Définition de la machine

- une machine à pile (pas de registres)
- 3 zones mémoires distinctes : programme, données et pile
- opérations arithmétiques et logiques entières
- 17 instructions élémentaires (seulement) !

Données disponibles

- `program[]` mémoire d'instructions
- `pc` *program counter*, instruction suivante à exécuter
- `data[]` mémoire de données
- `stack[]` pile de données entières
- `sp` *stack pointer*, prochaine donnée libre sur la pile

17 instructions élémentaires

mouvements de données PUSH LOAD STORE SWAP

entre les différentes zones mémoires

opérations arithmétiques/logiques ADD SUB MUL DIV AND OR NOT

sur les données de la pile, résultat dans la pile

controle du programme BEZ BGZ STOP GOTO

modifications de l'exécution

entrées/sorties interactives IN OUT

Mouvements de données

PUSH `pc++; stack[sp++] = program[pc++]`

LOAD `pc++; stack[sp-1] = data[stack[sp-1]]`

STORE `pc++; data[stack[sp-2]] = stack[sp-1]; sp-=2`

SWAP `pc++; stack[sp-1] <-> stack[sp-2]`

Opérations arithmetiques et logiques

ADD `pc++; stack[sp-2] = stack[sp-2]+stack[sp-1]; sp--`

SUB `pc++; stack[sp-2] = stack[sp-2]-stack[sp-1]; sp--`

MUL `pc++; stack[sp-2] = stack[sp-2]*stack[sp-1]; sp--`

DIV `pc++; stack[sp-2] = stack[sp-2]/stack[sp-1]; sp--`

AND `pc++; stack[sp-2] = stack[sp-2]&stack[sp-1]; sp--`

OR `pc++; stack[sp-2] = stack[sp-2]|stack[sp-1]; sp--`

NOT `pc++; stack[sp-1] = ~ stack[sp-1];`

Controle du programme

BEZ `pc++; pc = (stack[--sp]==0)?program[pc]:pc+1;`

BGZ `pc++; pc = (stack[--sp]>0)?program[pc]:pc+1;`

STOP `pc++; exit(0)`

GOTO `pc = stack[--sp];`

Entrée/Sorties interactives

IN `pc++; stack[sp++] = IN;`

OUT `pc++; OUT = stack[--sp];`

L'assembleur `asm`

- instructions symboliques
- calcul des labels pour les branchements

```
lab EQU *
```

- réservation et nommage de la mémoire donnée

```
nom DS taille
```

- élimination des commentaires

```
;/ ceci est un commentaire
```

Exemple de code assembleur

```
;/ réservation de mémoire donnée
var1    DS  1
var2    DS  1
;/ définition d'un label
lab1    EQU *
;/ un peu de code symbolique...
        PUSH  var1
        LOAD
        OUT
;/ branchement vers le label
        PUSH  lab1
        GOTO
```

$$A < B$$

```
        calculer B-A sur la pile...
        BGZ sivrai
        PUSH 0
        PUSH lafin
        GOTO
sivrai  EQU *
        PUSH 1
lafin   EQU *
```

if $A < B$ then code

```
;/ calculer A<B sur la pile...
        code-condition
        BEZ lafin
        code-then
lafin   EQU *
;/ suite du code...
```

IF cond THEN code-then ELSE code-else

```
;/ calculer la condition sur la pile
        code-condition
        BEZ labelse
        code-then
        PUSH labfin
        GOTO
labelse EQU *
        code-else
labfin  EQU *
```

WHILE cond code

```
debut   EQU *
;/ calculer la condition sur la pile
        code-condition
        BEZ fin
        code-corps
        PUSH debut
        GOTO
fin     EQU *
;/ et la suite...
```