ARCHI – Architecture des ordinateurs
Sylvain Brandel
2023 – 2024
sylvain.brandel@univ-lyon1.fr

CM 10

LANGAGE D'ASSEMBLAGE DU LC-3

Partie 2

Routines, pile

Instructions

ISA (Instruction Set Architecture)

	Syntaxe	action	NZP	codage															
				opcode				arguments											
				F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
Arith - logique	NOT DR, SR	DR <- not SR	*	1	0	0	1	DR			SR			1	1	1	1	1	1
	ADD DR, SR1, SR2	DR <- SR1 + SR2	*	0	0	0	1	DR			SR1			0	0	0	SR2		
	ADD DR, SR1, Imm5	DR <- SR1 + SEXT(Imm5)	*	0	0	0	1	DR			SR1			1	Imm5				
	AND DR, SR1, SR2	DR <- SR1 and SR2	*	0	1	0	1	DR			SR1			0	0	0	SR2		
	AND DR, SR1, Imm5	DR <- SR1 and SEXT(Imm5)	*	0	1	0	1	DR			SR1			1	Imm5				
charg. rang.	LEA DR,label	DR <- PC + SEXT(PCoffset9)	*	1	1	1	0	DR			PCoffset9								
	LD DR,label	DR <- mem[PC + SEXT(PCoffset9)]	*	0	0	1	0	DR			PCoffset9								
	ST SR,label	mem[PC + SEXT(PCoffset9)] <- SR		0	0	1	1	SR			PCoffset9								
	LDR DR,BaseR,Offset6	DR <- mem[BaseR + SEXT(Offset6)]	*	0	1	1	0	DR			BaseR			Offset6					
	STR SR,BaseR,Offset6	mem[BaseR + SEXT(Offset6)] <- SR		0	1	1	1	SR			BaseR			Offset6					
branchement	BR[n][z][p] label Si (cond)	PC <- PC + SEXT(PCoffset9)		0	0	0	0	n	z	p	PCoffset9								
	~~NOP~~	~~No Operation~~		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	RET	PC <- R7		1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0
	JSR label	R7 <- PC ; PC <- PC + SEXT(PCoffset11)		0	1	0	0	1	PCoffset11										

Exemple 2

- Compte le nombre de caractères d'une chaîne et met le résultat en mémoire

```
          .ORIG x3000
          ; ici viendra notre code
          HALT
string:  .STRINGZ "Hello World »    ; la chaine dont on veut calculer la
                                    ; longueur
res:      .BLKW #1                  ; le résultat sera mis ici (1 case)
          .END
```

- R0 : pointeur de chaîne ; R1 : compteur ; R2 : caractère courant

```
R0 <- string;     // R0 pointe vers le début de la chaîne
R1 <- 0;          // Le compteur R1 est initialisé à 0
while((R2 <- Mem[R0]) != '\0') {
  R0 <- R0+1;     // Incrémentation du pointeur
  R1 <- R1+1;     // Incrémentation du compteur
}
res <- R1;        // Rangement du résultat
```

Exemple 2

```
        .ORIG x3000
        LEA R0,string     ; Initialisation du pointeur R0
        AND R1,R1,0       ; Le compteur R1 est initialisé à 0
loop:   LDR R2,R0,0       ; Chargement dans R2 du caractère pointé par R0
        BRz end           ; Test de sortie de boucle
        ADD R0,R0,1       ; Incrémentation du pointeur
        ADD R1,R1,1       ; Incrémentation du compteur
        BR loop
end:    ST R1,res
        HALT
string: .STRINGZ "Hello World"
res:    .BLKW #1
        .END
```

- Maintenant, on a envie de réutiliser le calcul de la longueur

Routines

- Plusieurs fois la même opération / même code ? → routine.

- Routine : bloc de code accessible
  - Étiquette à la première instruction de la routine
  - Appel par saut (`JSR`)
    → Modification de PC
  - Retour (`RET`) à l'instruction suivant l'appel
    → Connaitre l'adresse de retour Mémorisation (R7)

- Routine : fonction dans un langage de haut niveau

Routines

branchement	BR[n][z][p] label Si (cond)	PC <- PC + SEXT(PCoffset9)		0	0	0	0	n	z	p	PCoffset9								
	~~NOP~~	~~No Operation~~		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	JMP BaseR	PC <- BaseR		1	1	0	0	0	0	0	BaseR			0	0	0	0	0	0
	RET	PC <- R7		1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0
	JSR label	R7 <- PC ; PC <- PC + SEXT(PCoffset11)		0	1	0	0	1	PCoffset11										
	TRAP Trapvect8	R7 <- PC ; PC <- mem[Trapvect8]		1	1	1	1	0	0	0	0	Trapvect8							

- JSR
  - Stocke l'adresse de retour dans R7
  - Modifie PC ← adresse de label

branchement

```
; programme principal
...
JSR sub  ; adresse de l'instruction suivante dans R7
...
```

- RET (équivalent à JMP R7)
  - Retour à la routine appelante
  - Modifie PC ← R7

ATTENTION AUX TRAP

```
sub: ...
     ...
     RET      ; retour au programme principal
```

Exemple 2 bis

- On reprend le calcul de la longueur d'une chaine

```
        .ORIG x3000
        LEA R0,string     ; Initialisation du pointeur R0
        AND R1,R1,0       ; Le compteur R1 est initialisé à 0
loop:   LDR R2,R0,0       ; Chargement dans R2 du caractère pointé par R0
        BRz end           ; Test de sortie de boucle
        ADD R0,R0,1       ; Incrémentation du pointeur
        ADD R1,R1,1       ; Incrémentation du compteur
        BR loop
end:    ST R1,res
        HALT
string: .STRINGZ "Hello World"
res:    .BLKW #1
        .END
```

- Routine pour le calcul de la longueur
  - Adresse de la chaine dans R0
  - Résultat dans R0

Exemple 2 bis

```
        .ORIG x3000
        LEA R0,string ; Initialisation du pointeur R0
        JSR length ; saut vers la routine
        ST R0,res
        HALT
; données
string:  .STRINGZ "Hello"
res:     .BLKW #1

; Routine pour calculer la longueur d'une chaîne (terminée par '\0')
; Entrée : R0 adresse de la chaîne
; Sortie : R0 longueur de la chaîne
length:  AND R1, R1, 0
loop:    LDR R2, R0, 0
         BRz end
         ADD R0, R0, 1
         ADD R1, R1, 1
         BR loop
end:     ADD R0, R1, 0
         RET
```

Routines imbriquées

- Appel à une routine dans une routine ? TRAP dans une routine ?

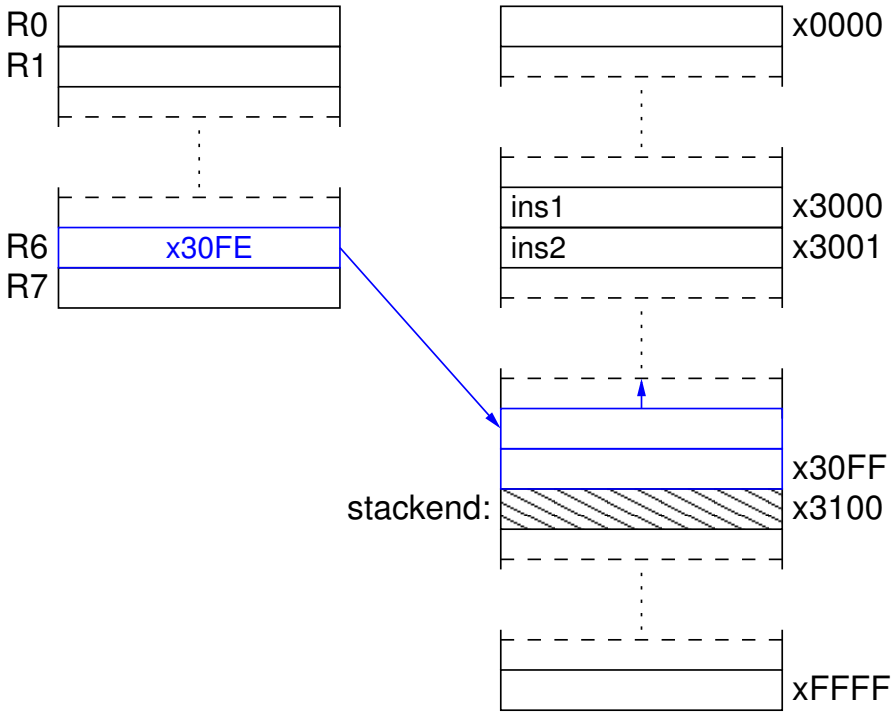
```
; programme principal
          ...
          JSR sub1
          ...

; routine sub1
sub1:     ...
          JSR sub2
          ...
          RET
                    ; Problème : ici R7 ne contient pas l'adresse de retour
                    ; vers sub1 (puisque sub2 a été appelée juste avant...)

; routine sub2
sub2:     ...
          RET
```

- Pareil pour les registres …

  → Pile d'exécution

Pile d'exécution

- Appel à une routine dans une routine ? R7 change → pile
- Pile : structure pour mémoriser adresses et registres
  - Sommet (dernier ajouté) pointé par R6 (fond : stackend)
  - Push → nouveau sommet → modification de R6 + mémorisation
  - Pop → lecture + nouveau sommet → modification de R6
  - La pile croît dans le sens des adresses décroissantes
  - Installation :
    - De la place (assez, pas trop)
    - Un fond
    - Initialisations



Pile d'exécution

```
            .ORIG x3000
; Programme principal
main:       LD R6,spinit ; on initialise le pointeur de pile
            ...
            HALT
; Gestion de la pile
spinit:     .FILL stackend
            .BLKW #15
stackend:   .BLKW #1 ; adresse du fond de la pile
.END
```

- Push
  - Bouger le pointeur de sommet
  - Stocker le registre à sauver
    → Avant appel de la routine

```
ADD R6, R6, -1
STR Rx, R6, 0
```

- Pop
  - Récupérer la donnée au sommet
  - Bouger le pointeur de sommet
    → Après retour de la routine

```
LDR Rx, R6, 0
ADD R6, R6, 1
```

Routines imbriquées

```
; programme principal
          ...
         LD R6, spinit      ; initialisation du pointeur de pile
         JSR sub1
         ...

; routine 1
sub1:    ...
          ADD R6, R6, -1    ; sauvegarde R7
         STR R7, R6, 0
         JSR sub2
         LDR R7, R6, 0      ; restauration R7
         ADD R6, R6, 1
          ...
         RET

; routine 2
sub2:    ...
         RET

; Mémoire réservée à la pile
spinit:    .FILL stackend
           .BLKW #15
stackend: .BLKW #1 ; adresse du fond de la pile
```

Exemple 3

- Somme des entiers de 1 à n version récursive

```
int sum(int n) {
  if (n==0) return 0; else return n + sum(n-1);
}
```

- R1 : paramètre d'entrée, R0 : paramètre de sortie

```
routine sum(R1) {
  push(R7);            // Sauvegarde de R7 sur la pile
  push(R1);            // Sauvegarde de R1 sur la pile
  if (R1 == 0) {
    R0 <- 0;
  }
  else {
    R1 <- R1 - 1;      // On prépare le paramètre de l'appel récursif
    call sum(R1);      // Appel récursif
    R1 <- top();       // R1 reprend sa valeur depuis le sommet de pile
    R0 <- R0 + R1;
  }
  R1 <- pop();         // Restauration de R1 depuis la pile
  R7 <- pop();         // Restauration de R7 depuis la pile
  return;              // Résultat retourné via R0
}
```

Exemple 3

```
; Programme principal
main:    LD R6,spinit      ; on initialise le pointeur de pile
         LD R1,n           ; on initialise R1 avant l'appel à la routine sum
         JSR sum           ; appel à la routine sum
         ST R0,r           ; on stocke le résultat à l'adresse r
         HALT

; Données
n:       .FILL #5
r:       .BLKW #1

; Routine sum
; paramètre d'entrée : R1
; paramètre de sortie : R0
sum:     ...
         RET               ; on retourne à l'appelant

; Mémoire réservée à la pile
spinit:  .FILL stackend
         .BLKW #15
stackend: .BLKW #1         ; adresse du fond de la pile
```

Exemple 3

```
; Routine sum.
; Paramètre d'entrée  : R1
; Paramètre de sortie : R0
sum:    ADD R6,R6,#-1
        STR R7,R6,#0     ; On sauvegarde l'adresse de retour R7.
        ADD R6,R6,#-1
        STR R1,R6,#0     ; On sauvegarde l'adresse de retour R1.

if:     ADD R1,R1,#0     ; Mise à jour du PSR d'après R1.
        BRnp else        ; Si R1 != 0, branchement sur le bloc else
        AND R0,R0,#0     ; Si R1 == 0, la valeur de retour est 0,
        BR endif         ;             et on ne fait pas d'appel récursif.

else:   ADD R1,R1,#-1    ; Dans le cas R1 == 0, on prépare l'appel récursif.
        JSR sum          ; Appel récursif.
        LDR R1,R6,#0     ; On récupère la valeur de R1 avant l'appel récursif.
        ADD R0,R0,R1     ; On met le résultat dans le paramètre de sortie R0.

endif:  LDR R1,R6,#0     ; On restaure le registre R1.
        ADD R6,R6,#1
        LDR R7,R6,#0     ; On restaure le registre R7.
        ADD R6,R6,#1
        RET              ; On retourne à l'appelant
```