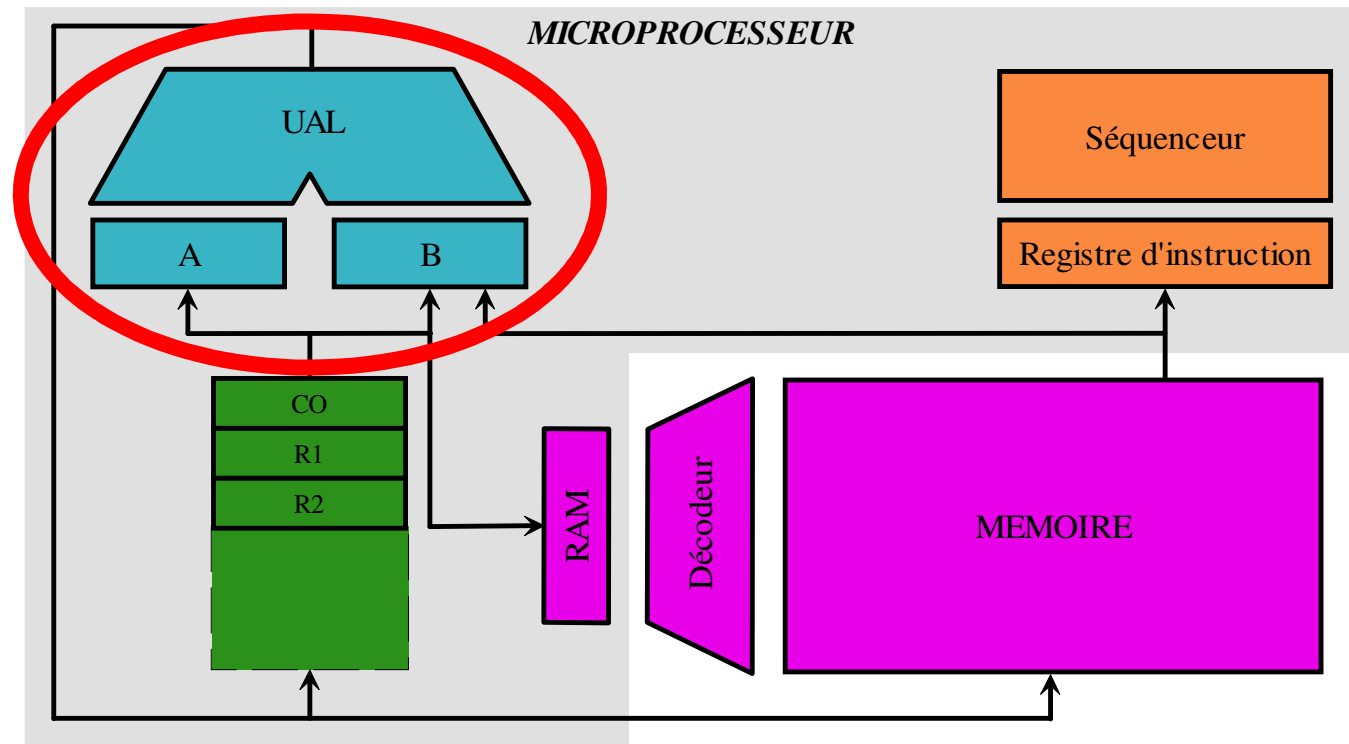
Définitions



- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

Modèle de microprocesseur

Modèle simplifié de processeur (Dalmau)

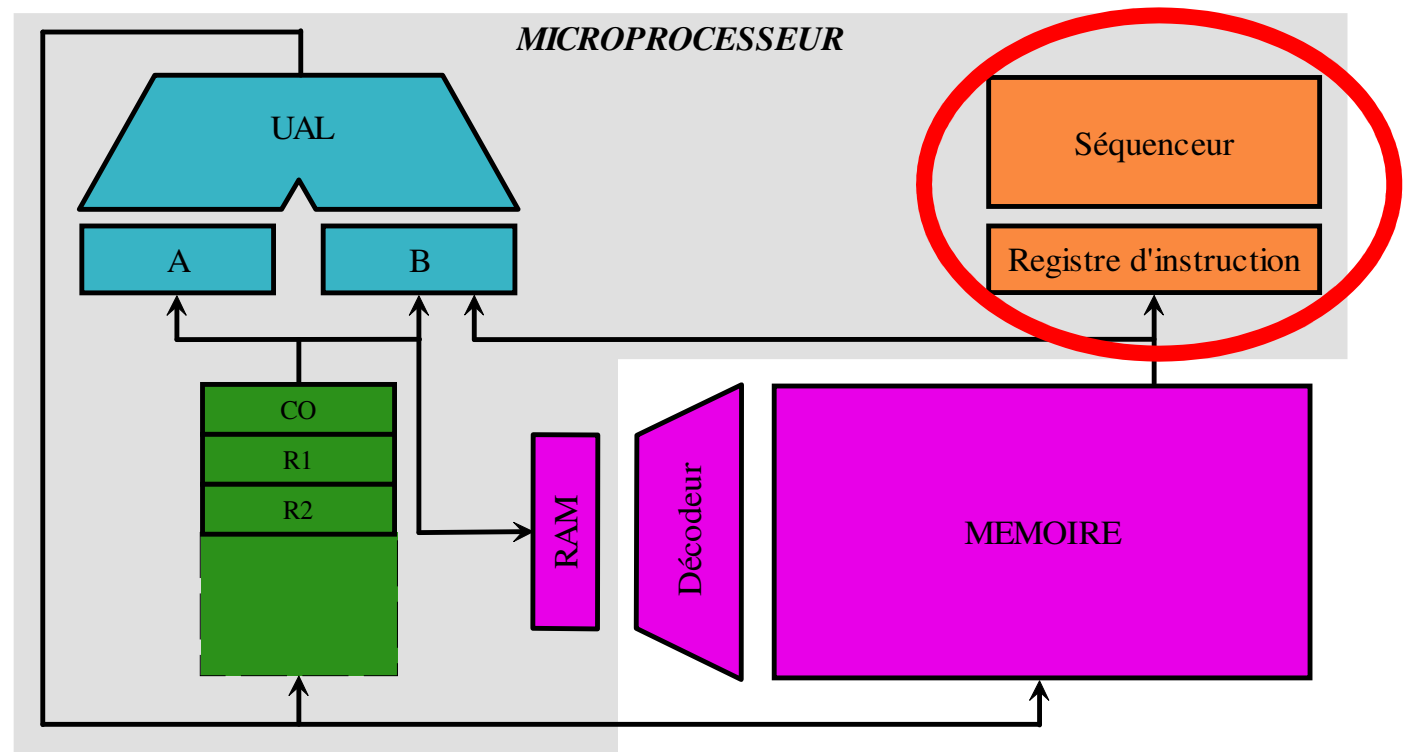


- **UAL (Unité Arithmétique et Logique) = UT (Unité de traitement) = Unité de calcul:**
 - Réalisation des calculs
 - Résultats dans l'accumulateur

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

Modèle de microprocesseur

Modèle simplifié de processeur (Dalmau)

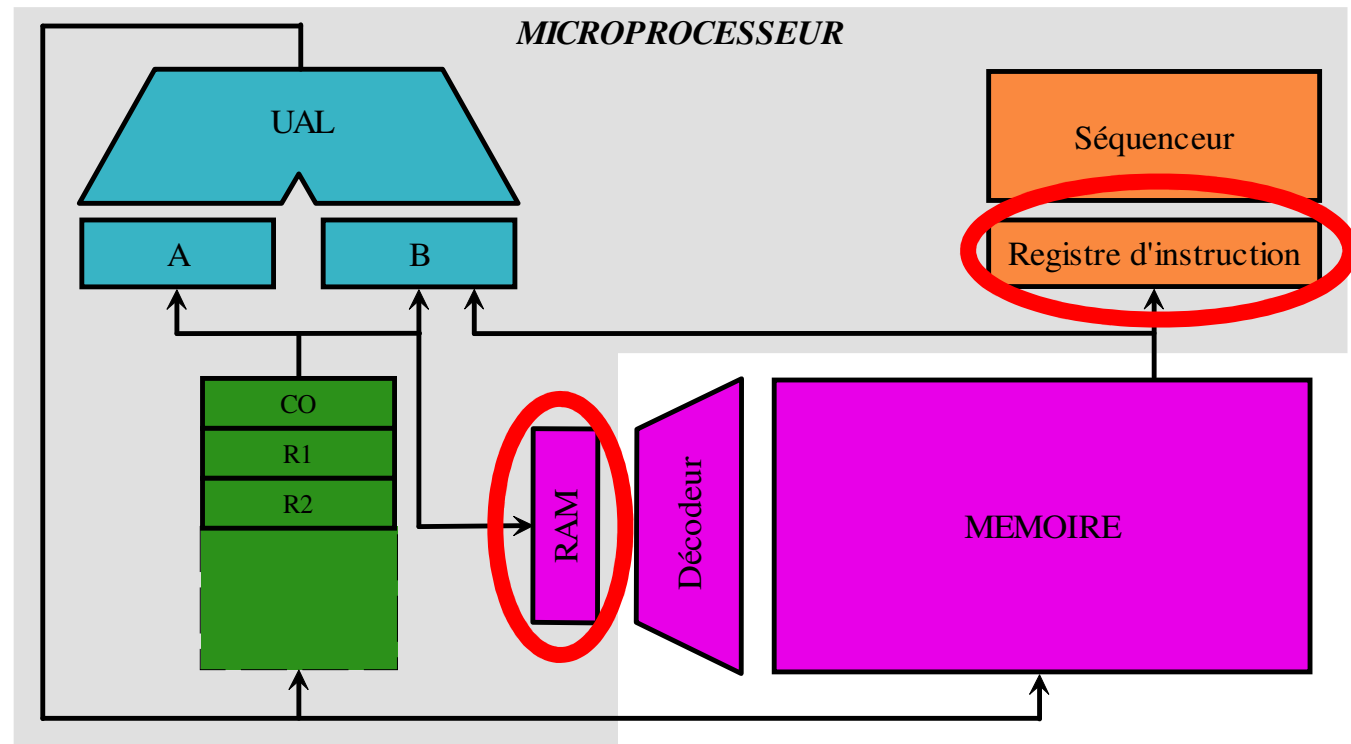


- **UC (Unité de contrôle):**
 - Séquenceur: description en séquence des opérations élémentaires qui permettent l'exécution d'une instruction
 - Câblé ou microprogrammé
 - Ordres à tous les organes du microprocesseur

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

Modèle de microprocesseur

Modèle simplifié de processeur (Dalmau)

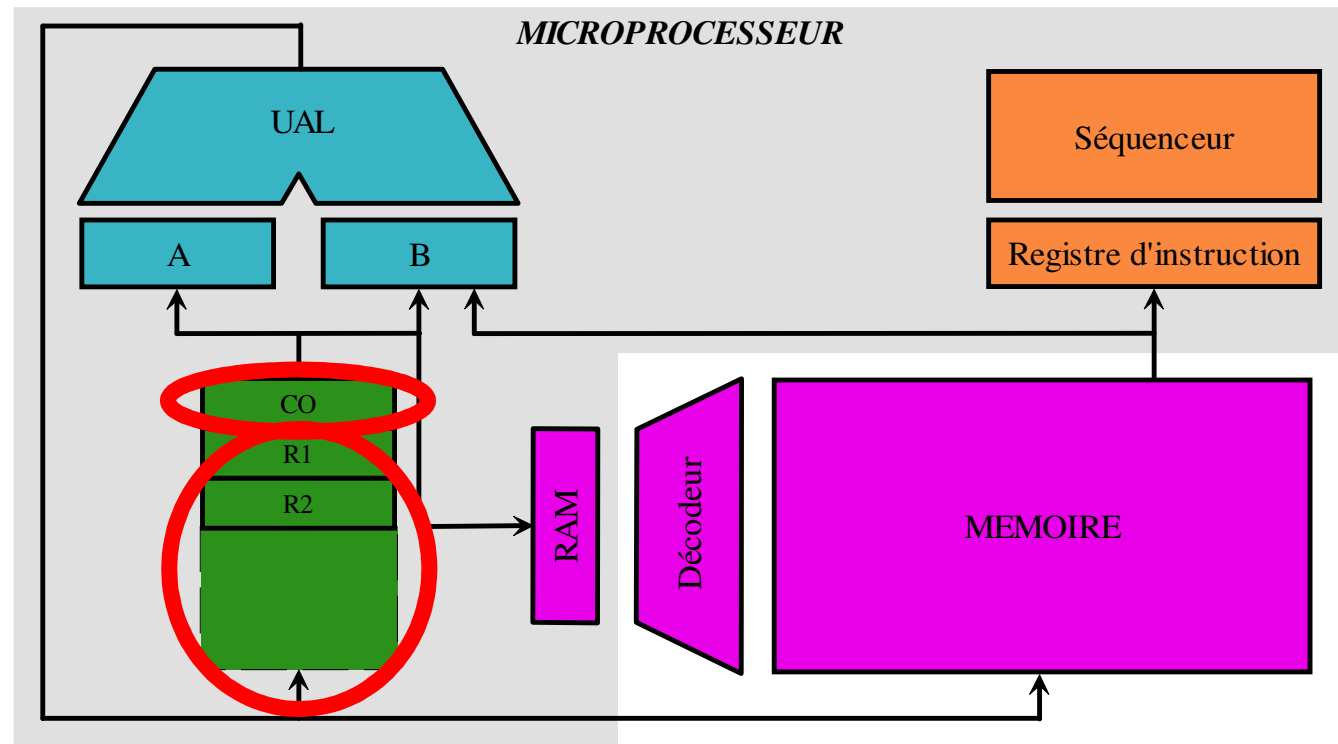


- **Registre d'instructions (RI):**
Instruction en cours d'exécution
- **Registre d'adresses mémoire (RAM)**
Adresse mémoire de la donnée à laquelle accéder

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

Modèle de microprocesseur

Modèle simplifié de processeur (Dalmau)



- **Compteur Ordinal (CO):**
Adresse de la prochaine instruction à exécuter
- **Registres de données (Ri):**
Stockage des informations à stocker ou extraire de la mémoire centrale

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

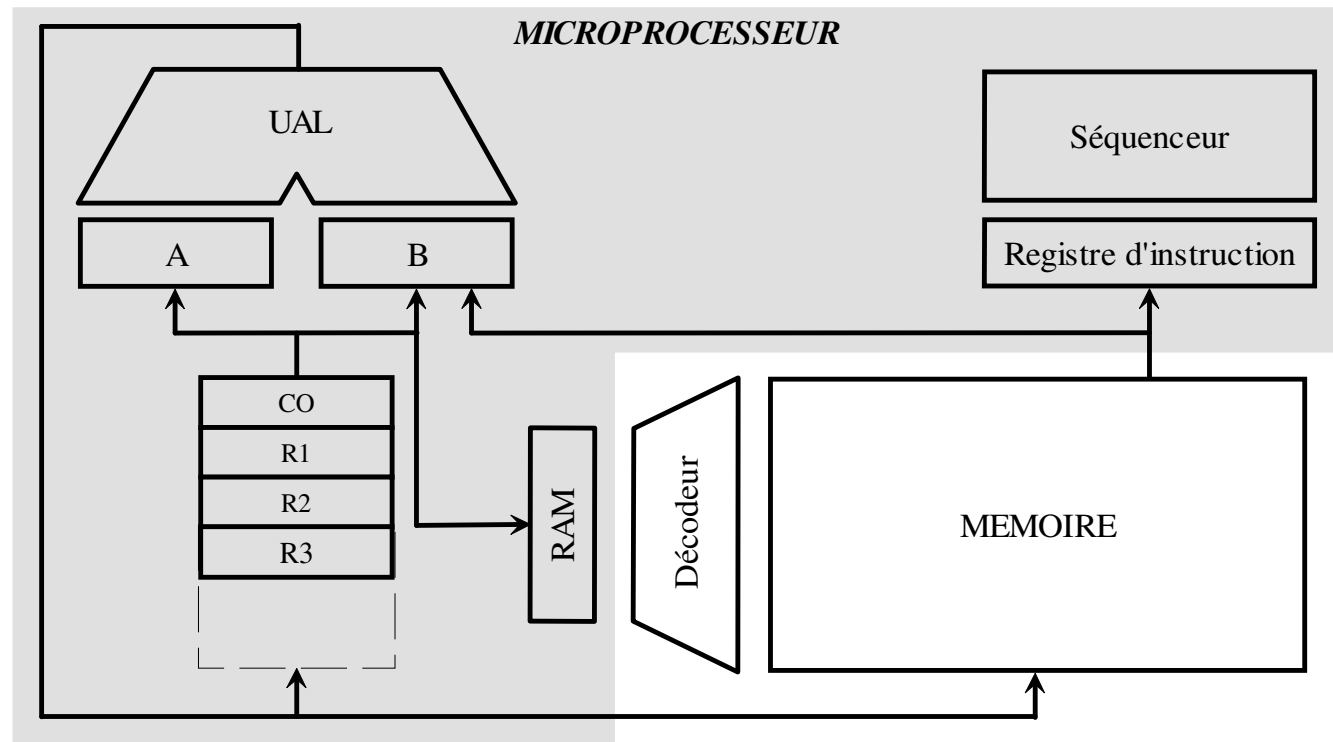
Modèle de microprocesseur

Modèle simplifié de processeur (Dalmau)

- Exécution de l'instruction: $R2 \leftarrow R2 + R3$

ADD	R2	R3
-----	----	----

Code opération 1ère opérande 2ème opérande
et résultat



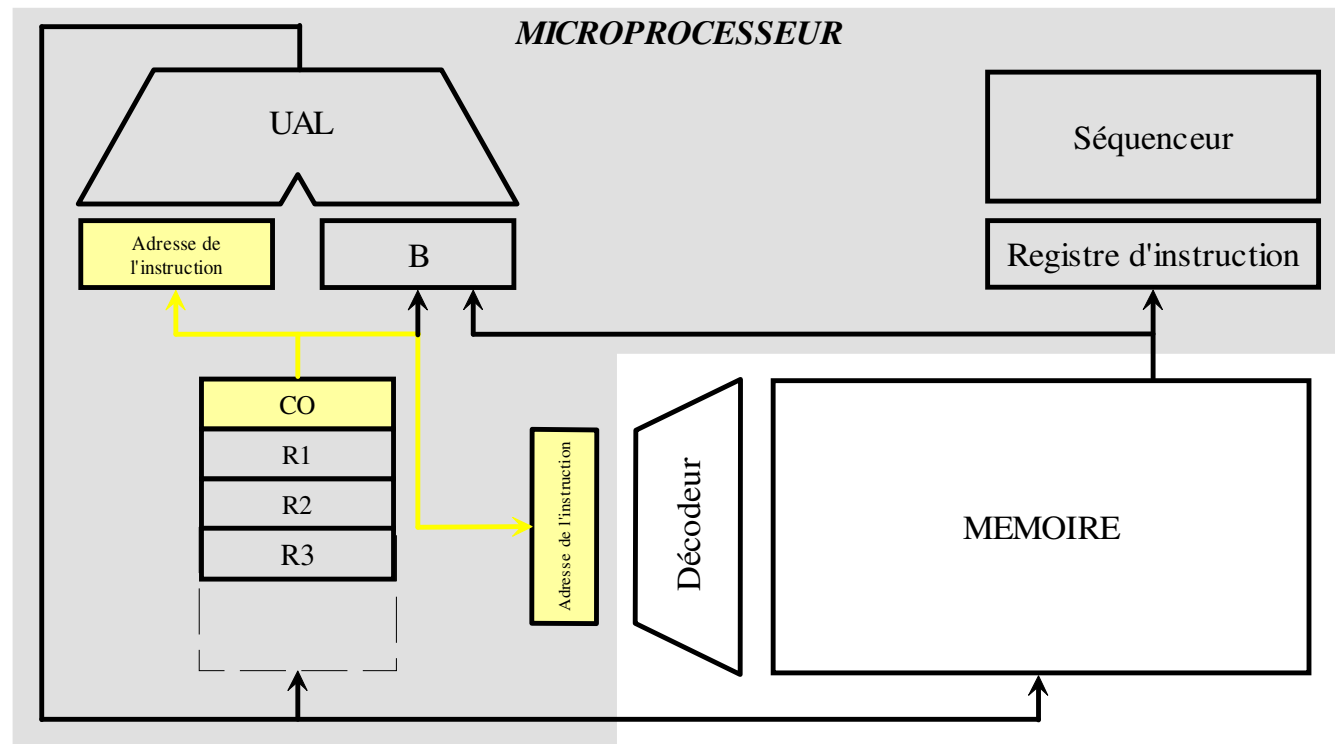
- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

Modèle de microprocesseur

Modèle simplifié de processeur (Dalmau)

- Boucle d'exécution: étapes permettant de réaliser l'instruction

1. Compteur Ordinal: $CO \rightarrow A$ et $CO \rightarrow RAM$



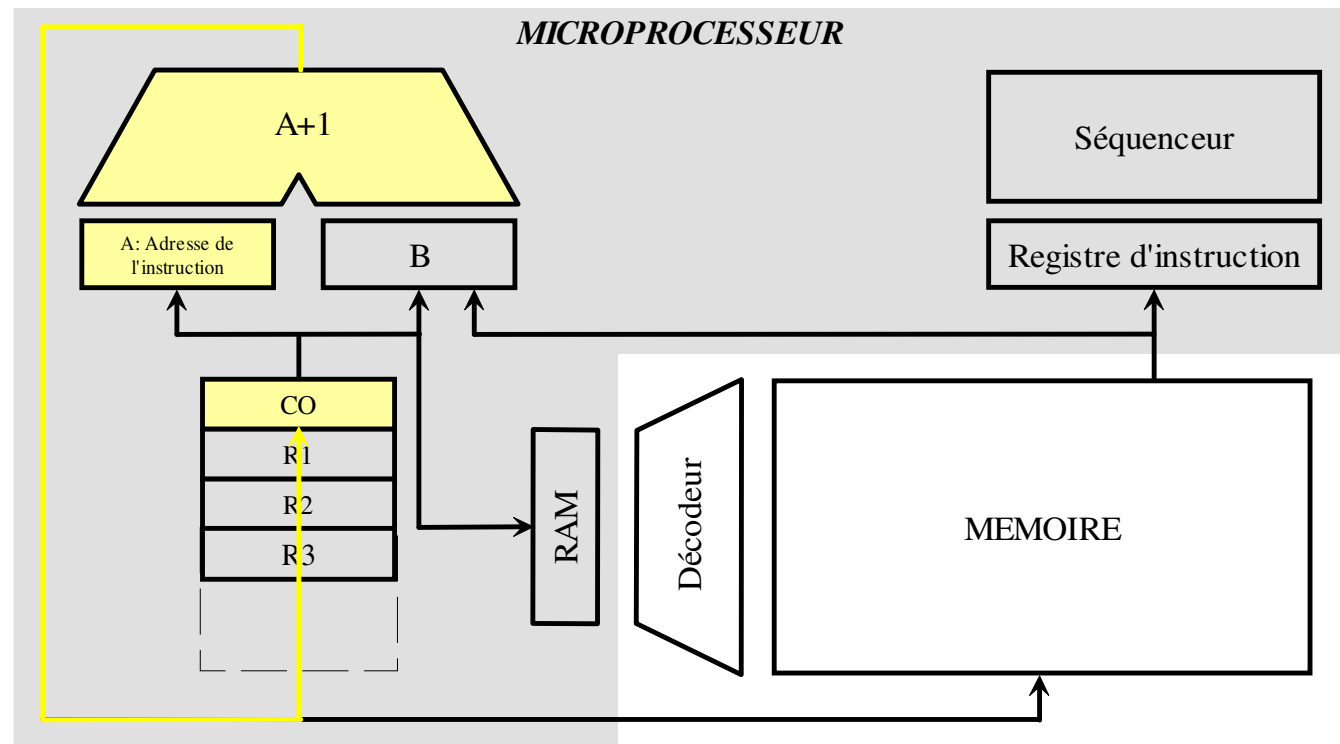
- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

Modèle de microprocesseur

Modèle simplifié de processeur (Dalmau)

- Boucle d'exécution: étapes permettant de réaliser l'instruction

2. UAL incrémente le registre A (CO) et le replace dans CO



- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

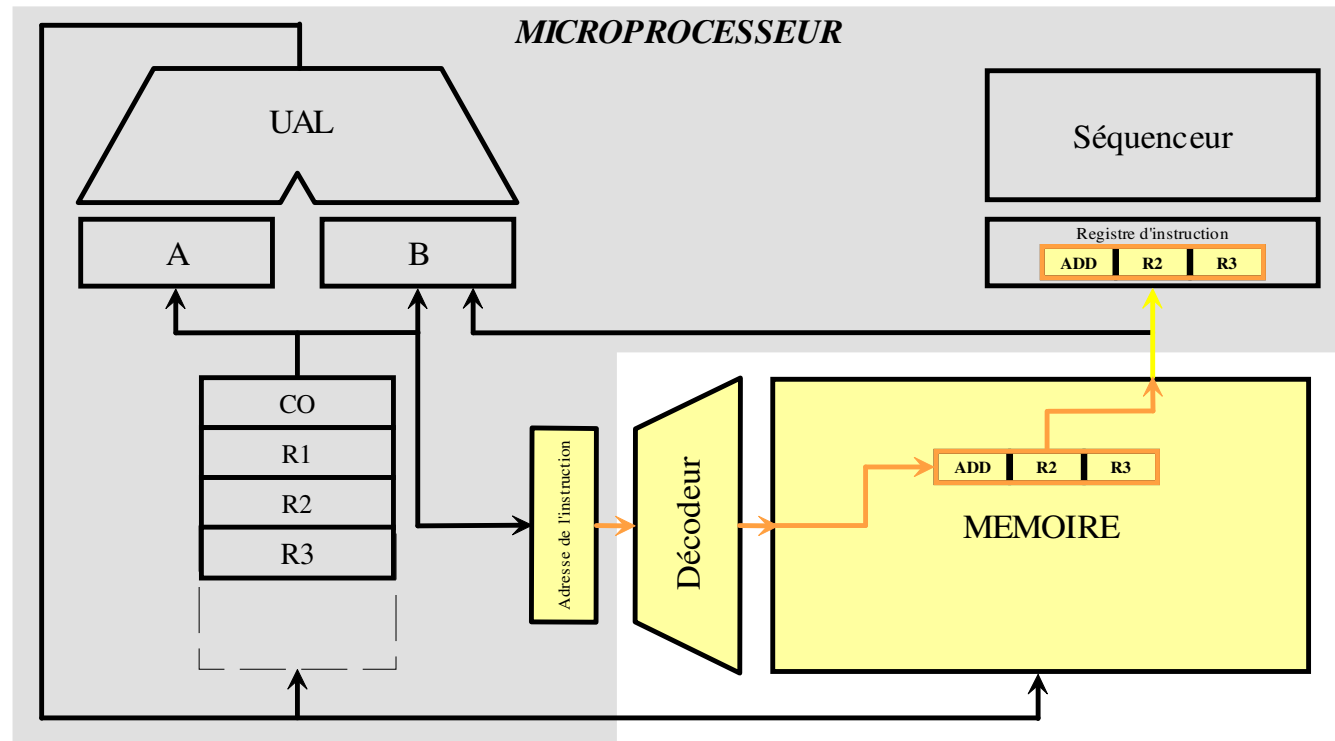


Modèle de microprocesseur

Modèle simplifié de processeur (Dalmau)

- Boucle d'exécution: étapes permettant de réaliser l'instruction

3. Lecture en mémoire de l'instruction ADD R2 R3 et chargement dans le registre d'instruction RI



- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

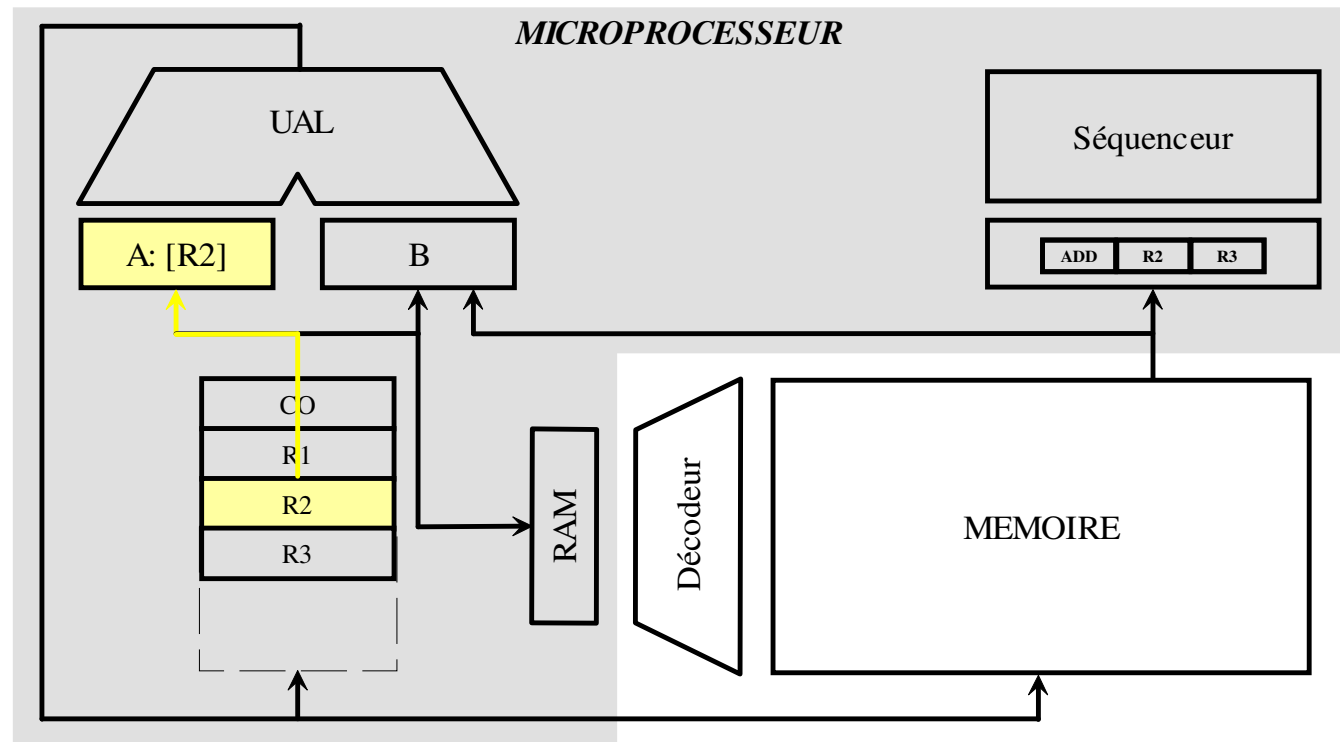
Modèle de microprocesseur

Modèle simplifié de processeur (Dalmau)

- Boucle d'exécution: étapes permettant de réaliser l'instruction

4. $R2 \rightarrow A$

notation: $[R2]$ contenu de R2



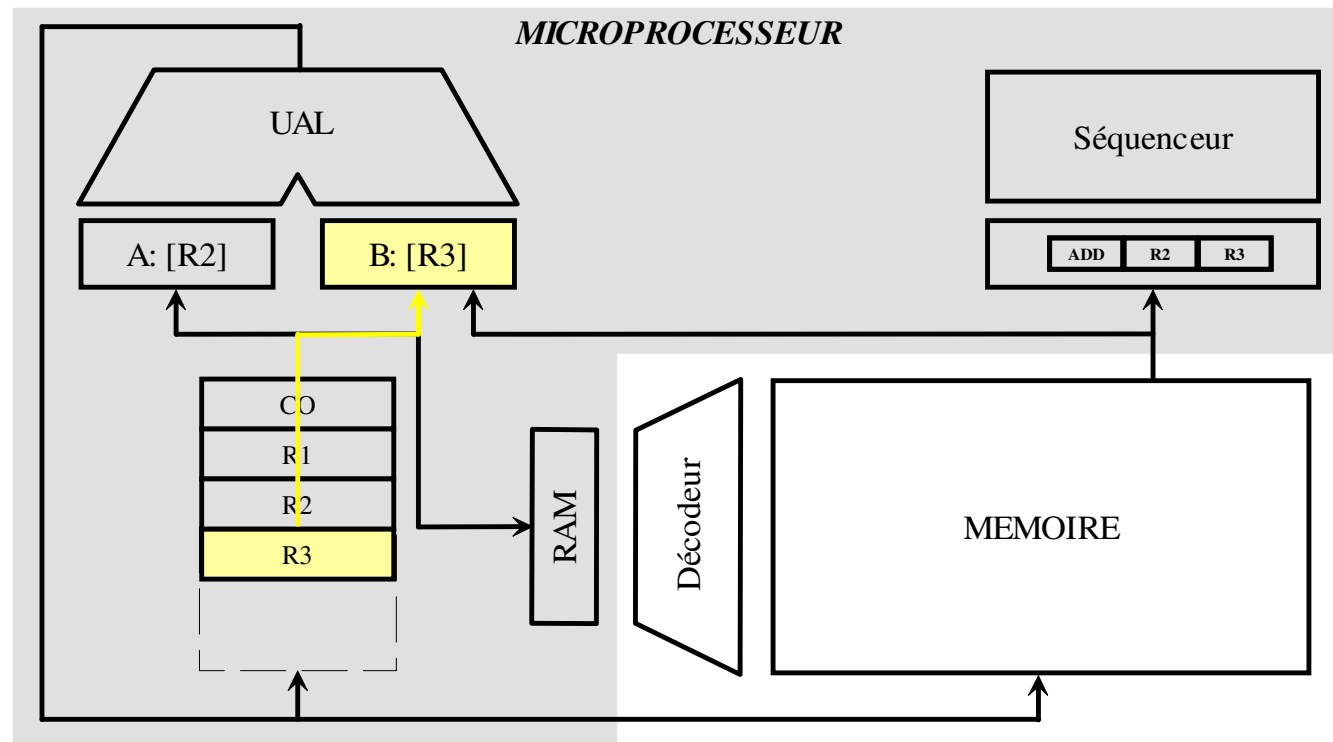
- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

Modèle de microprocesseur

Modèle simplifié de processeur (Dalmau)

- Boucle d'exécution: étapes permettant de réaliser l'instruction

5. $R3 \rightarrow B$



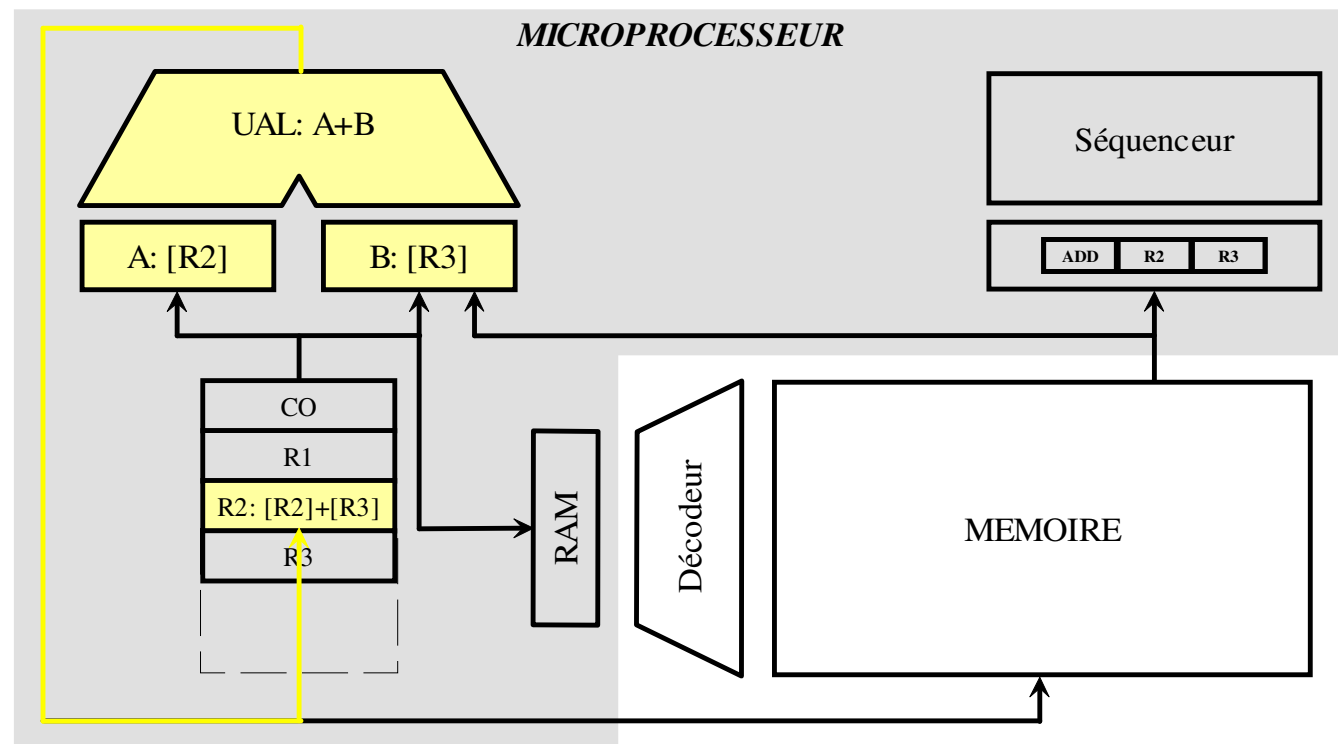
- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

Modèle de microprocesseur

Modèle simplifié de processeur (Dalmau)

- Boucle d'exécution: étapes permettant de réaliser l'instruction

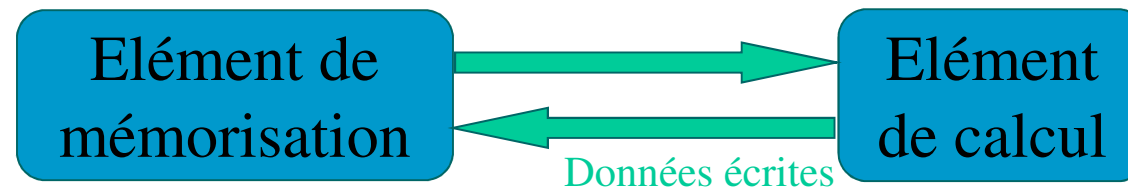
6. UAL calcule $R2+R3$ et résultat placé dans R2



- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

Modèle de microprocesseur

Modèle simplifié de processeur (Dalmau)



- **Boucle d'exécution:**
 - **Passage à l'instruction suivante:**
 - Lecture du compteur ordinal, incrémentation
 - Lecture en mémoire de l'instruction
 -
 - **Orchestration des trois dernières étapes par le séquenceur: dépendance vis-à-vis de l'instruction**
 4. $R2 \rightarrow a$
 5. $R3 \rightarrow b$
 6. UAL calcule $R2+R3$ et résultat placé dans R2

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Définitions

- **Le μ p vu par un programmeur**
 - μ p: exécute des instructions
 - **Jeu d'instruction**
 - Transferts → Mémoire
 - Traitement → UT=UAL
 - E/S → Unité d'échange
 - Ruptures de séquences → UC
 - **Utilisation de registres comme opérandes à ces instructions \Rightarrow accès plus rapide qu'en mémoire**
 - RISC (*reduced instruction-set computer*) : registres généraux
 - CISC (*complex instruction-set computer*) : registres spécialisés
 - **Registres inaccessibles au programmeur: CO, RI, RAM**

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Définitions

- **L'UE vue par un programmeur:**
 - **UE = registres accessibles par les instructions d'E/S**
 - **Lecture des registres d'état → ce que fait ou a fait l'UE**
 - **Ecriture dans les registres de commande:**
 - paramétrer le fonctionnement de l'UE
 - commander le fonctionnement de l'UE (faire faire)
 - **Registres de données: échange d'information avec les périphériques**

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Présentation

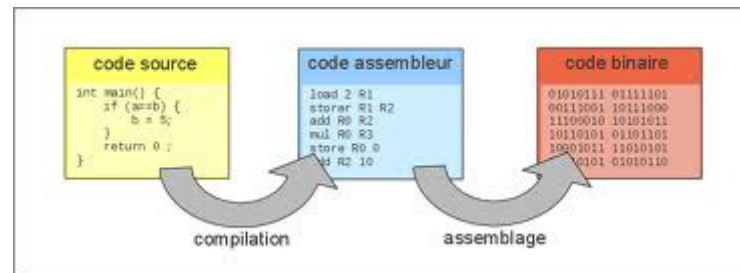
- **Programmation: suite d'instructions**
- **Instruction= code de l'opération+ désignation des opérandes**
LD R0,4
- **Instructions connues par le μ p:**
 - en binaire
 - en assembleur:
 - Code opération: mnémoniques (LD, ADD, JMP...)
 - Opérandes: noms de registres (R5...), variables

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

Programmation en assembleur

Présentation

- Passage du langage aux instructions du μ p:
 - Langage de programmation: un compilateur
 - Langage machine: un assembleur



Compilateur

Assembleur

Haut niveau

Langage humain :
"Si x est plus grand que 10,
alors décrémenter x..."

Langage haut niveau (C, PHP, ...):
"if(x > 10) x--;"

Langage assembleur :
"cmp x, 10 dec: dec x
jb dec "

Langage machine :
"0011101001110110110101011101
000011000110111011011001101001"

Bas niveau

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

Programmation en assembleur

Description des opérandes

- par registre:
 - opérande dans un registre LD R0,4
 - décrite par le registre la contenant
- immédiat: valeur contenue dans l'instruction
- direct: LD R0,4
 - opérande en mémoire LD R0,adop
 - décrite par son adresse en mémoire (adop)
- indirect: LD R0,var
 - opérande en mémoire
 - décrite par le registre ou la variable contenant son adresse (pointeur)
- indirect avec déplacement: LD R0,R2+2
 - opérande en mémoire
 - décrite par le registre ou la variable contenant une adresse et un déplacement à partir de cette adresse²⁷

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Description des opérandes

- indirect avec en post ou pré une incrémentation ou une décrémentation:
 - opérande en mémoire
 - décrite par le registre ou la variable contenant une adresse et une incrémentation ou décrémentation soit de l'adresse soit de la variable
- avec un segment \Rightarrow protection des données
- une combinaison de tout cela:
 - Segment + indirect + déplacement + post incrément
 - ...

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Instructions en langage machine

- Syntaxe:

[etiq i:] COP [OP1[,OP2]] ;commentaire

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Instructions en langage machine

- **Différents types d'instruction:**
 - **Déplacement d'information:**
 - De mémoire à mémoire
 - De mémoire à registre
 - De registre à registre
 - **Traitements:**
 - Arithmétiques
 - Logiques
 - Comparaisons
 - Décalages
 - **Traitements spécifiques à certains μp :**
 - Mathématiques
 - Vectoriels
 - Chaînes de caractères
 - Traitement d'images

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

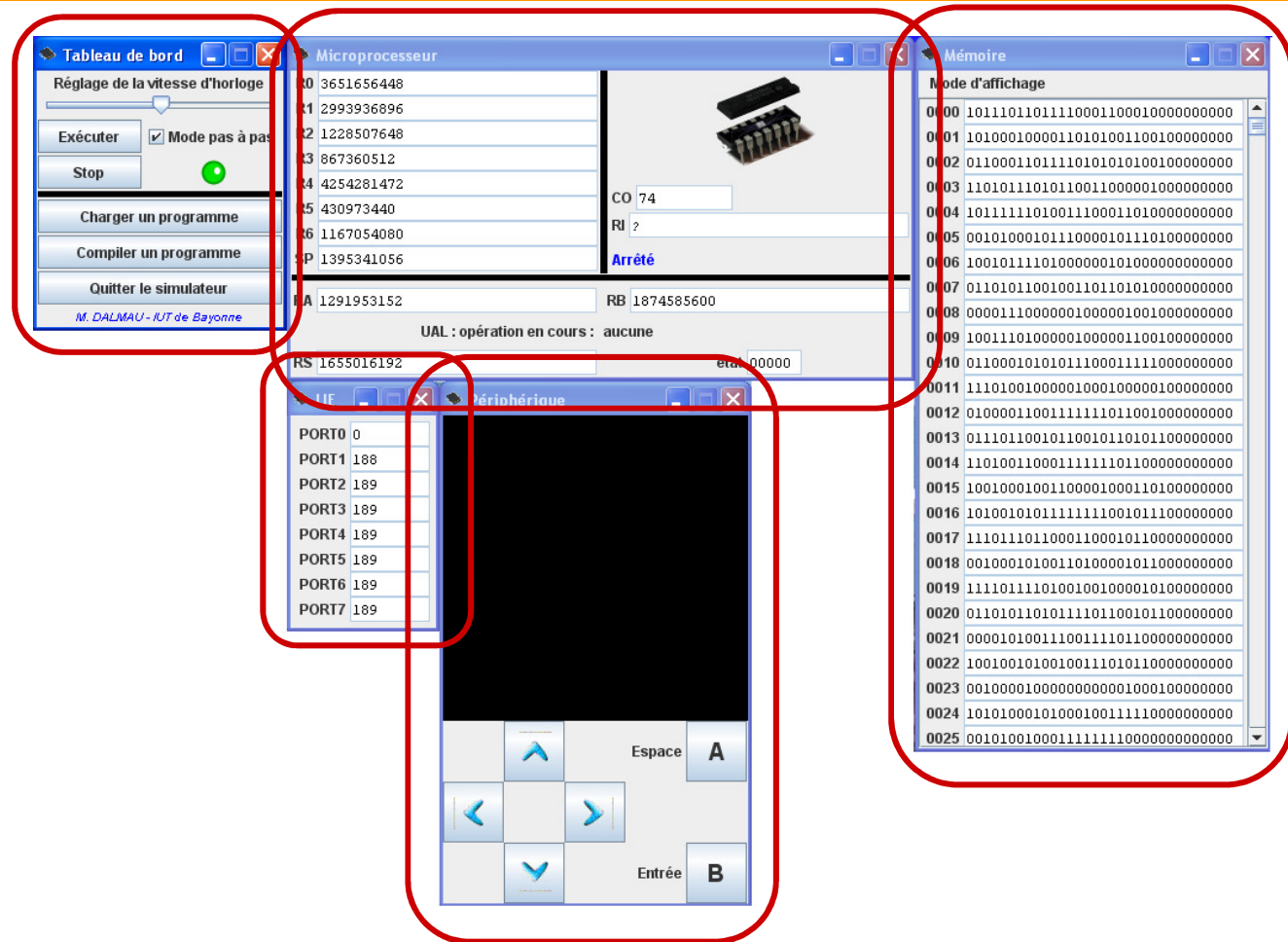
Instructions en langage machine

- **Différents types d'instruction:**
 - **Rupture de séquences:**
 - Passage d'une instruction à une autre selon une condition
 - Condition sur le registre d'état de l'Unité de Traitement (UAL)
 - Description de l'instruction destinataire: étiquette (adresse)
 - **Contrôle:**
 - Appel, retour de procédures
 - Manipulation de pile
 - Gestion des interruptions

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

Programmation en assembleur

Exemple de μ p : simulateur de Marc Dalmau



Voir guide de programmation en fin de polycopié

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

Programmation en assembleur

Exemple de μ p : simulateur

- **Mémoire: mots de 32 bits**
- **Registres:**
 - Accessibles par programmeur:

Microprocesseur	
R0	3651656448
R1	2993936896
R2	1228507648
R3	867360512
R4	4254281472
R5	430973440
R6	1167054080
SP	1395341056

- R0 à R6: 7 registres d'usage général
- SP: pointeur de pile (Stack Pointer)

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

Programmation en assembleur

Exemple de μp : simulateur

- Registres:
 - Non accessibles par programmeur:



- CO
- RI

RA 1291953152	RB 1874585600
UAL : opération en cours : aucune	
RS 1655016192	etat 00000

- Registre d'état
- RA, RB, RS: registres de l'UT (UAL)

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Exemple de μp : simulateur

- **UT:**
 - Opérations de type arithmétique, logique et décalage uniquement
 - Entiers naturels, entiers relatifs en complément à 2

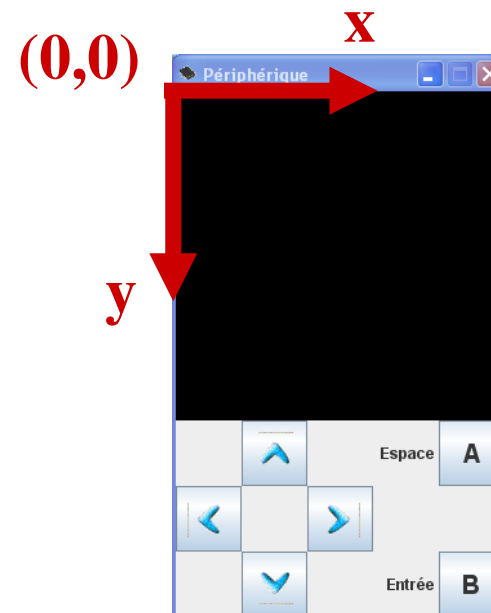
- Exemple: -7 en complément à 2
 - 7: 0111
 - Complément: 1000
 - Complément +1 \rightarrow complément à 2 soit -7: 1001

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

Programmation en assembleur

Exemple de μp : simulateur

- **UE:**
 - **Gestion:**
 - Clavier: 6 touches + 1 souris
 - Écran graphique 256x256 en couleur

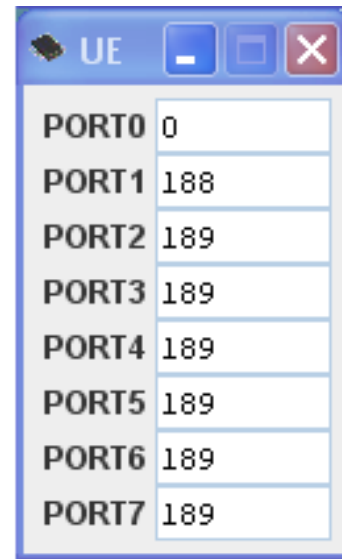


- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

Programmation en assembleur

Exemple de μp : simulateur

- **UE:**
 - **8 registres appelés ports:**
 - Port 0: touches du clavier et souris
 - Port 1 à 5: écran graphique
 - Port 6: coordonnée x de la souris
 - Port 7: coordonnée y de la souris



PORT0	0
PORT1	188
PORT2	189
PORT3	189
PORT4	189
PORT5	189
PORT6	189
PORT7	189

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Programme en langage machine

- **Programme en langage machine: définition**
 - variables : adresse en mémoire + taille
 - instructions : COP + opérandes
- **3 zones en mémoire:**
 - Code
 - Variables
 - Pile: utilisée pour les procédures
- **Organisations possibles des 3 zones mémoires:**
 - mélangées
 - par segments si le μ p les gère \Rightarrow protection

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Programme en langage machine

- Squelette d'un programme pour simulateur:

.DATA

déclaration des variables et constantes

.CODE

écriture du code

.STACK

réservation de place pour la pile

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Programme en langage machine

- **Déclaration des variables et constantes**
 - **Définition:**
 - Réservation de place en mémoire pour une variable
 - Attribution d'un nom
 - **Absence de typage:**
 - Connue seulement par programmeur
 - ***Au programmeur à effectuer les bons traitements***
 - **Manipulation des informations par leur nom (adresse)**
 - Possibilité de manipuler toute la mémoire: variables, instructions
 - ***Responsabilité du programmeur***
 - **Deux cas:**
 - Variables non initialisées
 - Variables initialisées

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Programme en langage machine

- **Déclaration des variables et constantes**
 - **Variables non initialisées:**

Nom	DSW	taille (en mots mémoire)
------------	------------	---------------------------------
 - **Variables initialisées:**

Nom	DW	valeur
------------	-----------	---------------

Valeur: Convertie en binaire par le compilateur
 - **une variable = un mot mémoire (sauf pour chaîne)**

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Programme en langage machine

- Types des valeurs:
 - Entier décimal positif ou négatif: 1225 ou -6
 - de 0 à 4 294 967 295
 - ou de – 2 147 483 647 à 2 147 483 647
 - Valeur sur 32 bits en complément à 2
 - Entier hexadécimal: **\$1a** ou **\$ffa3**
valeur sur 16 bits (4 digits hexadécimaux maximum)
Rappel sur le codage hexadécimal

0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Programme en langage machine

- **Types des valeurs:**
 - **Valeur binaire:** **%10111010**
valeur sur 16 bits (16 chiffres binaire maximum)
 - **Caractère ASCII: 'v'**
 - code ASCII du caractère étendu à 32 bits
 - ' et ; inutilisables car délimitent les chaînes de caractères et les commentaires
 - **Chaine de caractères :** **« egun on »**
un code ASCII étendu à 32 bits par mot

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Programme en langage machine

- Exemple de déclaration:

<i>var1</i>	DW	12	var 1 occupe 1 mot qui contient 12
<i>car</i>	DW	'!'	1 mot qui contient le code ASCII de '!'
<i>var2</i>	DW	-1	1 mot qui contient -1
<i>var3</i>	DSW	12	12 mots non initialisés
<i>var4</i>	DW	"ABC"	3 mots dont le 1er contient le code ASCII de 'A', le deuxième celui de 'B' et le dernier celui de 'C'.
<i>var5</i>	DW	11	2 mots initialisés à 11 pour le premier et 22 pour le second
	DW	22	
<i>var6</i>	DW	"je"	3 mots dont le 1er contient le code ASCII de 'j', le 2ème celui de 'e' et le dernier la valeur 0.
	DW	0	

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Programme en langage machine

- **Ecriture du code**
 - **Désignation des opérandes**
 - Immédiat : valeur **Val**
 - Registre : nom du registre **RG**
ex: R0 ou r1
 - Direct : nom de la variable **Var**
 - ex: var1
 - Nom commence par une lettre
 - délimiteurs (' " ; , []), mot clé STACK, nom de registres interdits
 - Casse (majuscules ≠ minuscules)
 - Accent possible.
 - Indirect : **[RG+d]**
 - **Registre contient l'adresse de l'opérande [R1]**
 - **déplacement positif: $0 \leq d \leq 1023 = 2^{10}-1$**
 - **adresse dans le registre: 10 bits de faible poids (sur 32)**

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Programme en langage machine

- **Attention:**
 - Dans les exemples de déclaration (Var1, var2,): variables sur 1, 2 ou x mots mémoires
 - Dans la description des instructions:
 - Var uniquement sur 1 mot mémoire
 - Var ≠ chaîne
 - Var ≠ tableau

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Programme en langage machine

- **Ecriture du code**
 - **Déplacement de données**
 - **LD** dest, source ; écriture dans un registre
 - chargement dest \leftarrow source
 - destination: RG ou [RG]
 - source: Val, Var, RG, [RG], [RG+d]
 - **ST** source, Var ; lecture d'un registre
 - chargement Var \leftarrow source
 - source: RG, [RG]
 - **SWP** oper
 - échange les 16 bits de fort poids et les 16 bits de faible poids
 - oper: Var, RG, [RG], [RG+d]
 - **LEA** dest, var
 - chargement dest \leftarrow adresse de la variable
 - dest: RG, [RG]
 - initialisation de pointeur (SP)
 - Adresse du premier élément de la variable

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Programme en langage machine

- Exemples d'écriture du code (Déplacement de données avec LD):

- LD R0,r1 ; R0 \leftarrow R1
- LD R1,\$ff03 ; R1 \leftarrow FF03 en hexadécimal
- LD R2,'A' ; R2 \leftarrow code ASCII de 'A'
- LD R0,var1 ; R0 \leftarrow contenu de var1
- ST R2,var2 ; var2 \leftarrow contenu de R2
- LD R0,[SP] ; R0 \leftarrow contenu de la mémoire à l'adresse contenue dans SP
- LD [R1],12 ; mémoire à l'adresse contenue dans R1 \leftarrow 12
- LD [r1],var ; mémoire à l'adresse contenue dans R1 \leftarrow contenu de var
- LD R0,[R0] ; R0 \leftarrow contenu de la mémoire à l'adresse contenue dans R0

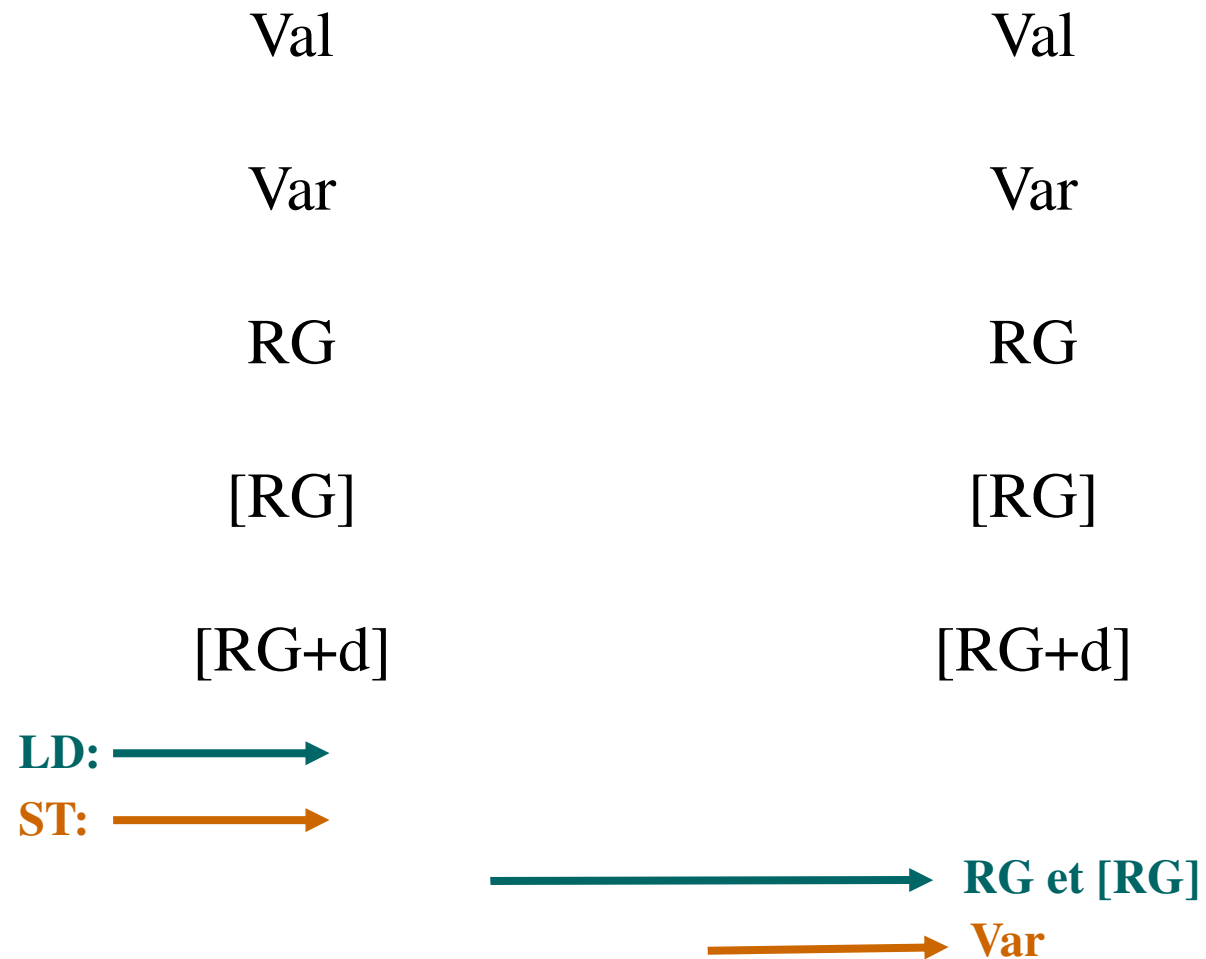
- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Programme en langage machine

- Différence entre ST et LD pour les déplacements:



- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Programme en langage machine

- **Ecriture du code**
 - **Instructions arithmétiques**
 - Indicateurs de débordements des entiers naturels et relatifs du registre d'état de l'UT (sauf NEG)
 - Type: **XXX op1, op2**
 - $op1 \leftarrow op1 \text{ XXX } op2$
 - **XXX**= ADD, SUB, MUL (**entiers relatifs**), MULU (**entiers naturels**), DIV, DIVU
 - **op1**: RG, [RG]
 - **op2**: Val, Var, RG, [RG], [RG+d]
 - Type: **YYY oper**
 - **oper** \leftarrow résultat de YYY sur oper
 - **YYY** = INC, DEC, NEG (**-oper**)
 - **oper**: Var, RG, [RG], [RG+d]

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Programme en langage machine

- **Ecriture du code**

- **Instructions logiques:**

- **OR op1, op2**
 - **AND op1, op2**
 - **NOT oper**

		OU OR	ET AND	NON NOT
x	y	x+y	x.y	\bar{x}
0	0	0	0	1
0	1	1	0	1
1	0	1	0	0
1	1	1	1	0

- **Types de données:**

- op1: **RG, [RG]**
 - op2: **Val, Var, RG, [RG], [RG+d]**
 - oper: **Var, RG, [RG], [RG+d]**

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Programme en langage machine

- **Ecriture du code**
 - **Comparaison**
 - **CMP op1, op2**
 - **Positionnement des indicateurs du registre d'état de l'UT: Instructions de branchement**
 - op1: **RG, [RG]**
 - op2: **Val, Var, RG, [RG], [RG+d]**

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Programme en langage machine

- **Ecriture du code**
 - **Instructions de rupture de séquence: branchements**
 - JMP etiq: branchement inconditionnel
 - Bxx etiq: branchement conditionnel si la condition est vérifiée

Condition	Comparaison d'entiers naturels	Comparaison d'entiers relatifs
op 1 = op2	BEQ	BEQ
op1 ≠ op2	BNE	BNE
op1 < op2	BLTU	BLT
op1 ≤ op2	BLEU	BLE
op1 > op2	BGTU	BGT
op1 ≥ op2	BGEU	BGE
débordement	BDEBU	BDEB

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Programme en langage machine

- **Ecriture du code**
 - **Instructions de rupture de séquence: branchements**
 - **Etiquette**
 - désigne l'instruction résultante
 - **mot alphanumérique**
 - » sans délimiteur du langage: ' " ; , []
 - » sans espace
 - » caractères accentués possibles
 - » terminé par :
 - **Ne pas la faire suivre d'un espace**

- **Exemple:**

Debut:	LD	R0,0
	CMP	R0,0
	BNE	debut
	INC	R0

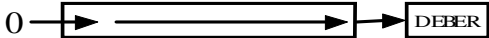
CO incrémenté donc si R0 = 0 passage à la suite

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Programme en langage machine

- **Ecriture du code**
 - **Instructions de décalage**
 - Oper: Var, RG, [RG], [RG+d]
 - DEBER: indicateur de débordement des entiers naturels du registre d'état de l'UT
 - logique:
 - **SHR oper (droite)** 

- **SHL oper (gauche)** 

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Programme en langage machine

- **Ecriture du code**
 - **Instructions de décalage**
 - Arithmétique

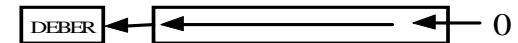
- **SAR oper (droite)**

conserve le signe du nombre (division par 2^n)



- **SAL oper (gauche)**

(identique SHL)



- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Programme en langage machine

- Exemple: décalage arithmétique droite

1	→	1	1	0	1	1
déb						

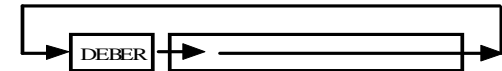
0	→	0	1	1	0	0
déb						

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

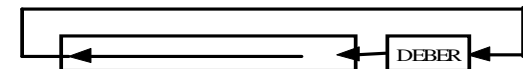
Programmation en assembleur

Programme en langage machine

- **Ecriture du code**
 - **Instructions de décalage**
 - Cyclique
 - ROR oper (droite)



- ROL oper (gauche)



- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Structures de contrôle en langage machine

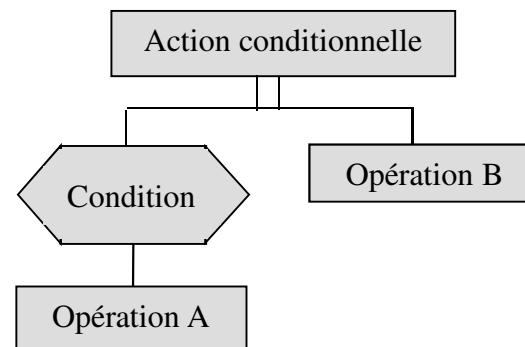
- PB: Traduction des structures de contrôle en langage machine
- Traduites avec les opérations de rupture de séquence conditionnelles ou inconditionnelles
- Compilateur: réalise cela
- Contraintes possibles imposées par les langages sur la machine :
 - vérification des types des variables
 - interdiction de la modification du code
 - limitation des structures de contrôle
- Machine seule: pas de contraintes, tout est possible

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

Programmation en assembleur

Structures de contrôle en langage machine

- **Action conditionnelle**



- **Évaluation de la condition**
- **Si faux branchement à instruction associée (B)**
- **Si vrai**
 - effectuer l'opération associée à condition vérifiée (A)
 - puis effectuer l'opération non conditionnelle (B)

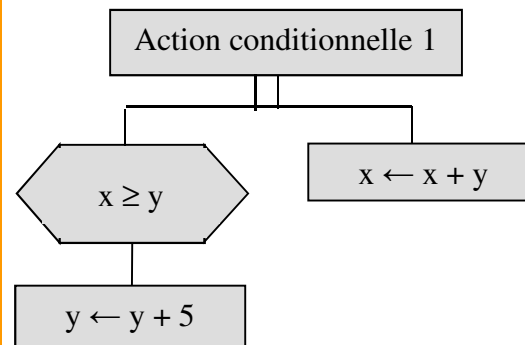
- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Structures de contrôle en langage machine

- Exemple d'action conditionnelle



1. Évaluation de la condition $x \geq y$
2. Si faux branchement à instruction associée $x \leftarrow x + y$
3. Si vrai:
 1. effectuer l'opération associée à condition vérifiée $y \leftarrow y + 5$
 2. Puis effectuer l'opération non conditionnelle $x \leftarrow x + y$

Si condition vraie (\geq)

Si condition fausse ($<$)

suite:



CMP
BLTU
ADD
ADD

x,y
suite
y,5
x,y



; si <
; si ≥

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Structures de contrôle en langage machine

- Remarque:

- limitations de notre simulateur: contraintes sur les types d'opérandes des instructions

CMP xy Variables \neq RG ou [RG]

BLTU suite ; si <

ADD y,5 ; si \geq

suite: ADD xy

- Nouvelle version du code

LD R0,x

LD R1,y

CMP R0,y

BLTU suite ; si <

ADD R1,5 ; si \geq

suite: ADD R0,R1

ST R0,x

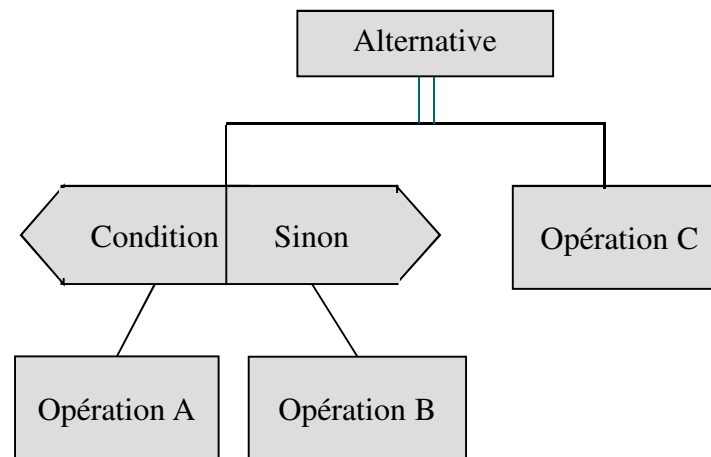
ST R1,y

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

Programmation en assembleur

Structures de contrôle en langage machine

- **Alternative:**



- Évaluation de la condition
- Si faux branchement à instruction associée (B)
- Effectuer l'opération associée à condition vérifiée (A) et branchement après opération pour faux
- Effectuer l'opération (C)

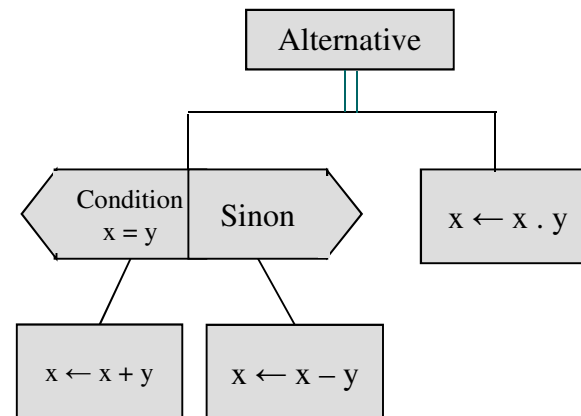
- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Structures de contrôle en langage machine

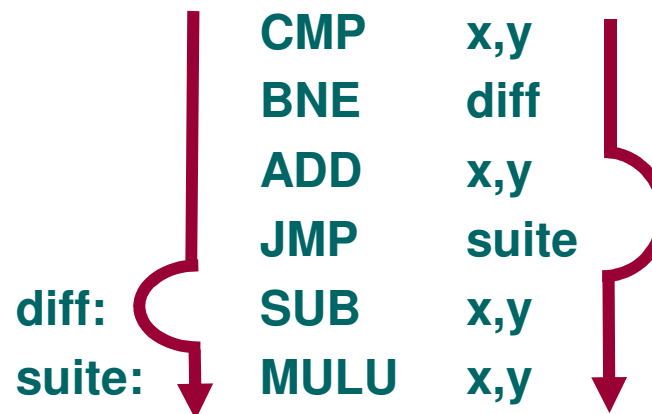
Exemple d'alternative:



1. Évaluation de la condition $x = y$
2. Si faux branchement à instruction associée $x \leftarrow x - y$
3. Si vrai:
 1. effectuer l'opération associée à condition vérifiée $x \leftarrow x + y$
 2. brancher après opération pour faux
4. Dans tous les cas, effectuer l'opération non conditionnelle $x \leftarrow x . y$

Si condition vraie (=)

Si condition fausse (≠)



; si≠
; si =

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Structures de contrôle en langage machine

- Remarque:

- limitations de notre simulateur: contraintes sur les types d'opérandes des instructions

	CMP	x y	Variables \neq RG ou [RG]
	BNE	diff	; si \neq
	ADD	x y	; si =
	JMP	suite	
diff:	SUB	x y	
suite:	MULU	x y	

- Nouvelle version du code

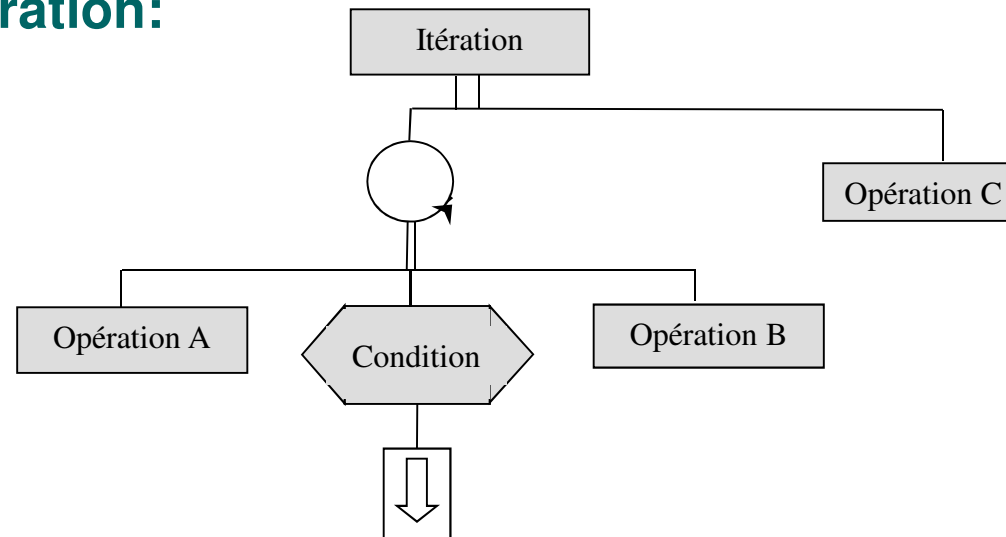
	LD	R0,x	
	CMP	R0,y	
	BNE	diff	; si \neq
	ADD	R0,y	; si =
	JMP	suite	
diff:	SUB	R0,y	
suite:	MULU	R0,y	
	ST	R0,x	

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

Programmation en assembleur

Structures de contrôle en langage machine

- **Itération:**



- Première opération de la boucle (A)
- Évaluation de la condition de sortie
- Branchement de sortie si condition de sortie vérifiée
- Autrement:
 - autres opérations (B)
 - Branchement à boucle
- Opération associée à la sortie de la boucle (C)

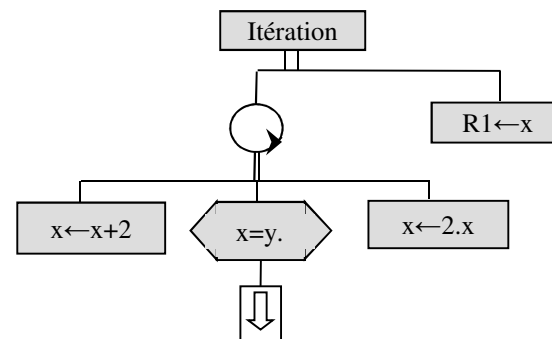
- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



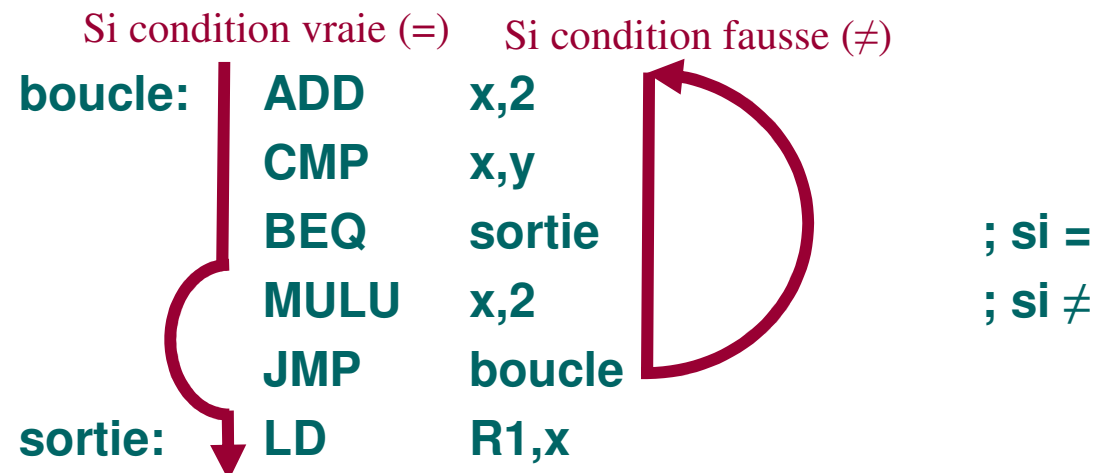
Programmation en assembleur

Structures de contrôle en langage machine

Exemple d'itération:



1. Première opération de la boucle $x \leftarrow x + 2$
2. Evaluation de la condition de sortie $x = y$
3. Branchement si condition de sortie vérifiée
4. Autrement:
 1. Autres opérations $x \leftarrow 2.x$
 2. Branchement à boucle
5. Opération associée à la sortie de la boucle $R1 \leftarrow x$



- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Structures de contrôle en langage machine

•Remarque:

–limitations de notre simulateur: contraintes sur les types d'opérandes des instructions

```

boucle:  ADD    x,2    Variables ≠ RG ou [RG]
          CMP    x,y
          BEQ    sortie ; si =
          MULU   x,2    ; si ≠
          JMP    boucle
    
```

sortie: LD R1,x
– Nouvelle version du code

```

          LD      R0,x
boucle:  ADD      R0,2
          CMP     R0,y
          BEQ     sortie ; si =
          MULU   R0,2    ; si ≠
          JMP     boucle
sortie:  LD       R1, R0
          ST      R0,x
    
```

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Manipulation d'adresse et pointeurs

- **Pb: Manipulation d'adresse et pointeurs**
- **Accès aux variables:**
 - **Directement: nom \Leftrightarrow adresse**
 - Exemple 1:

```
LD R0,25 ; R0 $\leftarrow$ 25
ST R0,total ; total $\leftarrow$ R0
```

}

donc 25 dans total

Accès direct et par registre
 - **Indirectement:**
 - adresse \Leftrightarrow pointeur
 - [Reg]: Reg = registre contenant l'adresse de la variable
 - Exemple 2:

```
LEA R1,total ; R1 $\leftarrow$ @ de total
LD [R1],25 ; variable dont l'@ est dans R1 $\leftarrow$ 25
```

}

donc 25 dans total comme exemple 1

Mais accès indirect

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Manipulation d'adresse et pointeurs

- Différence entre les deux modes d'accès:

Microprocesseur

R0	105
R1	228
R2	

Mémoire

@227	4
@228	52
@229	100

Microprocesseur

R0	105
R1	23
R2	229

Mémoire

	Var1
	Var2
@229	3

Programmation en assembleur

Manipulation d'adresse et pointeurs

- Exemple 3: **copie de var 1 dans var2**
 - Première version:

`LD R0, var1`
`ST R0, var2`

`; R0 ← var1`

`; var2 ← R0`

}

accès direct et par registre
 - Seconde version:

`LEA R1, var1`
`ST [R1], var2`

`; R1 ← @ de var1`

`; var2 ← variable dont l'@ est dans R1`

}

accès indirect
 - Troisième version

`LEA R1, var1`
`LEA R2, var2`
`LD [R2], [R1]`

`; R1 ← @ de var1`

`; R2 ← @ de var2`

`; var dt @ ds R2 ← var dt @ ds R1`

}

accès indirect

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Manipulation d'adresse et pointeurs

- Intérêt des pointeurs: accès à des variables composées:
 - enregistrements
 - tableaux
- Exemples:
 - Coordonnées d'un point (2 entiers)

Déclaration:

Coord DSW 2 ; taille=2 mots mémoire (2x32 bits)

Initialisation à (10,40):

LEA R0,coord ; adresse de coord dans R0

LD [R0],10 ; variable dont @ dans R0← 10

LD [R0+1],40 ; variable dont @-1 dans R0← 40

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Manipulation d'adresse et pointeurs

- Attention à la notation:

[R0] variable dont l'adresse est contenue dans R0

R0

125

124

\$0a1b

125

\$3d5f

126

\$0125

Mémoire

[R0+1] variable dont l'adresse est celle contenue dans R0 à laquelle on rajoute 1

R0

125

124

\$0a1b

125

\$3d5f

126

\$0125

Mémoire

NB: il aurait été plus logique de noter [R0]+1 mais le code aurait été illisible 73

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Manipulation d'adresse et pointeurs

- **Systèmes d'exploitation**
 - **Pointeurs utilisés par systèmes d'exploitation:**
 - Processus
 - Fichiers
 - Mémoire
 - **utilisation de structures de données très complexes**
 - Ex: tableau de pointeurs vers d'autres tableaux

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

Manipulation d'adresse et pointeurs

- **Tableaux utilisés par les systèmes d'exploitations:**
 - **Élément:**
 - variables de type enregistrement
 - certains champs de l'enregistrement: pointeurs
 - **Exemple d'élément de tableau pour un processus:**
 - n° du processus: PID
 - n° de l'utilisateur: UID
 - adresse de l'instruction à exécuter
 - tableau de n éléments (sauvegarde registres)
 - **Accès à cet élément par un pointeur:**
 - n+4 pour accéder à l'élément suivant
 - accès par déplacement : +1 UID, +2 adresse

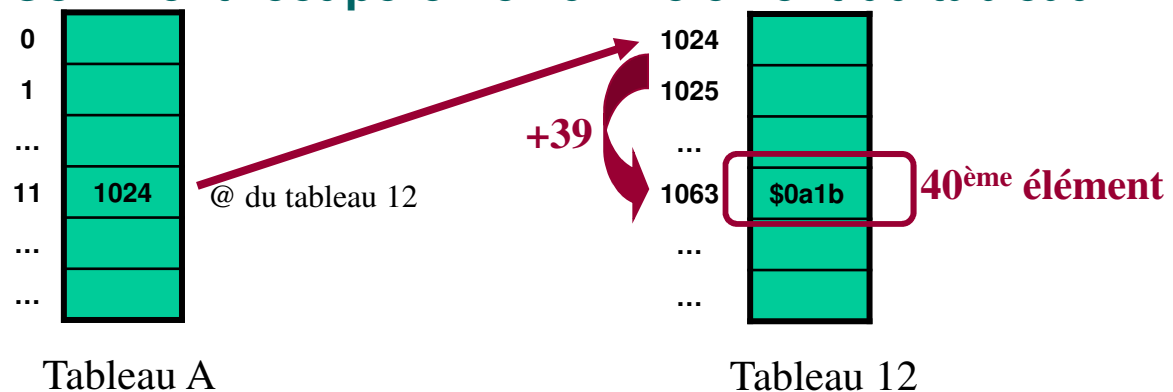
- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

Programmation en assembleur

Manipulation d'adresse et pointeurs

- Exemple d'utilisation de tableaux:

Comment récupérer le 40^{ème} élément du tableau n°12?



R0 0 R1 1024 R2 \$0a1b

```
LEA   R0,TA           ; R0 ← adresse de TA le tableau A

LD     R1, [R0+11]      ; R1 ← variable dont @ dans R0+11
                        ; R1 contient l'adresse du tableau 12

LD     R2,[R1+39]       ; R2 ← variable dont @-39 dans R1
```

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation en assembleur

A partir d'un algorithme

1. Identifier les instructions nécessaires à la réalisation de l'algorithme
2. Identifier les contraintes syntaxiques liées à ces instructions (types d'opérandes)
3. Rajouter les lignes de code permettant de respecter ces contraintes (transfert de données de ou vers des registres)

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Exercices d'application

EXERCICE 1 : Calcul de moyenne

- **Objectif:**
 - écrire un programme en assembleur qui calcule la moyenne des nombres entrés dans un tableau.
 - tableau = suite de mots mémoires qui se terminent par un mot contenant la valeur 0.
- 1. **partie du code :**
 - déclarer la variable moyenne appelée moy
 - initialiser à 0

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Exercices d'application

EXERCICE 1 : Calcul de moyenne

2. partie du code:

- déclarer la variable tableau T
- initialiser à 3-6-7-4-2 (0 marquant la fin du tableau)

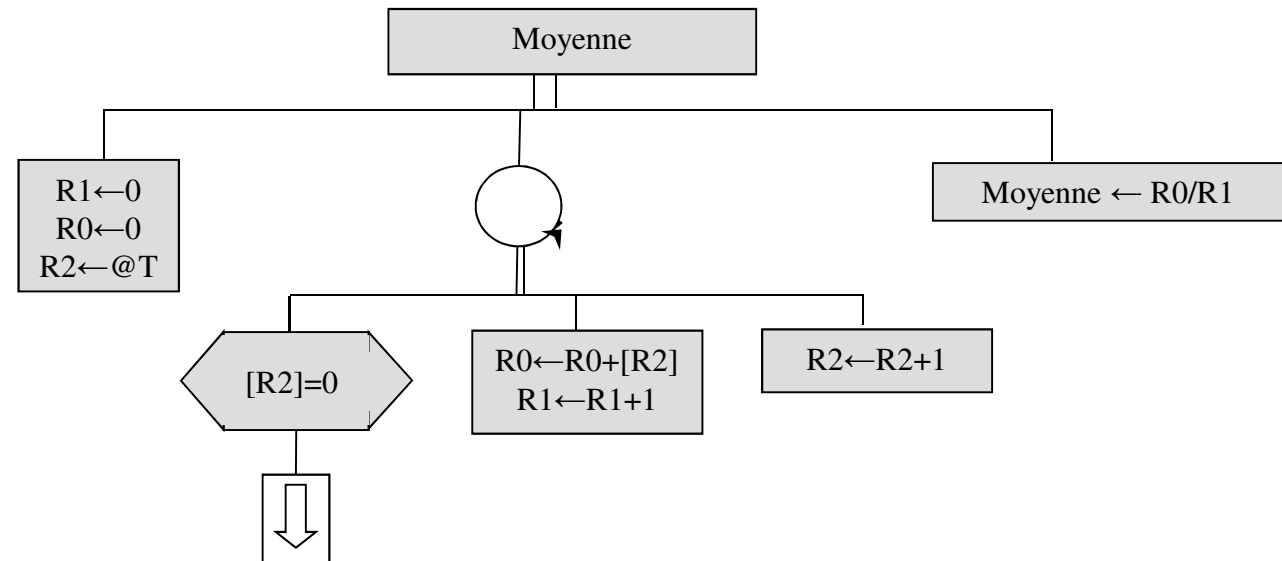
Exercices d'application

EXERCICE 1 : Calcul de moyenne

3. partie du code: calcul de la moyenne (pas de procédure)

- R0: valeur courante de la somme des nombres
- R1: nombre courant de chiffres pris en compte
- R2: pointeur sur le tableau

Algorithme



- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Exercices d'application

EXERCICE 1 : Calcul de moyenne

3. calcul de la moyenne

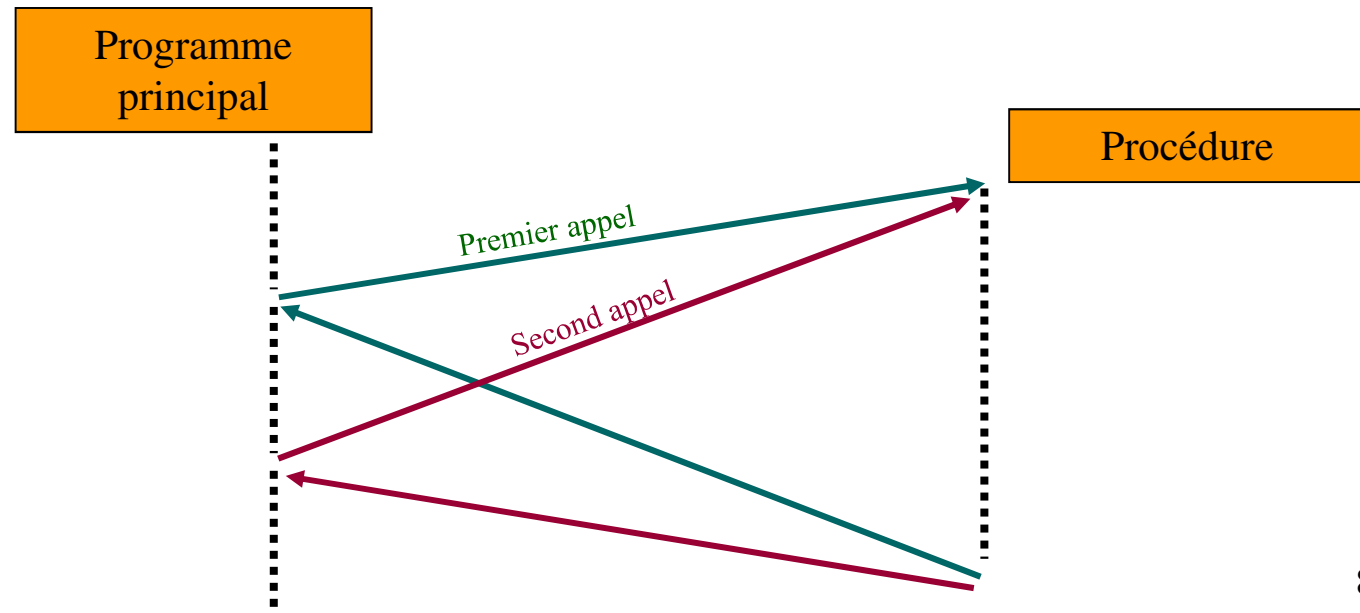
- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Procédures et fonctions

Définitions

- Principe:
 - morceau de code exécutable à la demande
 - transmission de paramètres
 - restitution d'une valeur si besoin
- Intérêt: réutilisation d'un morceau de code à différents endroits du programme

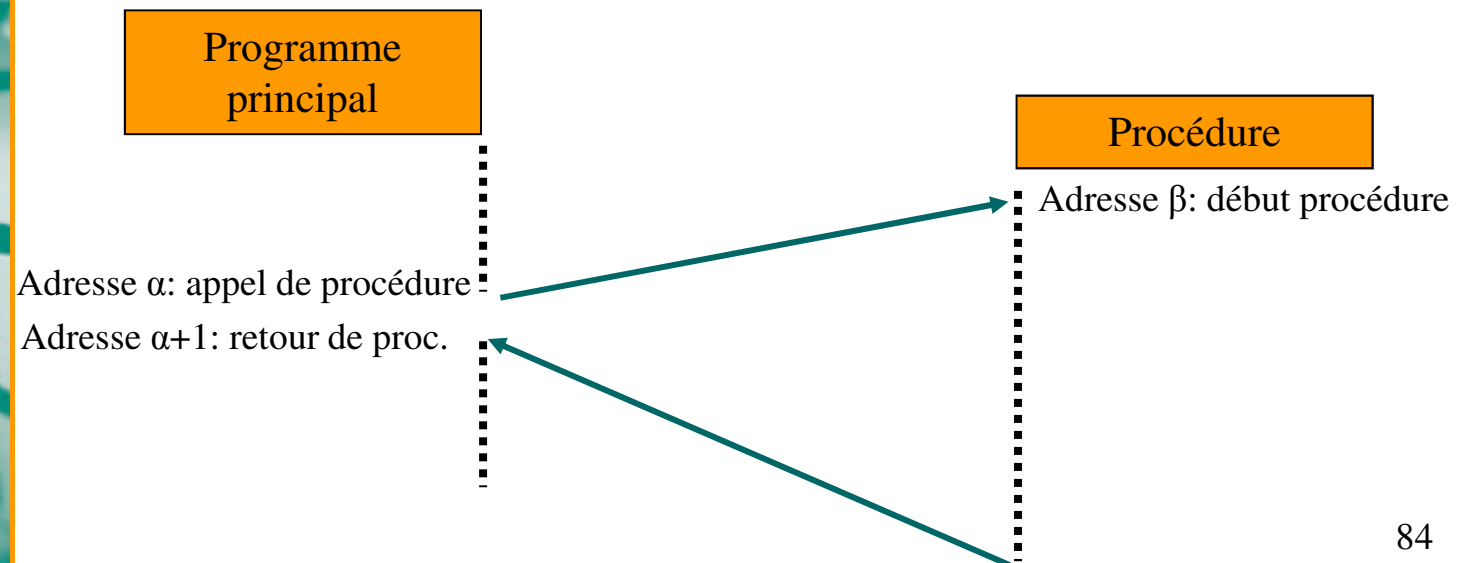


Procédures et fonctions

Gestion du contexte

- **Appel de procédure:**
 - quitter programme en cours
 - exécuter procédure
 - revenir au programme là où on l'a laissé

⇒ **nécessité de conserver l'adresse de retour**
- **Exemple:**



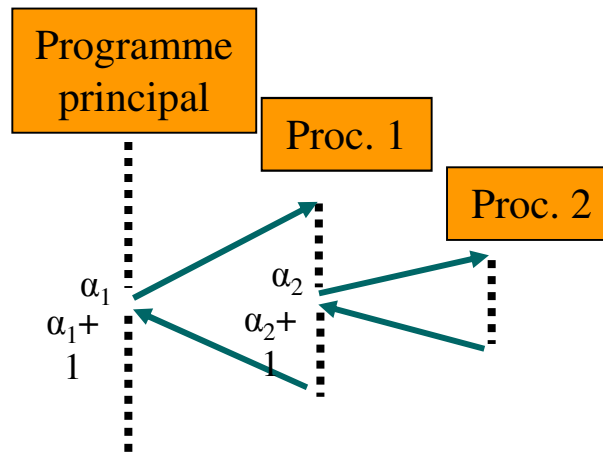
- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Procédures et fonctions

Gestion du contexte

- Solutions:
 - **Registre spécial contenant l'adresse de retour**
⇒ pb: impossible d'appeler la procédure dans la procédure
 - **Mémorisation de l'adresse de retour dans une structure de données adéquate: *laquelle?***



Structure de donnée de mémorisation des adresses:

Entrée:

α_1

α_2

Sortie:

α_2

α_1

- **Retour à l'adresse la plus récemment mémorisée**
Last in, First out **LIFO** **PILE**

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Procédures et fonctions

Gestion du contexte

- **Gestion de pile:**
 - **Registre spécial SP:**
 - SP: Sommet de pile
 - 1^{ère} entrée occupée
 - **Réservation de place mémoire pour la pile:**
 - directive: **.STACK** taille
 - **fin de code**
 - **Initialisation du registre SP sur le début de la pile:**
 - instruction: **LEA SP,STACK**
 - en début de code
 - **STACK** en majuscule

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Procédures et fonctions

Programme

- Squelette du programme:

.DATA

déclaration des variables et constantes

.CODE ; CO initialisé sur l'instruction qui suit .CODE

LEA SP,STACK

écriture du code

.STACK taille

réservation de place pour la pile

Rq: *possibilité d'utiliser une instruction HLT pour arrêter le processeur en fin de programme*

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Procédures et fonctions

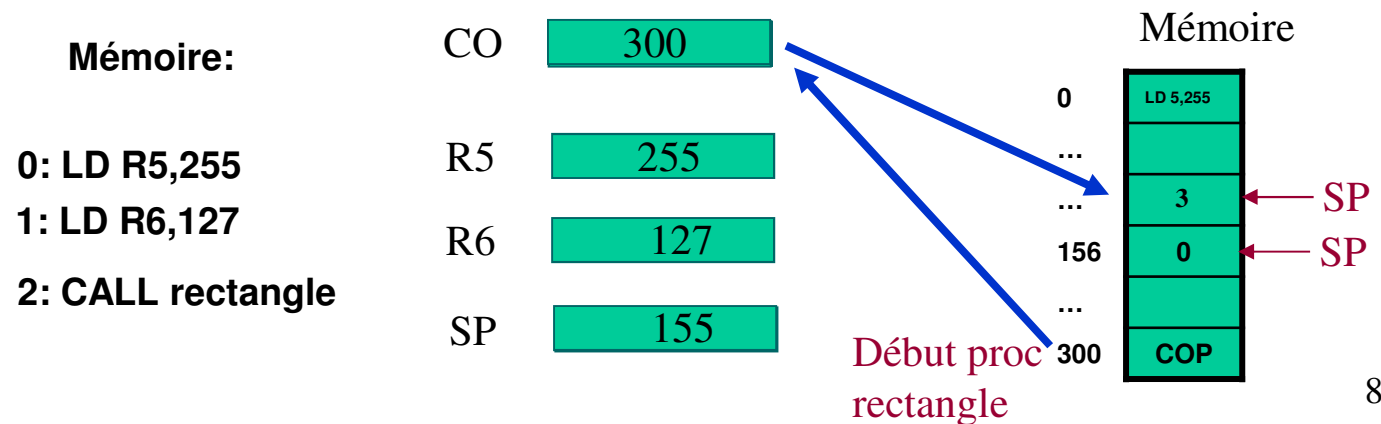
Programme

- Utilisation de deux registres pour désigner le début et la fin de la pile par certains processeurs:
 - Possibilité de détecter les débordements
 - En veillant à ne pas:
 - Dépiler une pile déjà vide
 - Empiler dans une pile déjà pleine
 - Possibilité d'arrêter les programmes en cours comme dans certains systèmes d'exploitation
- Cas de notre simulateur: bonne gestion de la pile de la responsabilité du programmeur

Procédures et fonctions

Instructions dédiées

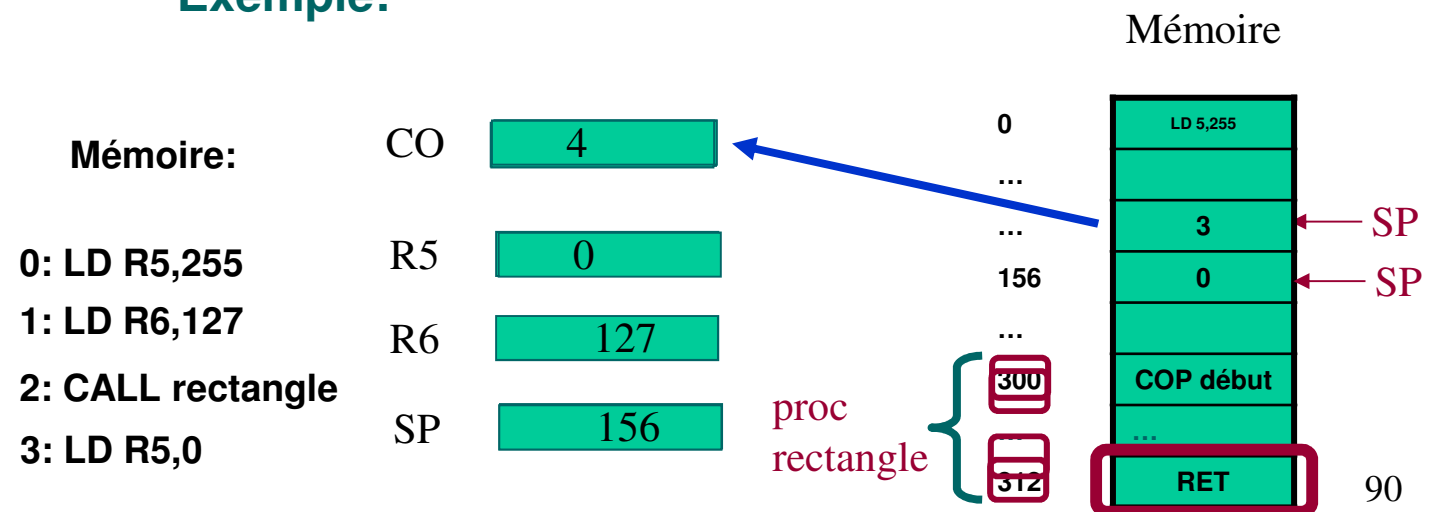
- Instruction de type rupture de séquence pour les procédures:
 - Appel: **CALL** étiquette
 - empile l'adresse de retour
 - $SP \leftarrow SP-1$
 - $[SP] \leftarrow CO$
 - Branche à la procédure \approx JMP étiquette
 - Exemple:



Procédures et fonctions

Instructions dédiées

- **Instruction de type rupture de séquence pour les procédures:**
 - **Retour:**
 - **RET**
 - dépile l'adresse de retour
 - **CO** \leftarrow **[SP]**
 - **SP** \leftarrow **SP+1**
 - **Exemple:**



- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Procédures et fonctions

Méthode de programmation

- Inconvénients liés aux procédures:
 - double accès aux registres de la machine et aux registres de variables globales:
 - depuis le programme
 - depuis les procédures
 - danger: modification par la procédure d'un registre utilisé par le programme qui l'appelle
⇒ perte de données

- Exemple:

	LD	R0,20	; R0 ← 20 nombre d'itération
Boucle:	CALL	calcul	; appel de la procédure calcul qui utilise R0 (R0← 356)
	DEC	R0	; R0 ← R0-1=355
	CMP	R0,0	; comparaison de 355 au lieu de 20 avec 0
	BNE	boucle	; si R0 ≠ 0 on boucle

Conclusion: perte du nombre de répétition de la boucle (infini) 91

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Procédures et fonctions

Méthode de programmation

- **solution 1:**
 - utiliser des registres différents pour les procédures et pour le programme principal
 - **problématique:**
 - nombre limité de registres
 - spécialisation de certains registres pour instructions
 - difficulté alors de réutilisation: appel de procédures écrites par d'autres (primitives du système d'exploitation)

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Procédures et fonctions

Méthode de programmation

- **solution 2:**
 - Sauvegarder les registres en début de procédure et les restituer à la fin
 - **Hypothèse : sauvegarde dans des variables**
 - Trop de variables
 - Impossible de s'assurer que chaque procédure utilise des variables différentes
 - Interdit la récursivité

⇒ Impossible

- **Conclusion:**
utilisation de la pile pour sauvegarder les registres
 - début de procédure: empilage des registres
 - fin de procédure: dépilage des registres

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Procédures et fonctions

Méthode de programmation

- Instructions:
 - Empiler:
 - **PUSH oper** ; empiler oper
 - Réalise:
 - $SP \leftarrow SP - 1$
 - $[SP] \leftarrow oper$
 - Exemple

Procédure en mémoire:

0: PUSH R5

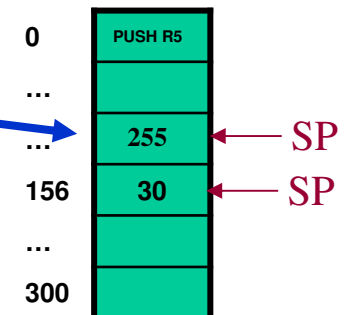
1: LD R6,127

CO 2

R5 255

SP 155

Mémoire



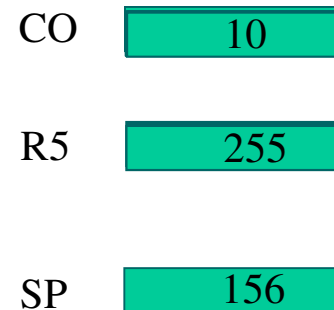
Procédures et fonctions

Méthode de programmation

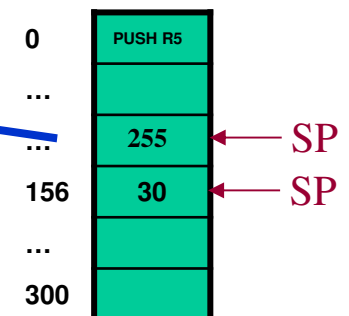
- **Instructions:**
 - **Dépiler:**
 - **PULL oper** ; dépiler oper
 - Réalise:
 - **oper** \leftarrow **[SP]**
 - **SP** \leftarrow **SP+1**
 - oper: **Var, RG, [RG], [RG+d]**

Procédure en mémoire:

0: PUSH R5
1: LD R6,127
...
7: ADD R5,2
8: PULL R5
9:



Mémoire



- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Procédures et fonctions

Méthode de programmation

- **Nécessité de prévoir une taille de pile suffisante:**
 - Stocker les adresses de retour
 - Stocker les sauvegardes de registres
- **Pour certains microprocesseurs:**
 - existence d'instructions pour empiler et dépiler tous les registres
 - en pratique inutile: aucun procédure n'utilise tous les registres
- **Variable locale des langages de programmation:**
 - Utilisation de la pile
 - Mécanisme:
 - début de procédure:
réservation d'espace mémoire dans la pile pour ces variables
 - fin de procédure:
libération de cet espace

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Procédures et fonctions

Passage de paramètres

- Passage de paramètres à une procédure dans les langages habituels:
 - par **valeur** :
 - en entrée
 - pas de possibilité de modification
 - par **référence**:
 - en entrée et sortie
 - possibilité de modification
- Ici étude du passage de paramètres en langage machine

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Procédures et fonctions

Passage de paramètres

- **Passage de paramètres par valeur :**
 - avant l'appel de la procédure: valeur placée dans un *registre*
 - utilisation du registre par la procédure: modification possible du registre mais valeur du paramètre inchangée
 - exemple

```
LD      R0,param      ; R0 ← param
CALL    calcul         ; paramètre d'entrée dans R0
```
 - avant l'appel de la procédure: valeur placée dans la *pile*
 - utilisation de la pile par la procédure: modification possible de la pile mais valeur du paramètre inchangée
 - Exemple

```
PUSH    param         ; SP ← SP-1 et [SP] ← param
CALL    calcul         ; paramètre d'entrée dans pile
```

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Procédures et fonctions

Passage de paramètres

- **Passage de paramètres par référence:**
 - avant l'appel de la procédure: adresse du paramètre placée dans un **registre**
 - procédure: accès au paramètre par indirection
 - exemple:

LEA	R0, param
CALL	calcul ; adresse du paramètre dans R0
 - avant l'appel de la procédure: adresse placée dans la **pile**
 - utilisation de la pile par la procédure
 - exemple:

LEA	R0, param
PUSH	R0
CALL	calcul ; adresse du paramètre dans pile via R0

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Procédures et fonctions

Passage de paramètres

- **Passage de paramètres à une procédure en langage machine:**
 - **par registre:**
 - plus rapide
 - limité en nombre et en taille des registres
 - **par pile:**
 - pas de limite en nombre et en taille
 - utilisé par les compilateurs

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Procédures et fonctions

Passage de paramètres

- Retour de la valeur d'une fonction: retour d'une unique valeur
 - ⇒ différent du cas des procédures
 - utilisation d'un registre: plus simple
 - (pile: plus compliqué, pas rentable pour une seule valeur)
- Utilisation de l'instruction: **RET ou RET n**
 - retour avec vidage de la pile de n mots
 - RET:
 - $CO \leftarrow [SP]$
 - $SP \leftarrow SP+1$
 - RET n:
 - $CO \leftarrow [SP]$
 - $SP \leftarrow SP+n+1$

Procédures et fonctions

Passage de paramètres

- Exemple pour RET:

- $CO \leftarrow [SP]$
- $SP \leftarrow SP+1$

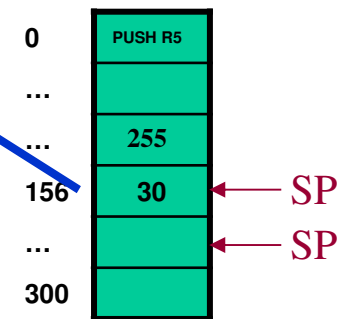
Procédure en mémoire:

```

0: PUSH R5
1: LD R6,127
  ⋮
7: ADD R5,2
8: PULL R5
9: RET
    
```

CO	30
R5	255
SP	157

Mémoire



- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Procédures et fonctions

Ecriture du programme

Quelle est l'écriture finale d'une procédure/fonction?

- Exemple de la fonction calcul (a,b): $a \leftarrow a + 2.b$
- Etude pour:
 - Passage des paramètres par valeur
 - Par registre
 - Par la pile
 - Passage des paramètres par références
 - Par registre
 - Par la pile
- Objectif: définir la structure d'écriture d'une procédure/fonction

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Procédures et fonctions

Ecriture du programme

- **Passage des paramètres par valeur**

- **Par registre:**

- Paramètres:

- **a dans R0**
 - **b dans R1**
 - **Valeur retournée par R0**

- Programme:

```
LD    R0, a    ; 1er paramètre
LD    R1, b    ; 2ème paramètre
CALL  calcul   ; appel de la fonction
```

- Procédure:

```
Calcul:  ADD    R0,R1
          ADD    R0,R1
          RET                      ; retour de sous-prog
```


Procédures et fonctions

Ecriture du programme

- **Passage des paramètres par valeur**

- **Par la pile:**

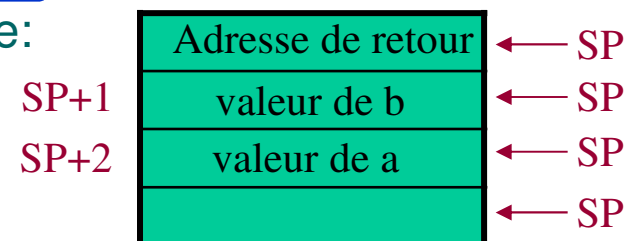
- Programme:

PUSH a	; 1 ^{er} paramètre
PUSH b	; 2 ^{ème} paramètre
CALL calcul	; appel de la fonction
ST R0,a	

- Procédure:

Calcul:	LD R0,[SP+2]	; R0 ← a
	ADD R0,[SP+1]	; ajout de b
	ADD R0,[SP+1]	
	RET	

- Etat de la pile:





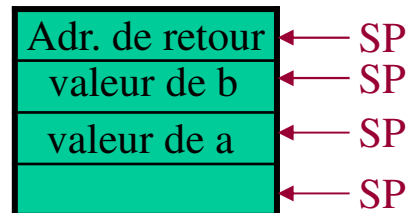
Procédures et fonctions

Ecriture du programme

- **Passage des paramètres par valeur**
 - **Par la pile:**
 - Inconvénient:
 - Pile non vide en fin de procédure (2 paramètres)
 - ⇒ Remplissage progressif de la pile
 - ⇒ Danger de débordement
 - Solution:
 - Programme appelant fait **ADD SP,2** après l'appel de la procédure (vidage de la pile)
 - Utilisation par certains processeurs de **RET n**
 - Nouveau code de la procédure

Calcul: **LD** **R0,[SP+2]** ; R0 ← a
 ADD **R0,[SP+1]** ; ajout de b
 ADD **R0,[SP+1]**

RET 2



Pile vide au moment du retour au programme

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Procédures et fonctions

Ecriture du programme

- **Passage des paramètres par référence**
 - **Par registre:**
 - Paramètres:
 - Adresse de a dans R0
 - b dans R1
 - Résultat dans a (donc adresse de a en donnée)
 - Programme:

```
LEA    R0, a    ; adresse de a dans R0
LD     R1, b    ; 2ème paramètre
CALL   calcul   ; appel de la fonction
```
 - Procédure:

```
Calcul: ADD    [R0],R1    ; a ← a + b
        ADD    [R0],R1
        RET                ; retour de sous-prog
```

Procédures et fonctions

Ecriture du programme

- **Passage des paramètres par référence**

- **Par la pile:**

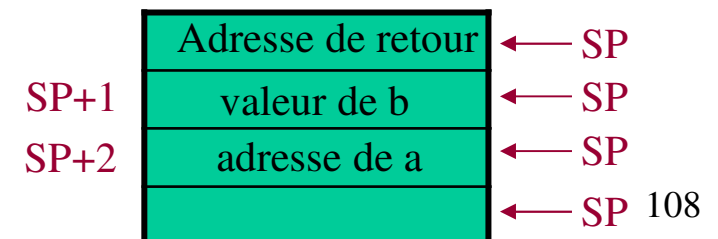
- Programme:

LEA R0,a	; R0 ← adresse de a
PUSH R0	; 1 ^{er} paramètre
PUSH b	; 2 ^{ème} paramètre
CALL calcul	; appel de la fonction

- Procédure:

Calcul:	LD R0,[SP+2]	; R0 ← adresse de a
	ADD [R0],[SP+1]	; ajout de b
	ADD [R0],[SP+1]	
	RET 2	

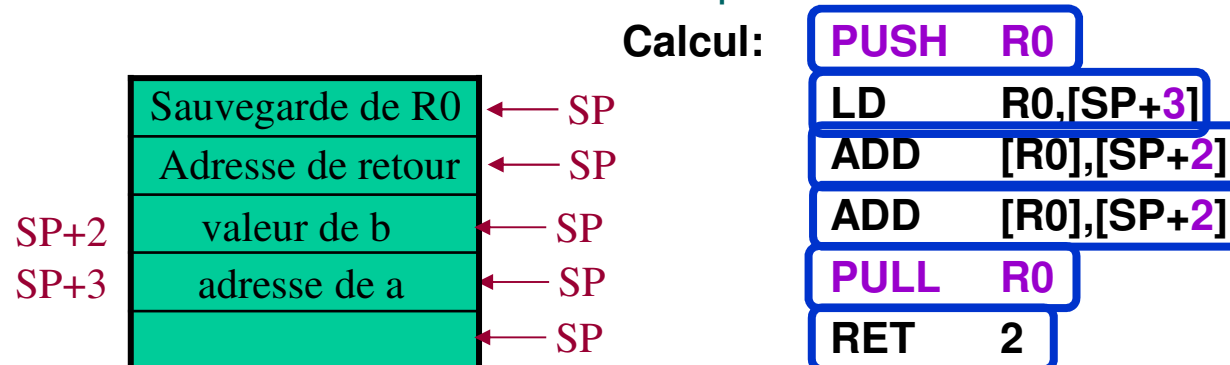
- Etat de la pile:



Procédures et fonctions

Ecriture du programme

- **Passage des paramètres par référence**
 - **Par la pile:**
 - Plus de retour de valeur par la procédure: modification directe de a
(version précédente: retour de résultat dans R0)
 - ⇒ Aucune raison de modifier R0 car ne sert pas à renvoyer le résultat
 - ⇒ Nécessité de sauvegarder R0
 - Nouveau code de la procédure:



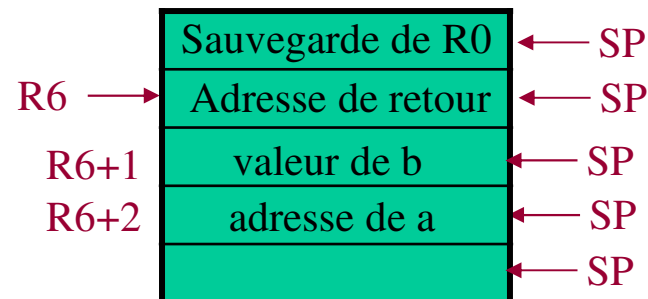
Procédures et fonctions

Ecriture du programme

- **Passage des paramètres par référence**
 - **Par la pile:**
 - **Problème:** décalage des références dans la pile à chaque sauvegarde d'un nouveau registre
 - **Solution:**
 - Accès à la pile par une copie de SP et non SP
 - Copie de SP faite en début de procédure avant la sauvegarde des registres
 - **Nouveau code de la procédure:**

Calcul:

```
LD    R6, SP
PUSH  R0
LD    R0,[R6+2]
ADD   [R0],[R6+1]
ADD   [R0],[R6+1]
PULL  R0
RET   2
```



- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Procédures et fonctions

Ecriture du programme

- **Passage des paramètres par référence**
 - **Par la pile:**
 - **Problème:** modification de R6 sans l'avoir sauvegardé avant
 - **Solution:**
 - Sauvegarde de R6 au début de la procédure
 - \Rightarrow décalage de 1 dans la pile \forall le nombre de registres sauvegardés dans la pile

Procédures et fonctions

Ecriture du programme

- Forme finale de l'écriture d'une fonction/procédure avec passage des paramètres par référence:

Calcul:

PUSH R6

LD R6, SP

PUSH R0

LD R0, [R6+3]

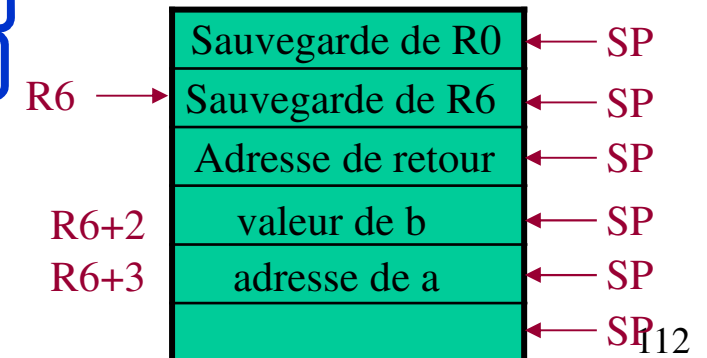
ADD [R0], [R6+2]

ADD [R0], [R6+2]

PULL R0

PULL R6

RET 2



- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Procédures et fonctions

Ecriture du programme

- En TD: Utilisation d'une version simplifiée du passage des paramètres par référence par la pile

Pas de copie du pointeur de pile

.DATA

a DW 5

;déclaration des variables et constantes

b DW 6

.CODE

;écriture du code

LEA SP,STACK

LEA R0,a

PUSH R0

PUSH b

CALL Calcul

;appel de la procédure

HLT

;code de la procédure

Calcul:

PUSH R0

LD R0,[SP+3]

ADD [R0],[SP+2]

ADD [R0],[SP+2]

PULL R0

RET 2

.STACK 3

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Exercice d'application

EXERCICE 2 : fonction XOR

- Objectif:
 - Calcul du résultat du XOR, OU EXCLUSIF, entre deux nombres binaires. $f = A.\overline{B} + \overline{A}.B$
 - Utilisation d'une procédure.
- 1. partie du code:
 - déclarer les variables A, B et f
 - les initialiser à 0.

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Exercice d'application

EXERCICE 2 : fonction XOR

2. code de la procédure lorsque le passage des valeurs des paramètres se fait grâce aux registres suivants :
 - R0 pour A
 - R1 pour B
 - R3 pour f

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Exercice d'application

EXERCICE 2 : fonction XOR

- 3. programme appelant cette procédure.

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Exercice d'application

EXERCICE 2 : fonction XOR

- 4. intégralité du code du programme lorsque le passage des paramètres se fait par la pile.

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation avancée d'un μp

Entrées Sorties

- **Unité d'échange (U.E.):** constituée de contrôleurs de périphériques
- **Contrôleur:**
 - pilote un ou plusieurs périphériques
 - deux visions:
 - ensemble de registres appelés **PORTS** qui sont accessibles par des instructions spéciales
 - mots en mémoire avec une adresse
- **UE:**
 - registres de contrôles: pilotage des périphériques
 - registres d'état: surveillance des périphériques
 - registres de données: communication avec les périphériques

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation avancée d'un μp

Entrées Sorties

- **UE du Simulateur:**
 - 8 registres de 8 bits correspondant aux n° de ports 0 à 7
 - **UE:**
 - connaître l'état des touches
 - dessiner ou écrire dans la fenêtre graphique du périphérique
 - détecter les mouvements et les clics de souris dans la zone graphique de l'écran du périphérique
 - **2 instructions:**
 - **IN oper, port** ; oper \leftarrow octet du registre de l'UE désigné par son n°
 - **OUT oper, port** ; octet du registre de l'UE désigné par son n° \leftarrow oper
 - Oper: RG ou [RG] sur 8 bits (plus faible poids)
 - Port: entier naturel désignant le n° de port de l'UE 119

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation avancée d'un μp

Entrées Sorties

- **Simulateur:**
 - **Clavier:**
 - Port 0 = registre d'état et de données

Etat du clavier		Numéro de la touche					
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

- **Etat du clavier:**
 - action qui a eu lieu sur le clavier depuis la dernière fois que le port 0 a été lu
 - Lecture du port 0 \Rightarrow remise à 0 (de B7 – B0)

00	aucune action
11	une touche appuyée
10	une touche relâchée

- B7: une action? B6: laquelle?

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

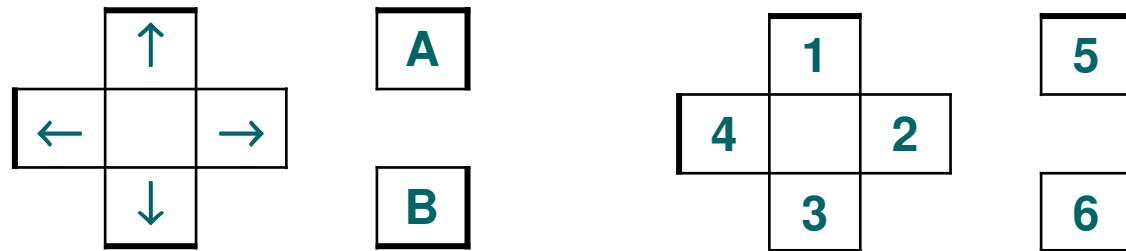
Programmation avancée d'un μp

Entrées Sorties

- **Simulateur:**

- **Clavier:**

- N° de la touche sur B5 – B0:



- **Souris:**

- Port 0 pour l'état:
 - **appui sur bouton souris = appui touche n°7**
 - **remise à 0 après lecture de son contenu**
 - Port 6 et 7 pour les données:
 - **en permanence les coordonnées de la souris**
 - **en x: port 6, en y: port 7**

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation avancée d'un μp

Entrées Sorties

- **Simulateur:**
 - **Ecran:**
 - Ports 1 à 5
 - **Ports 1 à 4:**
 - » registres de données
 - » paramètres de l'opération à exécuter
 - **Port 5:**
 - » registre de commande
 - » opération à exécuter

Couleur du tracé				Commande à exécuter			
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation avancée d'un μp

Entrées Sorties

- **Simulateur:**
 - **Ecran:**
 - **Commandes graphiques:**
 - **0000 : effacer l'écran**
 - **0001 : tracer le point**
Port1: x
Port2: y
Couleur: 4 bits *Couleur de tracé.*
 - **tracer une ligne (ports 1, 2, 3 et 4)**
 - **tracer un rectangle (ports 1, 2, 3 et 4)**
 - **tracer un ovale (ports 1, 2, 3 et 4)**
 - **tracer un rectangle plein (ports 1, 2, 3 et 4)**
 - **tracer un ovale plein (ports 1, 2, 3 et 4)**
 - **écrire un caractère ASCII (ports 1, 2, 3)**

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation avancée d'un μp

Entrées Sorties

- **Simulateur:**
 - **Méthode de tracé**
 1. Paramétrage du tracé sur les ports 1, 2, 3, 4 selon la définition des commandes graphiques
 2. Envoi de la commande de tracé sur le port 5 :
 - **Couleur sur les bits de fort poids**
 - **Figure sur les bits de faible poids**

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Programmation avancée d'un μp

Entrées Sorties

- **Simulateur:**

- **Exemple de tracé d'un rectangle bleu clair à l'écran:**

- Port 1: coordonnée en x du coin supérieur gauche
- Port 2: coordonnée en y du coin supérieur gauche
- Port 3: largeur
- Port 4 : hauteur
- Port 5: commande graphique 0011 et couleur 0101

```
LD      R0,200
OUT     R0,1  ;x
LD      R0,100
OUT     R0,2  ; y
LD      R0,20 ;
OUT     R0,3  ;largeur
LD      R0,80
OUT     R0,4  ;hauteur
LD      r0,$53 ; rectangle bleu soit 0101 0011
OUT     R0,5
```

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Exercices d'application

EXERCICE 3 : Test d'entrées-sorties

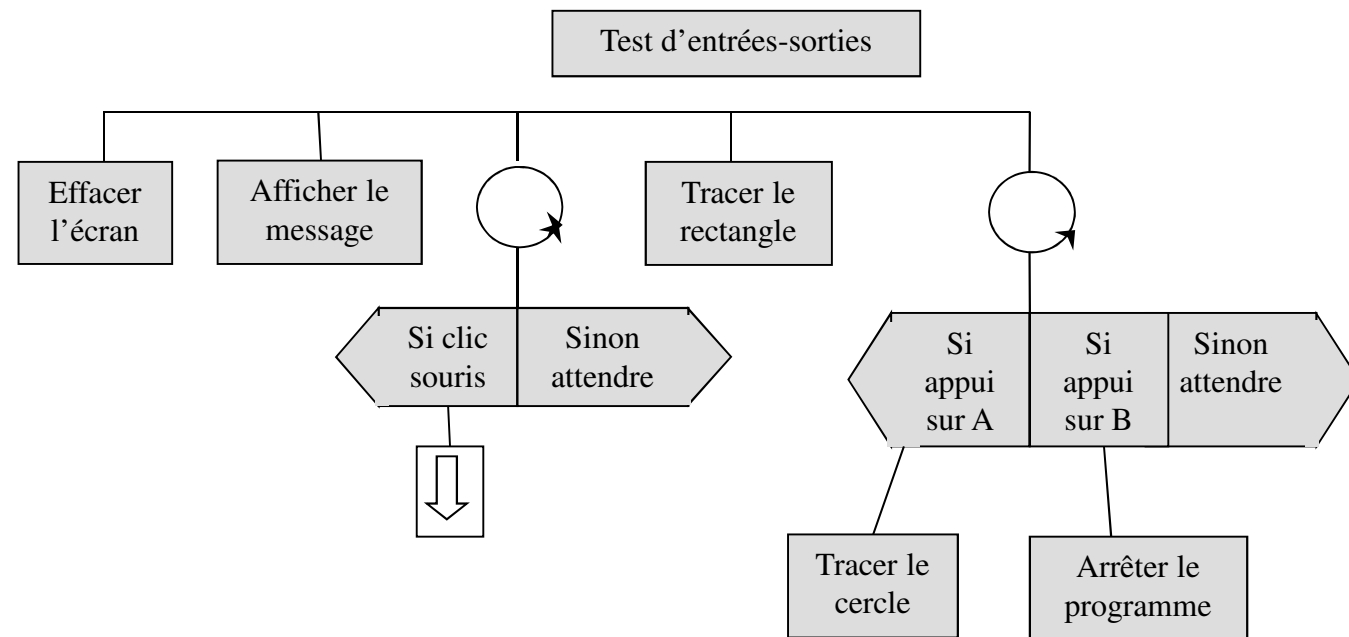
- **Objectif:**
 - programme en assembleur qui teste les entrées-sorties d'un microprocesseur.
 - **Déroulement du test:**
 - affiche un message à l'écran
 - attend un clic souris pour tracer un rectangle entre le point cliqué et les extrémités de l'écran (à droite et en bas)
 - attend ensuite un appui
 - soit sur la touche A pour tracer un petit cercle en haut à gauche du rectangle
 - soit sur la touche B pour arrêter le programme.

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

Exercices d'application

EXERCICE 3 : Test d'entrées-sorties

- **Algorithme:**



- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Exercices d'application

EXERCICE 3 : Test d'entrées-sorties

- message en début: « cliquer à l'endroit où vous voulez insérer le rectangle. »
- rectangle :
 - coin supérieur gauche: point où la souris a cliqué coordonnées (x ,y)
 - largeur=256-x
 - hauteur = 256-y.
 - plein, couleur cyan.
- cercle :
 - coin supérieur gauche du rectangle dans lequel il est inscrit: coin du rectangle précédent.
 - diamètre = 10.
 - plein, couleur noire.

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Exercices d'application

EXERCICE 3 : Test d'entrées-sorties

1. code permettant d'effacer l'écran
2. Déclaration et initialisation de la variable messa contenant le message à afficher.

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Exercices d'application

EXERCICE 3 : Test d'entrées-sorties

- 3. code permettant d'afficher le message.

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Exercices d'application

EXERCICE 3 : Test d'entrées-sorties

4. partie du code:

1. attendre un clic de souris

2. tracer le rectangle:

- procédure qui reçoit en paramètres les coordonnées de la souris par la pile.
- utiliser le registre R0

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Exercices d'application

EXERCICE 3 : Test d'entrées-sorties

4. 3. tracer le rectangle:

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Exercices d'application

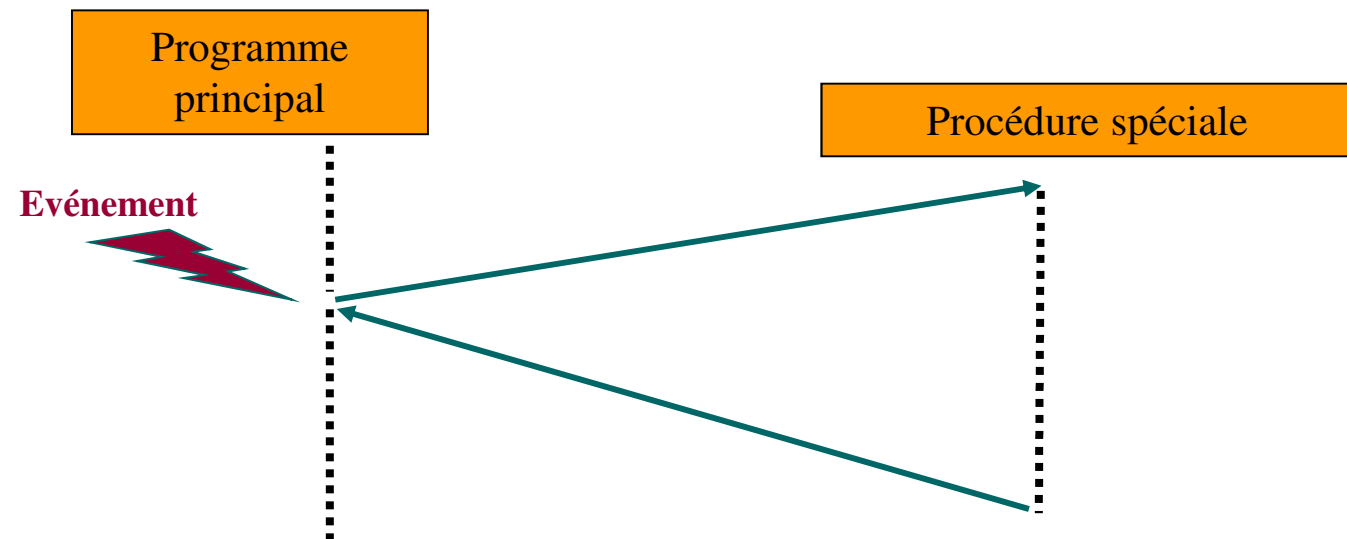
EXERCICE 3 : Test d'entrées-sorties

- 5. code restant pour terminer le programme.

Interruptions

Définitions

- **Interruptions (IT):**
 - utilisées par tous les microprocesseurs
 - absentes du simulateur
- **Principe:**
 - arrêt du programme provoqué par un événement
 - réalisation d'une procédure spéciale
 - reprise du programme



- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Interruptions

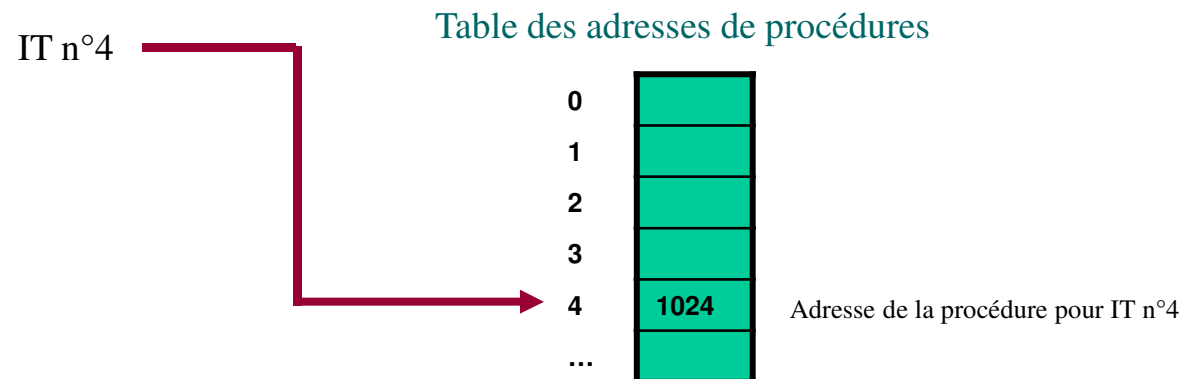
- **Utilisation:**
 - surveillance d'événements externes : génération d'événement par l'U.E. quand un registre d'état est modifié (ex: touche appuyée)
 - temps partagé
- **Deux types:**
 - logicielles: instruction (INT sur pentium) qui produit le même effet qu'une interruption physique
 - matérielles: interruption produite par une ligne physique

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Interruptions

- **Prise en compte d'une interruption:**
 - **Pb: quelle instruction exécuter?**
 - **Solutions**
 - S1: une seule procédure qui détermine l'origine de l'interruption et détermine l'instruction à exécuter
⇒ peu réaliste
 - S2:
 - **n° d'interruption ⇒ procédure adéquate**
 - **table des adresses de procédures indexées par le n° de l'interruption**



- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Interruptions

- **n° des interruptions:**
 - **interruption logicielle:**
 - n° dans l'instruction
 - Ex: INT 21
 - **interruption matérielle:**
 - n° généré par le dispositif qui a activé la ligne physique en fonction de la cause
 - Ex: IT1 \Leftrightarrow appui d'une touche
IT2 \Leftrightarrow lâché d'une touche
- **n° donnés par le système d'exploitation à l'UE lors de l'initialisation (idem pour n° de ports)**
- **Plug and Play:**
 - Interrogation de l'UE par le système d'exploitation: Nbr de ports et d'interruptions qu'elle peut gérer
 - SE fournit n° de ports et d'interruptions à l'UE

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Interruptions

- Procédure de l'interruption fournie par:
 - le système d'exploitation, ex: temps partagé
 - le pilote du périphérique fourni avec celui-ci
- Traitement d'une interruption (identique procédure):
 - **sauvegarde du registre d'état de l'UT**
 - sauvegarde de l'adresse de retour dans la pile
 - récupération du n° de l'interruption sur le bus de données (matériel) ou dans l'instruction (logiciel)
 - lecture de l'adresse de la procédure dans la table
 - exécution de la procédure
 - **restauration du registre d'état**
 - restauration de l'adresse de retour
- Instruction **RETI**:
 - dépile registre d'état
 - dépile adresse de retour

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions

Interruptions

- Problématique liée au registre d'état lors du traitement d'une interruption:

Événement



CMP
BNE

R0,10
sinon

; si différent

- Cas d'une interruption après la comparaison
 - Évaluation de la condition à partir du registre d'état de l'UT
 - pb si l'interruption le modifie
- ⇒ Nécessité de sauvegarder le registre d'état

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Interruptions

- **Priorités entre interruptions:**
 - **Cas: Interruption lors d'une procédure d'interruption**
 - Cas 1: ignorer si moins important
 - Cas 2: traiter si plus important
 - **Hiérarchisation: Priorité + ou – importante**
 - **Choix des priorités:**
 - systèmes d'exploitation classiques: SE
 - systèmes industriels: définies par les programmeurs grâce à des systèmes d'exploitations spécialisés
 - **En général: interruption de priorité supérieure aux autres sur une seconde ligne physique NMI (non-maskable - interrupt interruption non masquable)**

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Interruptions

- **Masquage des interruptions:**
 - Pour interdire l'interruption de l'exécution d'un morceau de code sensible
 - Masquage grâce à des instructions spéciales
 - ex Pentium: CLI, STI
 - instructions parfois réservées aux systèmes d'exploitation

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Interruptions

- **Niveau de privilèges:**
 - accordé à un programme
 - autorise ou non l'accès à certaines instructions

- **Minimum 2 niveaux:**
 - programme normal
 - système d'exploitation

- **Exemple:**
 - programme normal
 - extensions du systèmes d'exploitation (personnalisation, sécurité, optimisation...)
 - système d'exploitation
 - noyau du système d'exploitation (ordonnancement, gestion mémoire et matériel, appels système)

- 1- Modèle de microprocesseur
- 2- Programmation en assembleur
- 3- Procédures et fonction
- 4- Programmation avancée
- 5- Interruptions



Interruptions

Exemple du Pentium

- **Niveau de privilèges:**
 - **augmentation du niveau de privilège: instruction**
 - **baisse du niveau de privilège: uniquement par interruption**
 - procédure d'interruption: niveau de privilège maximal
 - possibilité pour l'interruption de baisser son niveau de privilège
 - après l'interruption retour au niveau de privilège précédent:
 - **début d'IT: sauvegarde dans le registre d'état**
 - **fin d'IT: restitution**