Énoncé

Interpréter un code intermédiaire

Questions

- Qu'est-ce qu'un code intermédiaire ?

Reformulation de l'énoncé

Interpréter un code intermédiaire.
Le code intermédiaire ne comporte aucune instruction de lecture ou d'écriture.

À chaque étape de l'éxécution du programme, on affiche la valeur de CP (compteur de programme) et l'entièreté de la mémoire regroupant les valeurs des différentes variables manipulées par le programme.

```
Les instructions du langage intermédiaire sont représentées en mémoire par des quadruple

x y op z
GOTO Label
IF X Goto Label

Le premier élément du quadruplet x, GOTO ou bien IF permet d’identifier la nature de
l’instruction.
```

Raffinage

R0 :

```
Interpréter un code intermédiaire
```

R1 : Comment "interpréter un code intermédiaire" ?

```
Générer une représentation en mémoire du code intermédiaire
Intepréter le programme en mémoire
```

R2 : Comment "générer une représentation en mémoire du code intermédiaire" ?

R2 : Comment "intepréter le programme en mémoire" ?

```
Se placer au début du programme
Lire une instruction
Afficher le contenu de la mémoire en mode debug
Tant que l'on n'a pas atteint la fin du programme
    Interpréter l'instruction courante
    Afficher le contenu de la mémoire en mode debug
    Lire une instruction
```

R3 : Comment "se placer au début du programme" ?

```
ProgramCounter <- 1                (ProgramCounter : in ENTIER)
```

R3 : Comment "lire une instruction" ?

```
CurrentInstruction <- Programme[ProgramCounter]
(CurrentInstruction : out T_Instruction; Programme : in TABLEAU de T_Instruction; Progra
```

R3 : Comment "afficher le contenu de la mémoire en mode debug" ?

```
Si Debug = True (Debug : in BOOLEAN)
    Pour chaque variable de Variables
        Afficher l'identifiant de la variable et sa valeur
```

R3 : Comment savoir si "on a atteint la fin du programme" ?

```
CurrentInstruction[63..48] = 0xFFFF      (CurrentInstruction : out T_Instruction)
```

R3 : Comment "interpréter l'instruction courante" ?

```
Selon CurrentInstruction[63..61]        (CurrentInstruction : in T_Instruction)
    Cas 0b000  -- NULL
        Passer à l'instruction suivante
    Cas 0b100 -- ARITHM
        Appeler la fonction Arithmetic
        (CurrentInstruction : in T_Instruction; Memory : in out T_Array_Data)
        Passer à l'instruction suivante
    Cas 0b010 -- GOTO
        Aller à l'instruction indiqué
    Cas 0b011 -- IF_GOTO
        Si la condition du IF_GOTO est vraie
            Aller à l'instruction indiqué
        Sinon
            Passer à l'instruction suivante
    Cas 0b101 -- READ
        Lire la valeur de l'entrée standard et la stocker dans la variable pointée par C
    Cas 0b110 -- WRITE
        Afficher la valeur de la variable pointée par CurrentInstruction[47..32] selon T
    Default
        Lever l'exception Unknown_Instruction
```

R4 : Comment "Aller à l'instruction indiqué" ?

```
ProgramCounter <- ProgramCounter + CurrentInstruction[31..0]
(CurrentInstruction : in T_Instruction; ProgramCounter : in out ENTIER)
```

R4 : Comment faire la "condition du IF_GOTO est vraie" ?

```
valeur de la variable pointée par CurrentInstruction[47..32] != 0
(CurrentInstruction : in T_Instruction)
```

R4 : Comment faire la "fonction Arithmetic" ?

```
Op0_Address <- adresse de la variable pointée par CurrentInstruction[31..16]    (Op(
Op0_Value <- valeur de la variable pointée par Op0_Address      (Op0_Value : in out
Op1_Address <- adresse de la variable pointée par CurrentInstruction[15..0]    (Op1_
Op1_Value <- valeur de la variable pointée par Op1_Address      (Op1_Value : in out
Result_Address <- adresse de la variable pointée par CurrentInstruction[47..32]    (
Result_Value <- valeur de la variable pointée par Result_Address    (Result_Value :

<!--
Pas implémenté pour le moment :
Résoudre la vraie valeur de Op0_Value, Op1_Value et Result_Value si nécessaire (pour
-->

Selon CurrentInstruction[63..48]    (CurrentInstruction : in T_Instruction)
    Cas OP_CODE_ADD ( + )
        Result_Value <- Op0_Value + Op1_Value
    Cas OP_CODE_SUBSTRACT ( - )
        Result_Value <- Op0_Value - Op1_Value
    Cas OP_CODE_MULTIPLY ( * )
        Result_Value <- Op0_Value * Op1_Value
    Cas OP_CODE_DIVIDE ( / )
        Result_Value <- Op0_Value / Op1_Value
    Cas OP_CODE_LOWER_THAN ( < )
        Result_Value <- Op0_Value < Op1_Value
    cas OP_CODE_LOWER_OR_EQUAL_TO ( <= )
        Result_Value <- Op0_Value <= Op1_Value
    cas OP_CODE_GREATER_THAN ( > )
        Result_Value <- Op0_Value > Op1_Value
    cas OP_CODE_GREATER_OR_EQUAL_TO ( >= )
        Result_Value <- Op0_Value >= Op1_Value
    cas OP_CODE_EQUAL_TO ( = )
        Result_Value <- Op0_Value = Op1_Value
    cas OP_CODE_NOT_EQUAL_TO ( /= )
        Result_Value <- Op0_Value /= Op1_Value
    cas OP_CODE_LOGICAL_OR ( OR )
        Result_Value <- Op0_Value OR Op1_Value
    cas OP_CODE_LOGICAL_AND ( AND )
        Result_Value <- Op0_Value AND Op1_Value
    Défaut
        Lever une exception "Unknown operator"
```

```
    Passer à l’instruction suivante
```

R5 : Comment "Résoudre la vraie valeur de Op0_Value, Op1_Value et Result_Value si nécessaire" ?

```
Si CurrentInstruction[50] = 1
    Result_Value <- valeur de la variable pointée par Result_Value
Si CurrentInstruction[49] = 1
    Op0_Value <- valeur de la variable pointée par Op0_Value
Si CurrentInstruction[48] = 1
    Op1_Value <- valeur de la variable pointée par Op1_Value
```

R3 : Comment "passer à l'instruction suivante" ?

```
ProgramCounter <- ProgramCounter + 1    (ProgramCounter : in out ENTIER)
```