PROGRAM ::= **program** ID **;** BLOCK **.**

BLOCK ::= CONSTS VARS INSTS

CONSTS ::= **const** ID = NUM **;** { ID = NUM **;** } | **ε**

VARS ::= **var** ID { **,** ID } **;** | **ε**

INSTS ::= **begin** INST { **;** INST } **end**

INST ::= INSTS | AFFEC | SI | TANTQUE | ECRIRE |LIRE||ε

AFFEC ::= ID **:=** EXPR

SI ::= **if** CO**ND then INST [else INST | ε]**

TANTQUE ::= **while** COND **do** INST

ECRIRE ::= **write (** EXPR { **,** EXPR } **)**

LIRE ::= **read (** ID { **,** ID } **)**

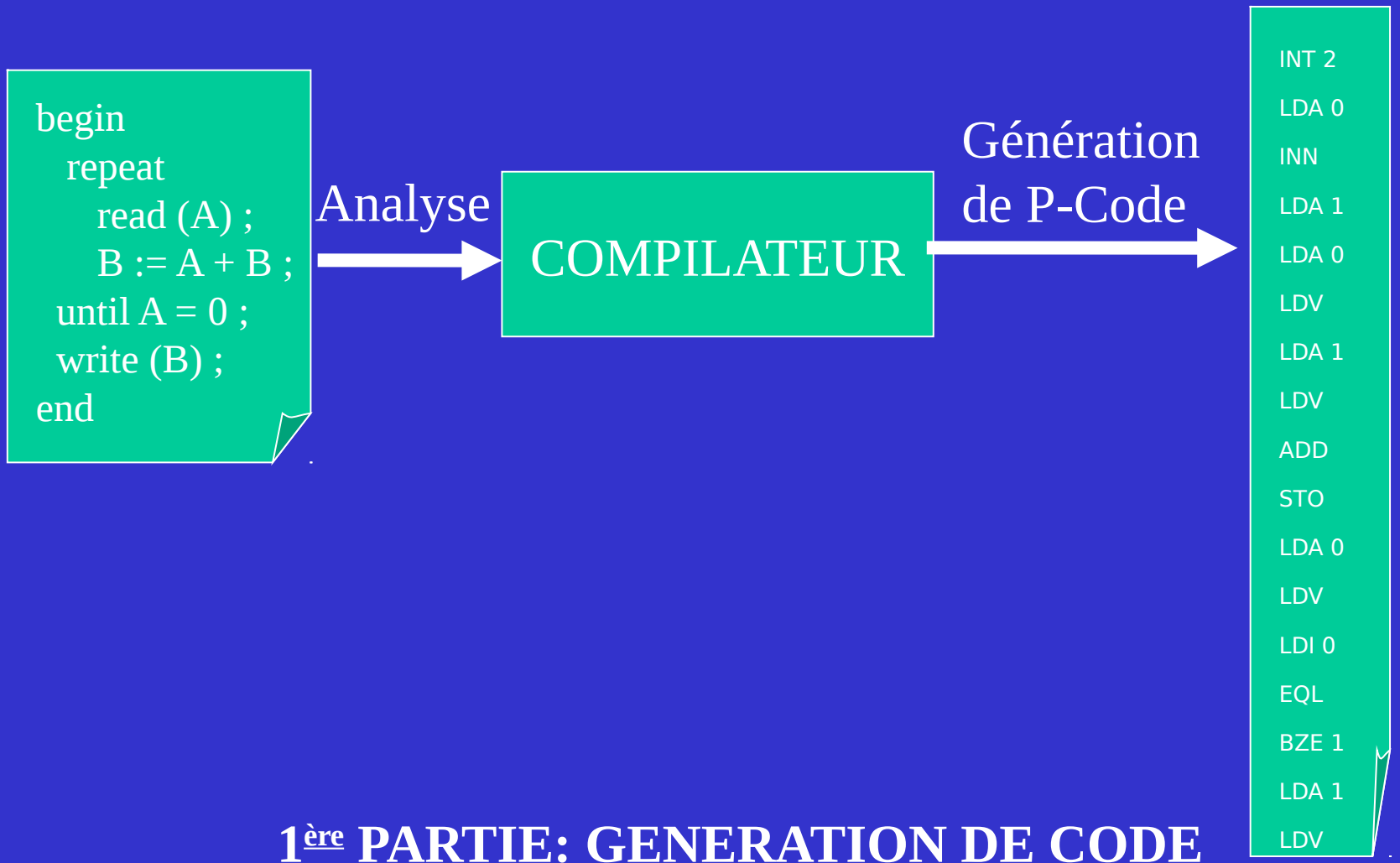
COND ::= EXPR RELOP EXPR

RELOP ::= = | <> | < | > | <= | >=

EXPR ::= TERM { ADDOP TERM }

ECRITURE DE L'INTERPRETEUR

SAUVEGARDER LE CODE GENERE DANS UN FICHIER



1ère PARTIE: GENERATION DE CODE

```
void SaveInstToFile(INSTRUCTION INST, int i)
{
  switch(  INST.MNE){
     case LDA:   fprintf(FICH_SORTIE, "%s \t %d \n", "LDA", INST.SUITE); break;
     case LDI:    fprintf(FICH_SORTIE, "%s \t %d \n", "LDI", INST.SUITE); break;
     case INT:    fprintf(FICH_SORTIE, "%s \t %d \n", "INT", INST.SUITE); break;
     case BZE:   fprintf(FICH_SORTIE, "%s \t %d \n", "BZE", INST.SUITE); break;
     case BRN:   fprintf(FICH_SORTIE, "%s \t %d \n", "BRN", INST.SUITE); break;
     case LDV:   fprintf(FICH_SORTIE, "%s  \n", "LDV");              break;
     case ADD:   fprintf(FICH_SORTIE, "%s  \n", "ADD");              break;
     case SUB:   fprintf(FICH_SORTIE, "%s  \n", "SUB");             break;
     case MUL:  fprintf(FICH_SORTIE, "%s  \n", "MUL");              break;
     case DIV:     fprintf(FICH_SORTIE, "%s  \n", "DIV");           break;
     case LEQ:   fprintf(FICH_SORTIE, "%s  \n", "LEQ");             break;
     case GEQ:   fprintf(FICH_SORTIE, "%s  \n", "GEQ");             break;
     case NEQ:   fprintf(FICH_SORTIE, "%s  \n", "NEQ");             break;
     case LSS:    fprintf(FICH_SORTIE, "%s  \n", "LSS");            break;
     case GTR:   fprintf(FICH_SORTIE, "%s  \n", "GTR");             break;
     case HLT:    fprintf(FICH_SORTIE, "%s  \n", "HLT");            break;
     case STO:    fprintf(FICH_SORTIE, "%s  \n", "STO");            break;
     case INN:    fprintf(FICH_SORTIE, "%s  \n", "INN");            break;
     case PRN:   fprintf(FICH_SORTIE, "%s  \n", "PRN");             break;
     default: Erreur(INST_PCODE_ERR);                    break;
  }
    }
```

SAUVEGARDER LE CODE GENERE DANS UN FICHIER

```
FILE  *FICH_SORTIE;

FICH_SORTIE=fopen("sortie.pc", "w" );

void SavePCodeToFile() {
 int i;
 for (i=0; i<=PC; i++) {SaveInstToFile(PCODE[i])}
     }

Fclose(FICH_SORTIE);
```

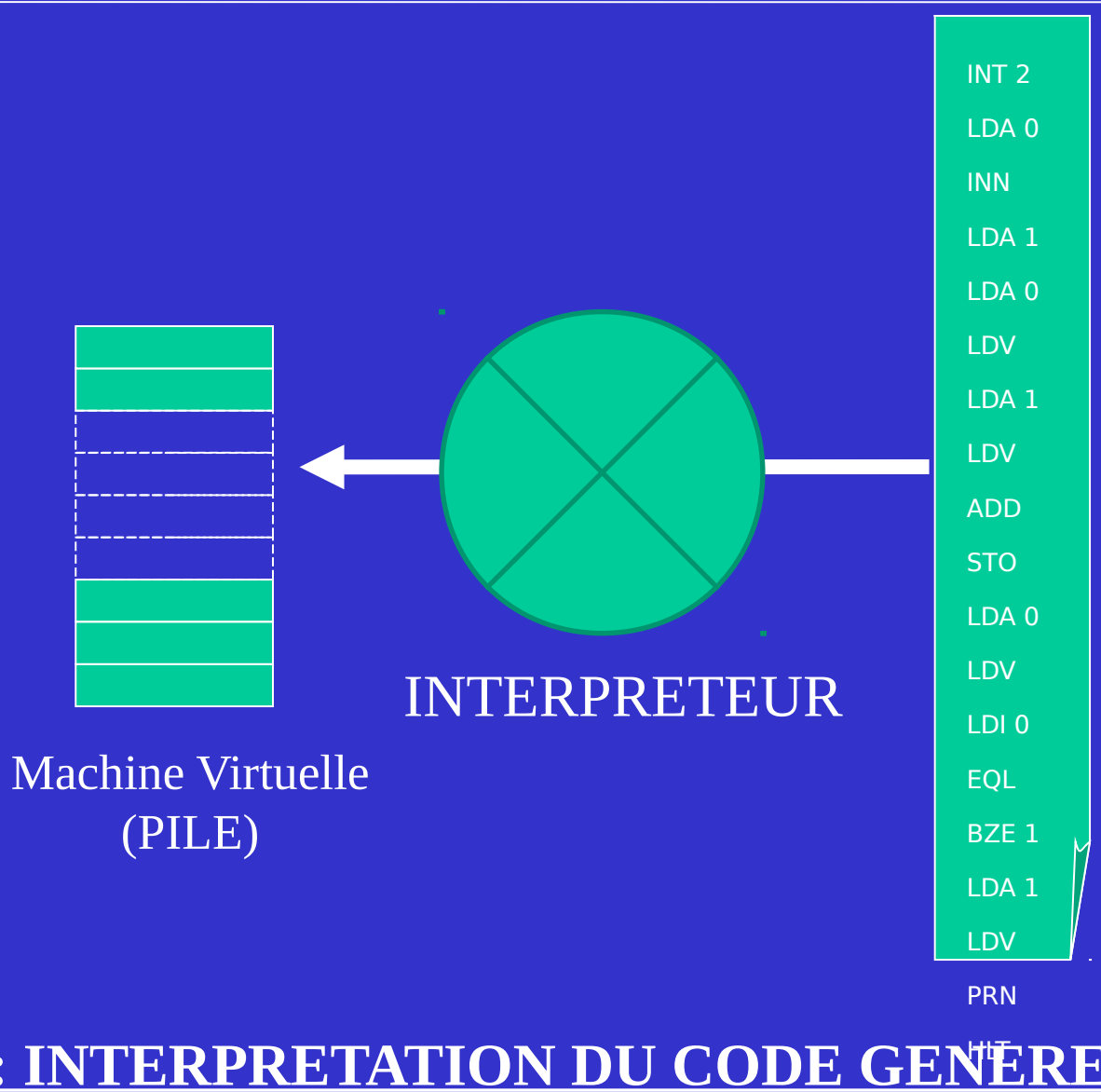
1. SAUVEGARDER LE CODE GENERE DANS UN FICHIER

2. DEFINIR UNE GRAMMAIRE QUI DECRIT LE CONTENU DU FICHIER

3. ECRIRE UN CHARGEUR (LOADER) DU PCODE D'UN FICHIER DANS LE TABLEAU PCODE

4. ECRIRE L'INTERPRETEUR

ARCHITECTURE DE L'INTERPRETEUR



2ième PARTIE: INTERPRETATION DU CODE GENERE

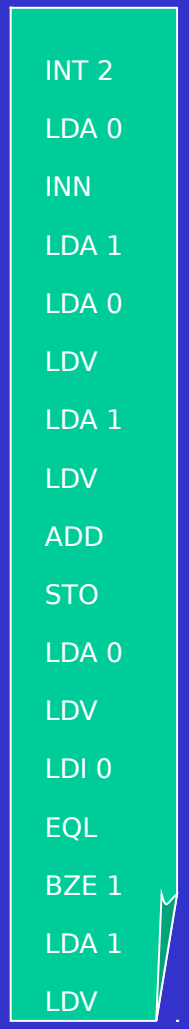
LA GRAMMAIRE

PCODE ::= **INT** NUM {INST_PCODE} **HLT**

INST_PCODE ::= **ADD** | **SUB**|**EQL**|…| [**LDA** | **BZE**|**BRN**|**LDI**] NUM

NUM ::= CHIFFRE {CHIFFRE}

CHIFFRE ::= **1**|..|**9**



LES DECLARATIONS

Les structures de données nécessaires lors de l'écriture d'un interprète simplifié pour le P-Code sont :

un tableau MEM représentant la pile de la machine et un pointeur de pile associé

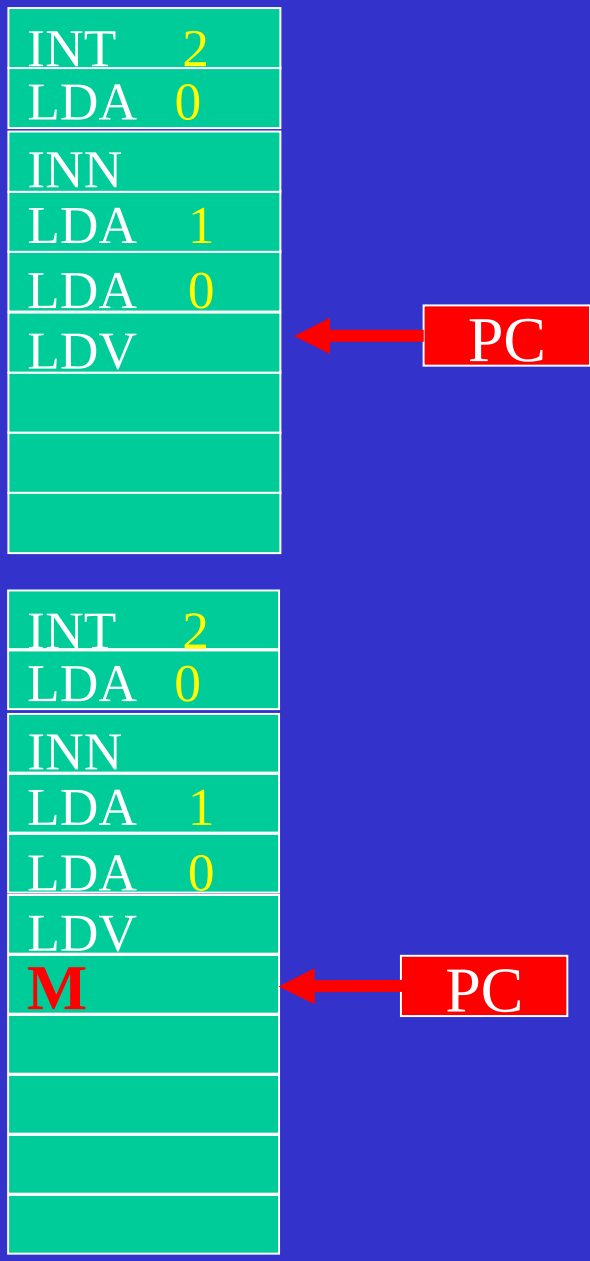
```
var
    PILE : TABLEAU [0 .. TAILLEMEM] DE ENTIER ;
    SP : ENTIER ;
```

```
Type MNEMONIQUE = (ADD,SUB,MUL,DIV,EQL,NEQ,GTR,
                   LSS,GEQ,LEQ, PRN,INN,INT,LDI,LDA,LDV,
                   STO,BRN,BZE,HLT, NUM) ;
```

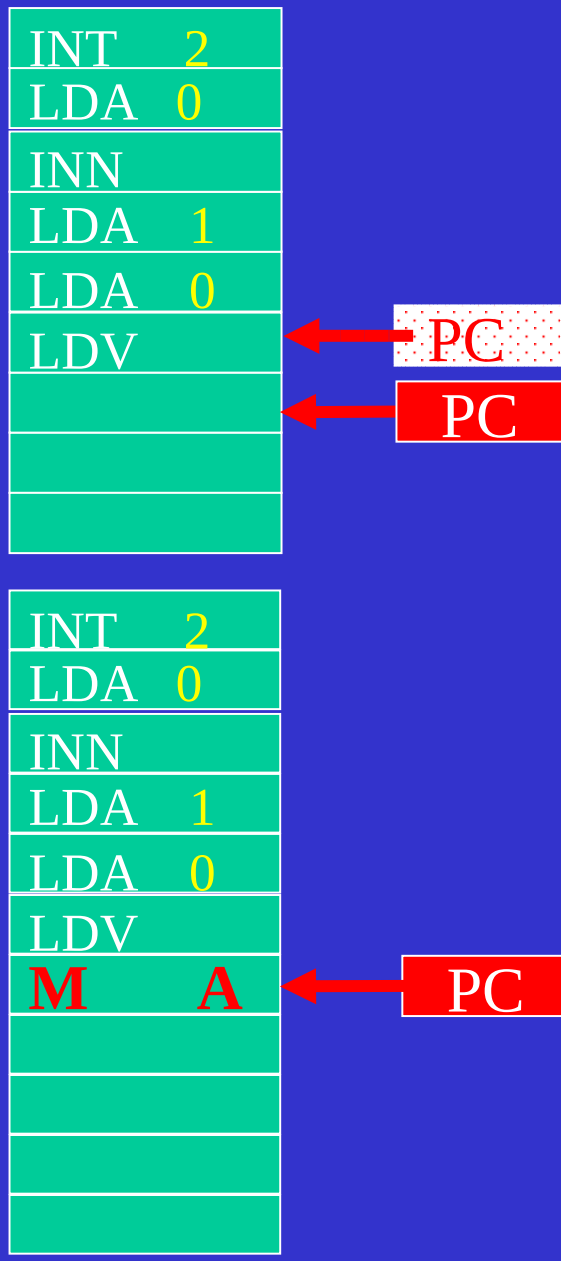
```
INSTRUCTION =      enregistrement
                       MNE : MNEMONIQUE ;
                       SUITE : entier
                   fin
VAR PCODE : tableau [0 .. TAILLECODE] de INSTRUCTION ;
        PC : entier ;  OFFSET=SP=-1; PC=-1;
```

Procédures de chargement de P-Code

```
procedure LOAD1 (M:MNEMONIQUES) ;
debut
      si  PC = TAILLECODE
            alors ERREUR ;
      PC := PC + 1 ;
      PCODE [PC]. MNE := M
fin ;
```



```
procedure LOAD2 (M:MNEMONIQUES ; A:entier) ;
debut
      si  PC = TAILLECODE
            alors ERREUR ;
      PC := PC + 1 ;
      PCODE [PC].MNE := M ;
      PCODE [PC].SUITE := A
fin;
```



INTERPRETATION DES INSTRUCTIONS MNEMONIQUES

jeu d'instruction du P-Code simplifié

ADD	additionne le sous-sommet de pile et le sommet, laisse le résultat au sommet (idem pour SUB, MUL, DIV)
EQL	laisse 1 au sommet de pile si sous-sommet = sommet, 0 sinon (idem pour NEQ, GTR, LSS, GEQ, LEQ)
PRN	imprime le sommet, dépile
INN	lit un entier, le stocke à l'adresse trouvée au sommet de pile, dépile
INT c	incrémente de la constante c le pointeur de pile (la constante c peut être négative)
LDI v	empile la valeur v
LDA a	empile l'adresse a
LDV	remplace le sommet par la valeur trouvée à l'adresse indiquée par le sommet (déréférence)
STO	stocke la valeur au sommet à l'adresse indiquée par le sous-sommet, dépile 2 fois
BRN i	branchement inconditionnel à l'instruction i
BZE i	branchement à l'instruction i si le sommet = 0, dépile
HLT	halte

```
void INTER_INST(INSTRUCTION INST){
        int val1, adr, val2;

        siwtch(INST.MNE){
                case INT:  OFFSET=SP=INST.SUITE;                    PC++;break;
                case LDI:  PILE[++SP]=INST.SUITE;                   PC++;break;
                case LDA: PILE[++SP]=INST.SUITE;                    PC++; break;
                case STO:  val1=MEM[SP--]; adr=PILE[SP--];PILE[adr]=val1;     PC++;break;
                case LDV:   adr=PILE[SP--]; PILE[++SP]=PILE[adr];           PC++;break;
                case EQL: val1=PILE[SP--];val2=PILE[SP--];
                          PILE[++SP]=(val1==val2);              PC++;break;
                case LEQ: val2=PILE[SP--];val1=PILE[SP--];
                          PILE[++SP]=(val1<=val2);              PC++;break;
                case BZE:  if (MEM[SP--]==0) PC=INST.SUITE;
                          else PC++;                    break;
                case BRN: PC=INST.SUITE;                        break;
                ....................................
                ....................................

        }

}
```

INTERPRETATION DES DE TOUT LE PCODE

```
void INTER_PCODE(){
    PC=0;
    while (PCODE[PC].MNE!=HLT)
                INTER_INST(PCODE[PC]);
}
```

FIN