Contenu de la micro mémoire et du décodeur d'instructions

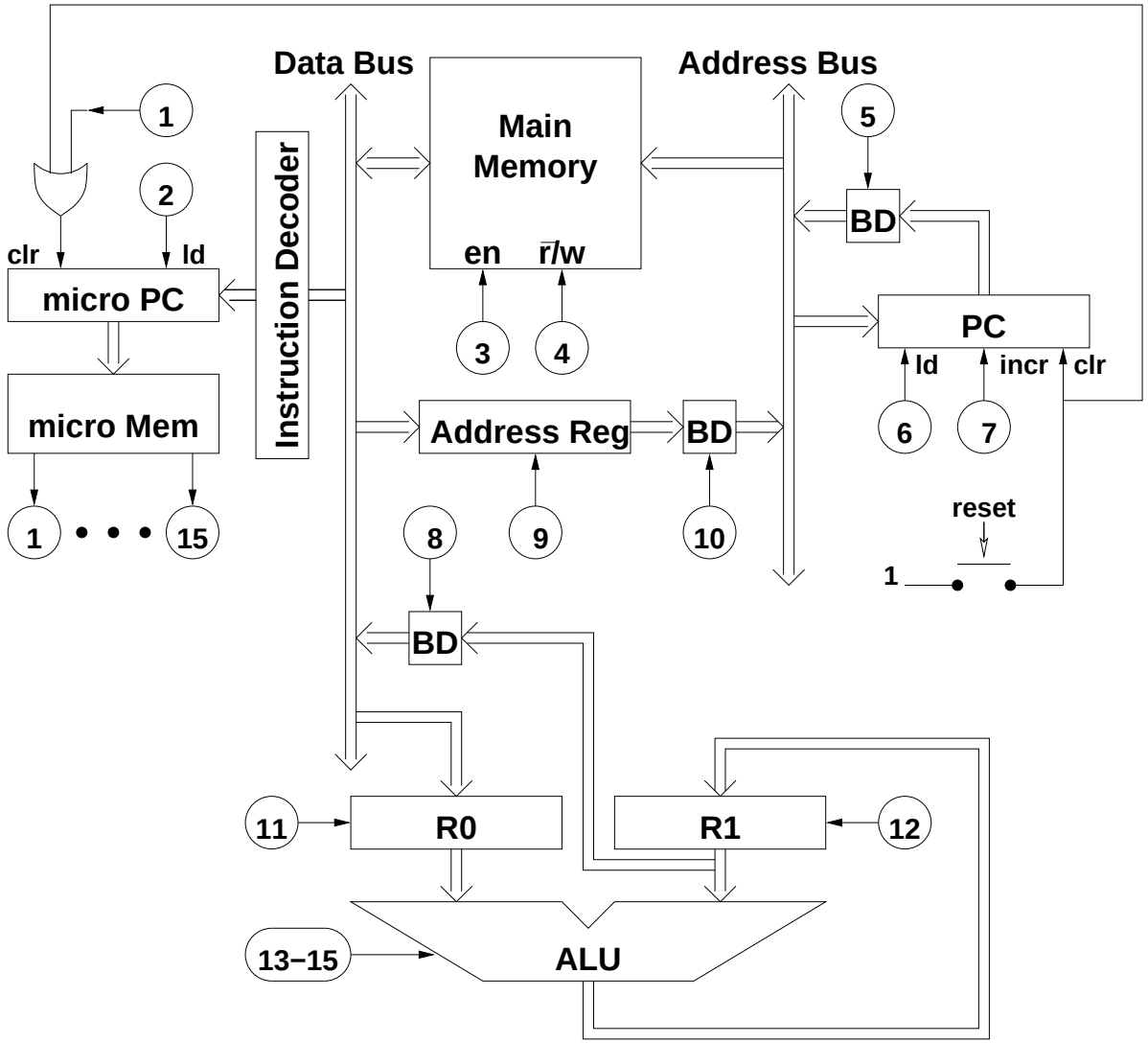
micro mémoire

000000:	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0
000001:	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
000010:	1	0	1	0	1	0	1	0	0	0	1	0	0	0	0
000011:	0	0	1	0	1	0	1	0	1	0	0	0	0	0	0
000100:	1	0	1	0	0	0	0	0	0	1	1	0	0	0	0
000101:	0	0	1	0	1	0	1	0	1	0	0	0	0	0	0
000110:	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0
000111:	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0
001000:	0	0	1	1	0	0	0	1	0	1	0	0	0	0	0
001001:	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0
001010:	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0
001011:	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0
001100:	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1
001101:	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0
001110:	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1
001111:	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0
010000:	1	0	0	0	0	0	0	0	0	0	0	1	1	0	1
010001:	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0
010010:	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1
010011:	0	0	1	0	1	0	1	0	1	0	0	0	0	0	0
010100:	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0
010101:															
...															

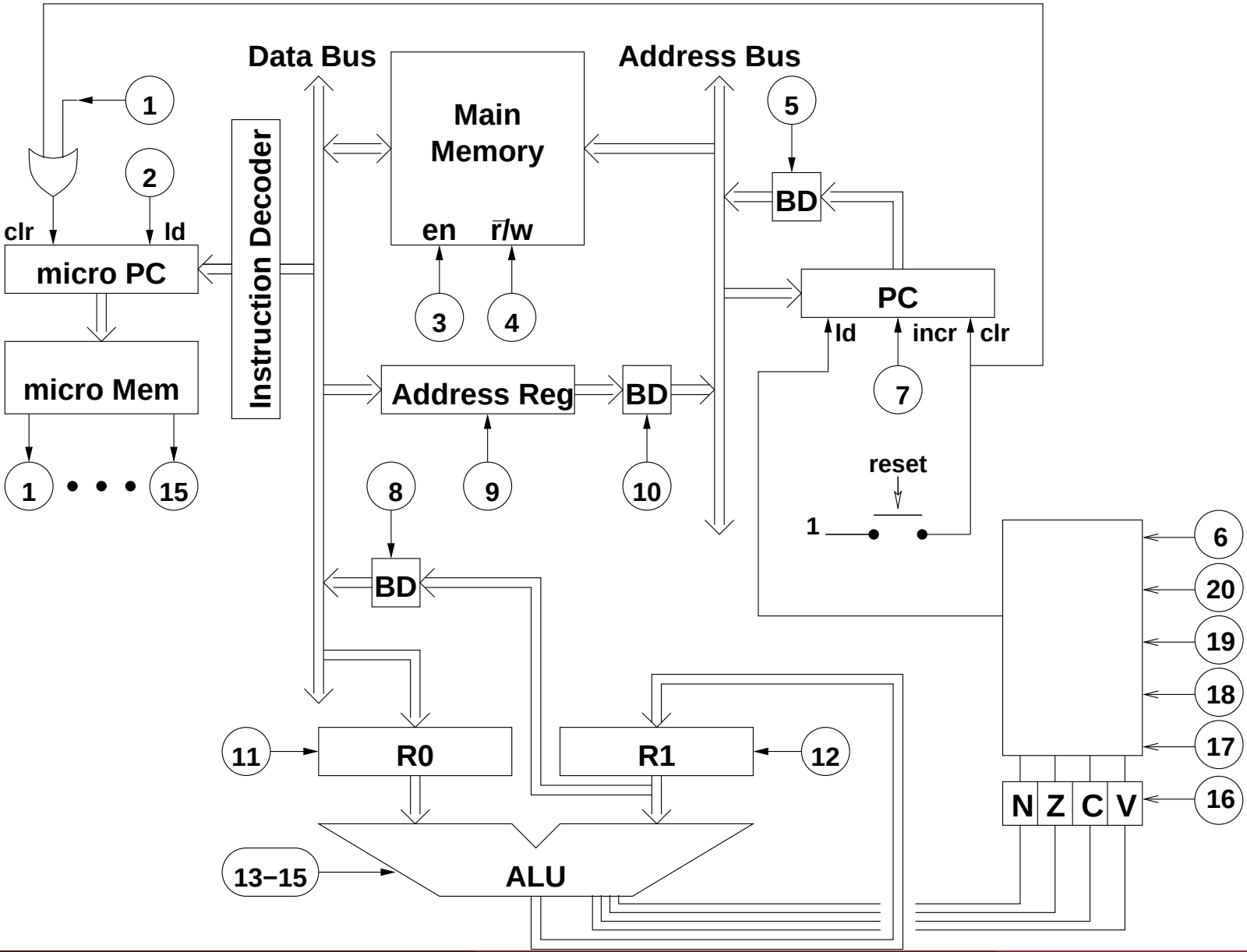
décodeur d'instructions

00000:	000001	NOP
00001:	000010	LDIMM
00010:	000011	LD
00011:	000101	ST
00100:	001011	COPY
00101:	001100	SHL
00110:	001101	SHR
00111:	001110	ADD
01000:	001111	SUB
01001:	010000	AND
01010:	010001	OR
01011:	010010	NOT
01100:	010011	JAL
01101:		
01110:		
01111:		
10000:		
10001:		
10010:		
10011:		
10100:		
10101:		
...		

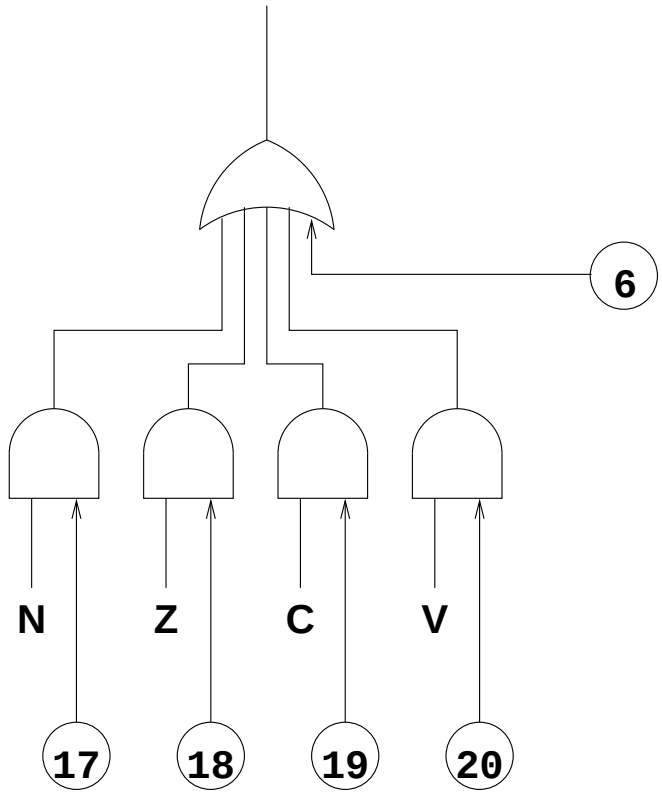
Le premier ordinateur



Sauts conditionnels



Contenu du circuit



Contenu de la micro mémoire et du décodeur d'instructions

micro mémoire

000000:	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
000001:	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
000010:	1	0	1	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0
000011:	0	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0
000100:	1	0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
000101:	0	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0
000110:	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
000111:	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
001000:	0	0	1	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
001001:	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
001010:	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
001011:	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
001100:	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0
001101:	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0
001110:	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	0	0
001111:	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0
010000:	1	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	0
010001:	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	0	0	0
010010:	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0
010011:	0	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0
010100:	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0
010101:																			
...																			

décodeur d'instructions

00000:	000001	NOP
00001:	000010	LDIMM
00010:	000011	LD
00011:	000101	ST
00100:	001011	COPY
00101:	001100	SHL
00110:	001101	SHR
00111:	001110	ADD
01000:	001111	SUB
01001:	010000	AND
01010:	010001	OR
01011:	010010	NOT
01100:	010011	JAL
01101:		
01110:		
01111:		
10000:		
10001:		
10010:		
10011:		
10100:		
10101:		
...		

Contenu de la micro mémoire et du décodeur d'instructions

micro mémoire

001010:	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
001011:	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
001100:	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0
001101:	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0
001110:	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	0	0
001111:	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0
010000:	1	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	0
010001:	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	0	0	0
010010:	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0
010011:	0	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0
010100:	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0
010101:	0	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0
010110:	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0
010111:	0	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0
011000:	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0
011001:	0	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0
011010:	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0
011011:	0	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0
011100:	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
...																			

décodeur d'instructions

00000:	000001	NOP
00001:	000010	LDIMM
00010:	000011	LD
00011:	000101	ST
00100:	001011	COPY
00101:	001100	SHL
00110:	001101	SHR
00111:	001110	ADD
01000:	001111	SUB
01001:	010000	AND
01010:	010001	OR
01011:	010010	NOT
01100:	010011	JAL
01101:	010101	JN
01110:	010111	JZ
01111:	011001	JV
10000:	011011	JC
10001:		
10010:		
10011:		
10100:		
...		

Sous-programmes

En C:

```
f()
{
  h();
}
...
g()
{
  h();
}
```

```
h()
{
  ...
  return;
}
```

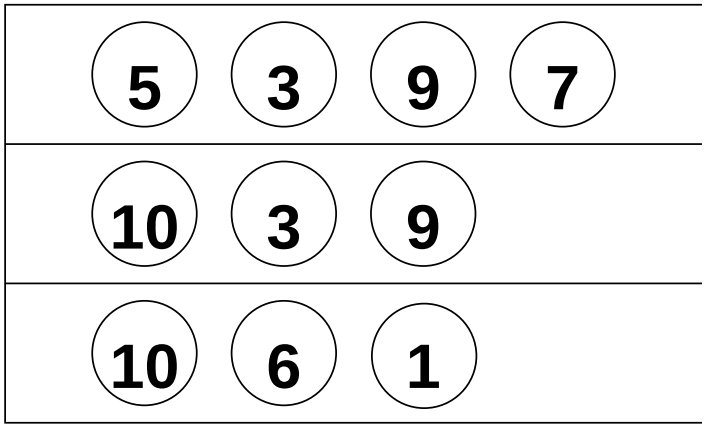
Problème principal: Le sous-programme doit connaître l'adresse de l'appelant

Sous-programmes

Solution utilisée par Fortran (il y a longtemps)
Cette solution dépend de l'existance d'une instrution jin (jump indirect)

Sous programmes

L'instruction jin est réalisable avec l'architecture actuelle :



Problème : On utilise les registres pour stocker l'adresse hret
Solution : Introduire une instruction jsr (jump to subroutine)

Sous-programmes

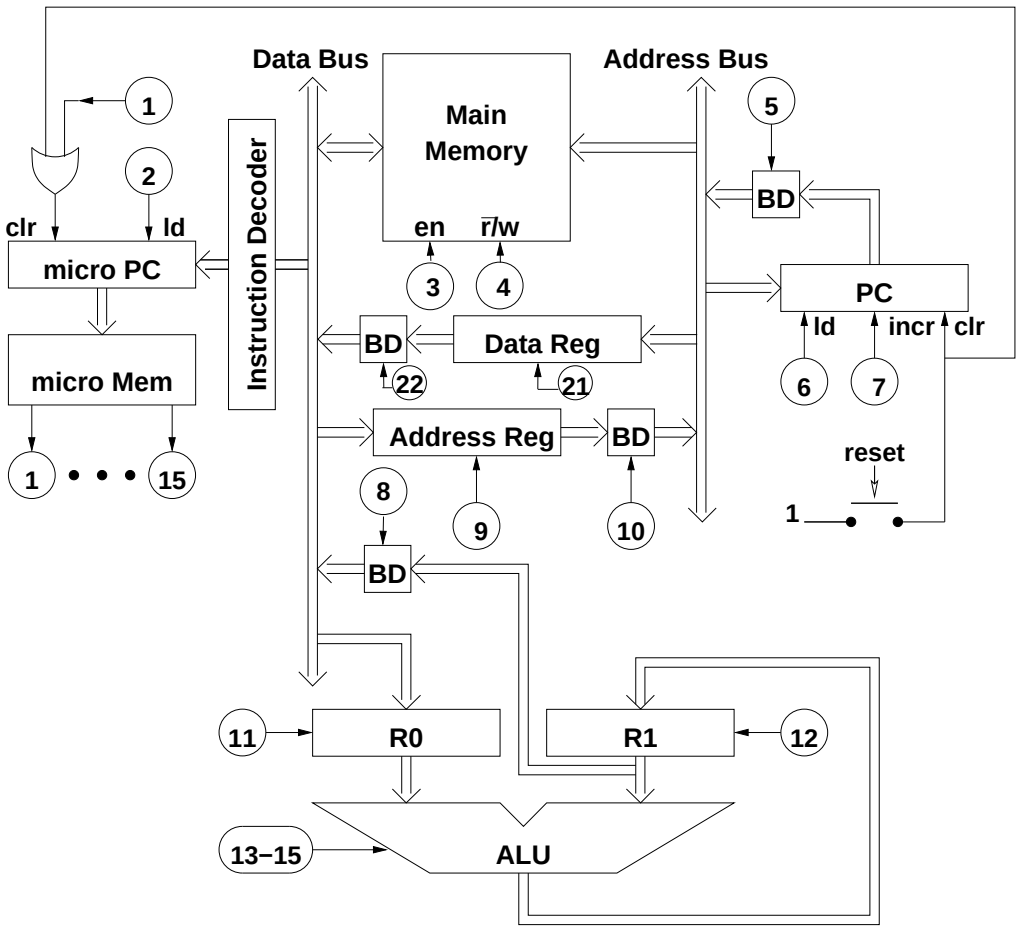
Utilisation de jsr :

```
f:      ...
        jsr h-1
        ...

        ...
g:      jsr h-1
        ...

        0
h:      ...
        jin h-1
```

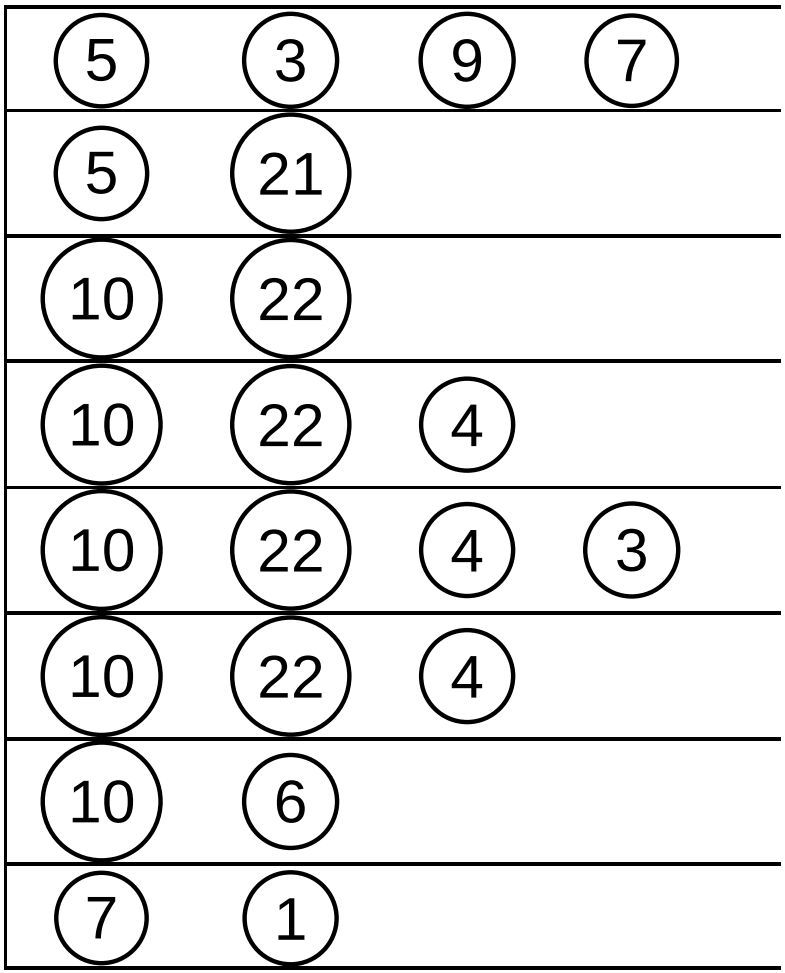
L'instruction jsr n'est pas réalisable avec l'architecture actuelle

1. Stocker la valeur de PC dans l'adresse donnée (ici h-1)
2. Charger l'adresse donnée + 1 (donc ici h) dans PC

Modifications pour jsr

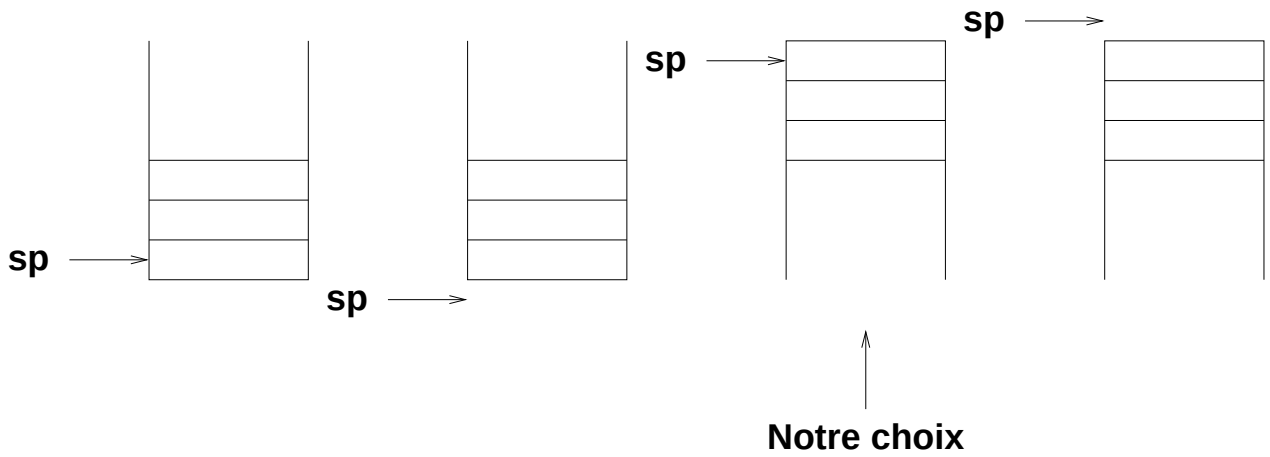


Micro-Programme pour JSR

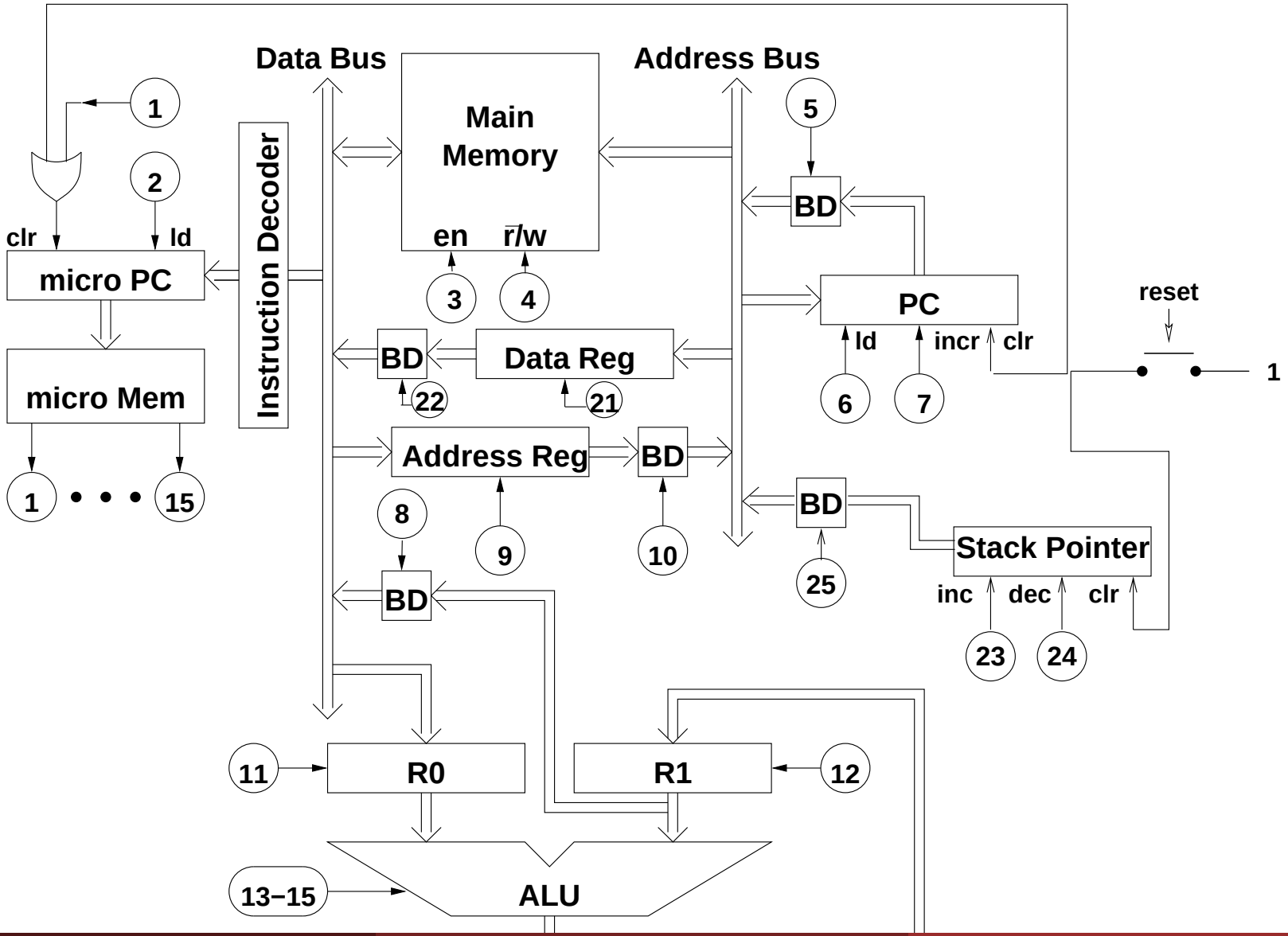


Récursivité

- Le jsr actuel ne permet pas la récursivité
- Pour la réaliser, il faut une pile
- La pile est définie par un registre : pointeur de pile (sp, stack pointer)
- Ce registre doit permettre l'incrémentation et la décrémentation
- Plusieurs conventions de la pile sont possibles :



Modification pour la pile



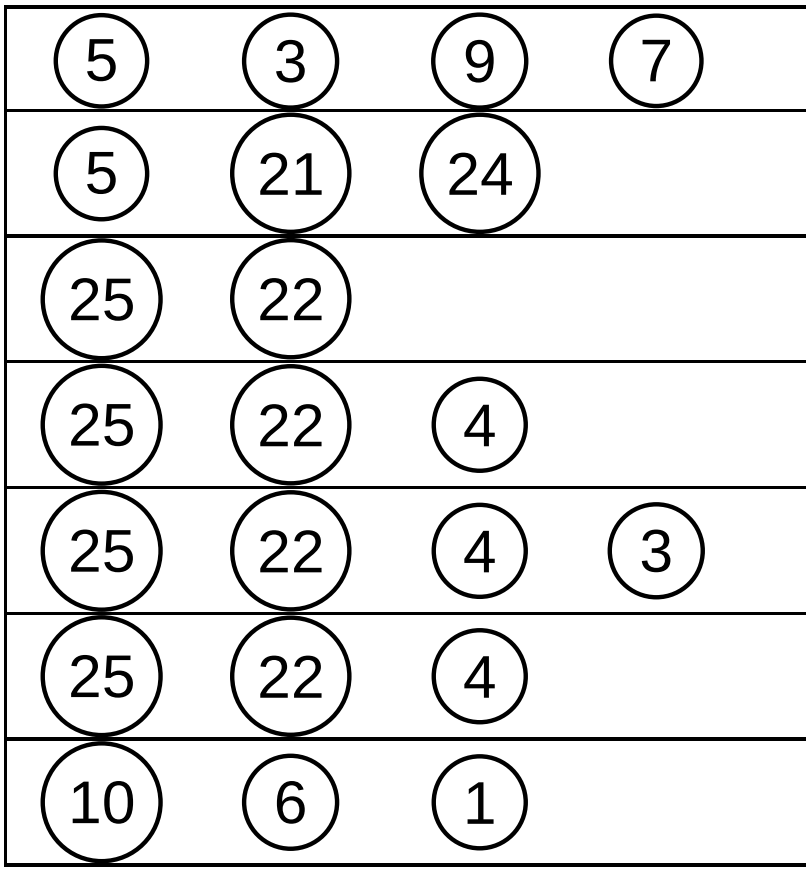
Instruction JSR et RET avec pile

Utilisation :

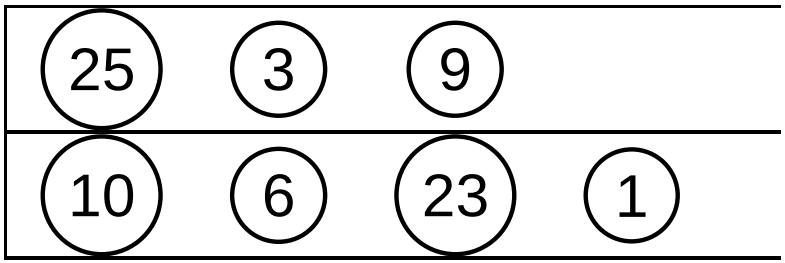
```
f:    ...
      jsr h
      ...

g:    ...
      jsr h
      ...

h:    ...
      ret
```

- Description de JSR :
  1. empiler PC
  2. charger l'adresse donnée dans PC
- Description de RET
  1. mettre le sommet de la pile dans PC
  2. Dépiler

Micro-Programme pour JSR avec pile



Micro-Programme pour RET avec pile

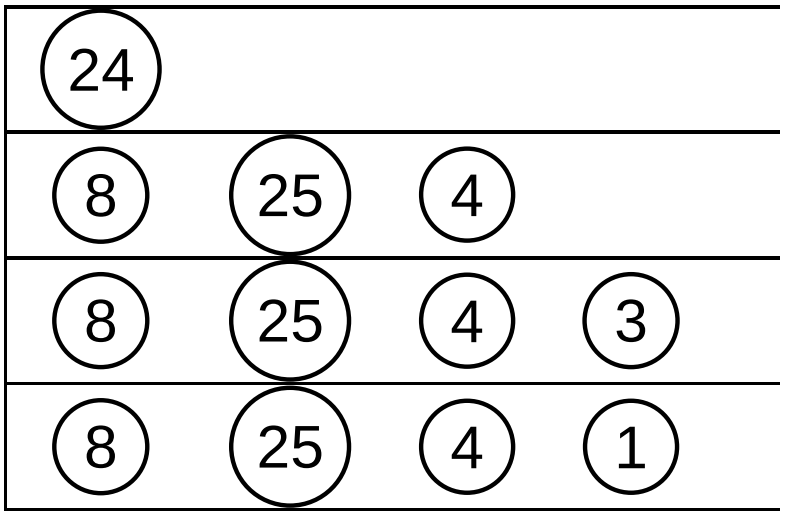


Utilisation de la pile pour passage de paramètres

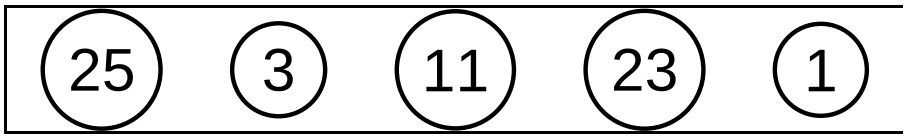
- Avant de faire un JSR, empiler les arguments à fournir au sous-programme
- Le sous-programme peut dépiler
- Nous avons besoin de 2 instructions supplémentaires :
  - PUSH : empiler le contenu de R1
  - POP : mettre le sommet de la pile dans R0 et dépiler
- Ces deux instructions sont réalisables avec l'architecture actuelle

Implémentation de PUSH et POP

PUSH:



POP:



Protocole d'appel de sous-programmes

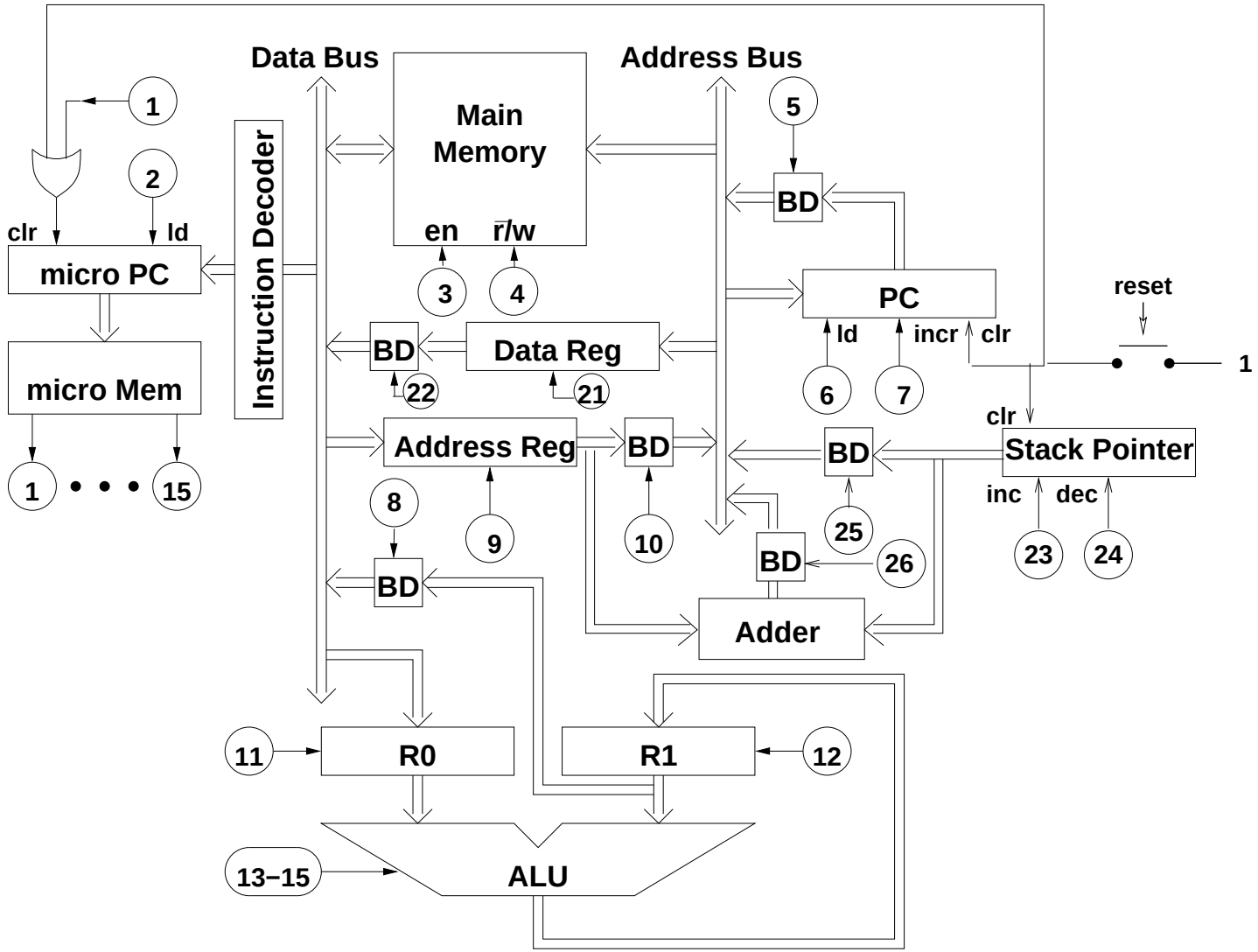
- Un protocole d'appel est un ensemble de règles précises pour déterminer le partage de responsabilités entre un sous-programme appelant et un sous-programme appelé.
- Exemple de protocole d'appel (plusieurs implémentations du langage C) :

Appelant	Appelé
calculer argument n dans R1 empiler R1 ... calculer argument 1 dans R1 empiler R1 JSR appelé dépiler les n arguments	 Calcul éventuel de valeur dans R1 RET

Accés aux arguments dans un sous-programme

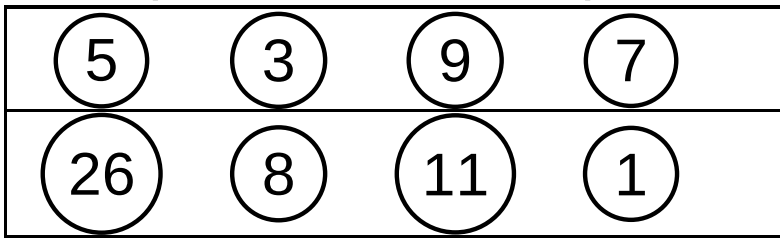
- Actuellement, la seule possibilité est d'utiliser POP, ce qui n'est pas pratique
- Il nous faut une possibilité d'accéder à une valeur en mémoire dont l'adresse est sp+constante
- De cette manière, le sous-programme peut récupérer argument i avec l'adresse sp+(i+1)
- Nous avons donc besoin d'additionner le contenu de sp avec une constante
- Cette constante sera stockée après le code d'instruction, et copiée dans le registre d'adresses

Modification pour l'accès aux arguments



Nouvelle instruction pour l'accès aux arguments

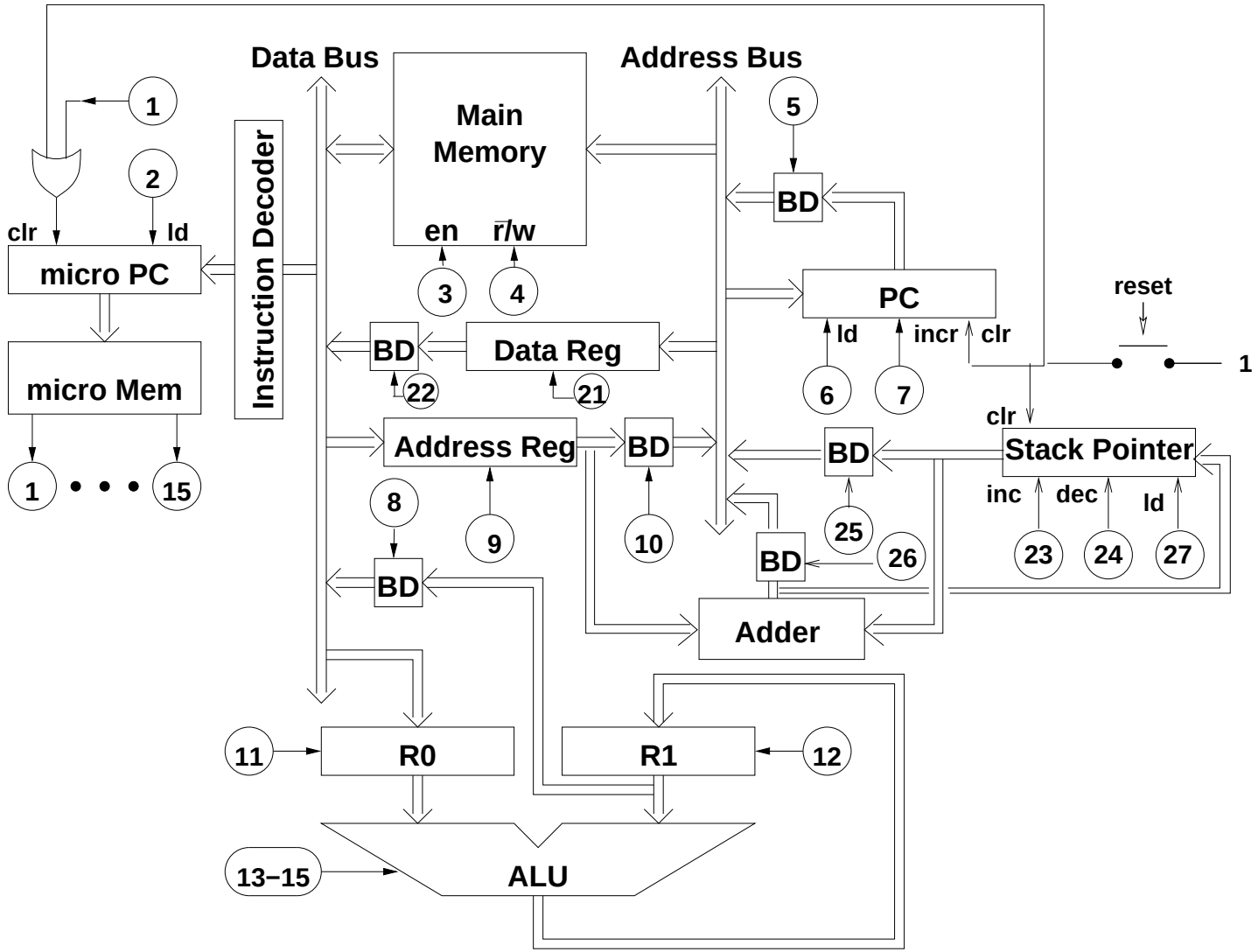
LDS (Load from stack)



Dépiler les arguments

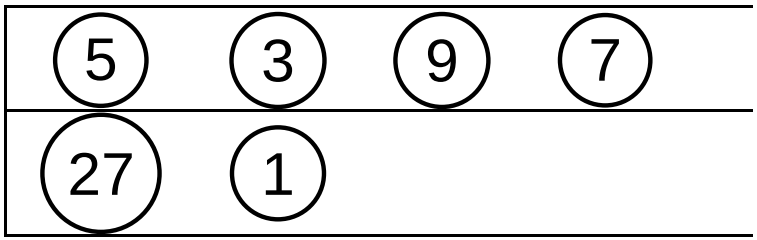
- Après le retour de l'appelé, l'appelant doit dépiler les arguments
- Actuellement, la seule possibilité est d'utiliser POP
- Une amélioration possible est de pouvoir additionner une constante à sp
- Il est possible d'utiliser l'additionneur existant, mais il faut pouvoir stocker le résultat dans sp
- Ceci nécessite des fils supplémentaires, et une modification de sp pour permettre un signal ld

Modifications pour l'arithmétique de SP



Nouvelle instruction pour l'arithmétique de SP

ADDSP:



l'addition est toujours modulo 256, donc pas besoin de SUBSP

Variables locales à un programme

Avec les instructions maintenant à notre disposition, il est possible d'avoir des variables locales allouées dans la pile.
Voici le format de la pile pour un sous programme avec n arguments et m variables locales :

var loc m
...
var loc 1
adresse de retour
arg 1
...
arg n

Nouveau protocole d'appel

Appelant	Appelé
calculer argument n dans R1 empiler R1 ... calculer argument 1 dans R1 empiler R1 JSR appelé	
	ADDSP -m calculer en utilisant LDS, STS ADDSP m RET
ADDSP n	