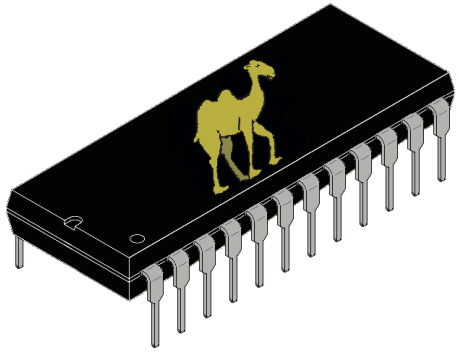
OCaPIC

Objective Caml pour programmer les microcontrôleurs PIC



Benoît Vaugon

encadré par

Emmanuel Chailloux & **Philippe Wang**

Plan

Présentation du projet

Présentation des PIC

La programmation sur PIC

Présentation d'Objective Caml

Pourquoi Objective Caml sur PIC

Présentation des PIC

Qu'est-ce ?

- Circuit intégré programmable
- Adapté aux applications embarquées

Domaines d'application

- Robotique industrielle
- Gadgets électroniques
- Domotique (appareils électroménagers)
- Cartes à puce

Caractéristiques du PIC18F4620

- Architecture : RISC 8 bits
- Horloge : jusqu'à 10 MIPS
- Mémoire programme : 64Ko
- Registres : $3968 \times 1o$
- Consommation : 2mA

La programmation sur PIC

Assembleur

- ```
  EXEMPLE_DE_CODE:
          tblrd*+
          addwf     FSR0L, W
          rcall     SOUS_PRGM
          btfsc     STATUS, C
          incf      FSR0H, F
          return
  ```

Compilation en natif

- Basic (BASIC84plus)
- C (sdcc, SIXPIC, MCC18, Hi-Tech C)
- Forth (PicForth)

Interprètes

- Basic (BasicStamps)
- Forth (FlashForth)

Utilisation d'une machine virtuelle

- PICOBIT (machine virtuelle Scheme pour PIC)
  ```

Présentation d'Objective Caml

Langage multi-paradigme

- Impératif, fonctionnel, objet

Fonctionnalités intéressantes

- Puissant système de modules
- Typage statique fort
- Gestion d'exceptions
- Polymorphisme paramétrique
- Évaluation paresseuse

Pourquoi OCaml sur PIC ?

Langage de haut-niveau riche

- Langage multi-paradigme

Machine virtuelle assez simple

- Assez peu d'instructions (146)
- Pas de vérification dynamique de type

Gestion fine des ressources

- Maîtrise de la représentation des données
- Gestion automatique de la mémoire

Réalisation

- Cycle de développement
- Chaîne de production
- Représentation des valeurs
- Gestion automatique de la mémoire
- Compression simple du code-octet
- Élimination de code mort
- Un simulateur de PIC

Cycle de développement

Étude de faisabilité

- Étude de la machine abstraite
- Choix du microcontrôleur
- Représentation des valeurs à l'exécution

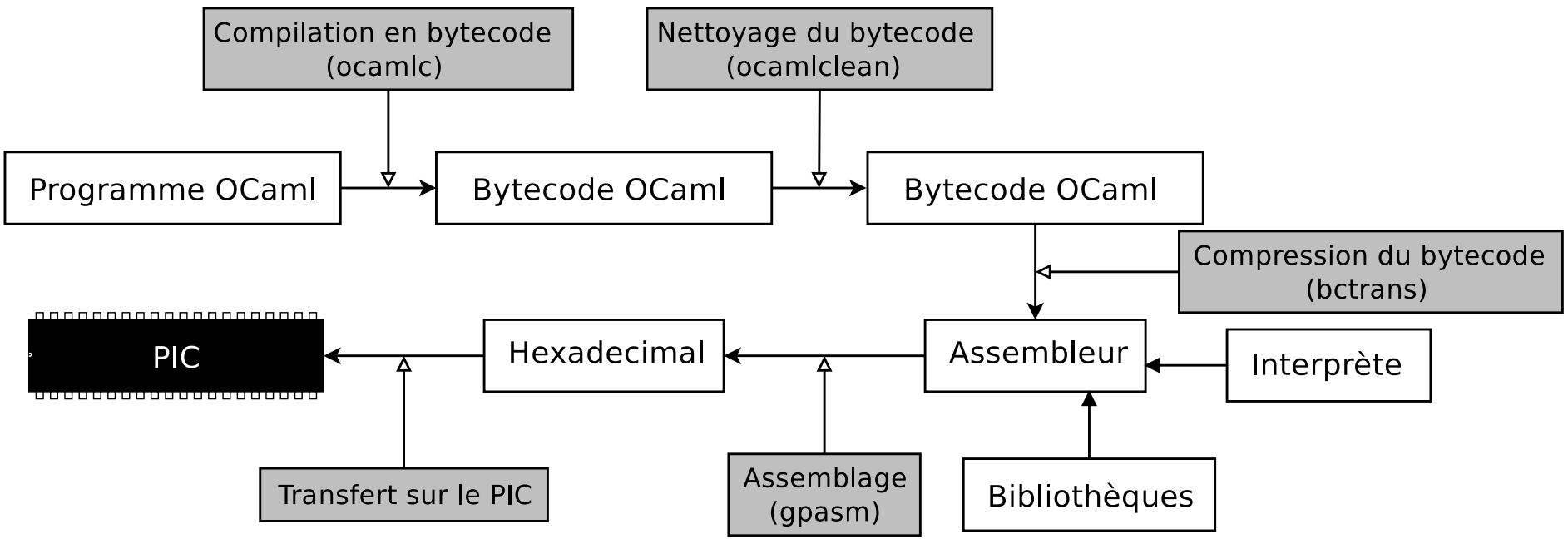
Implantation (total : 10 kloc)

- Interprète de code-octet (2,5 kloc asm)
- Bibliothèques (*runtime* (2 kloc asm) & *std* (0,5 kloc OCaml))
- Compression de code-octet (1,5 kloc OCaml)
- Nettoyeur de code-octet (1,8 kloc OCaml)
- Simulateur de PIC (0,9 kloc C + 0,8 kloc OCaml)

Tests (2 kloc C + OCaml + asm)

- Tests unitaires
- Réalisation d'un jeu de réflexion

Chaîne de production

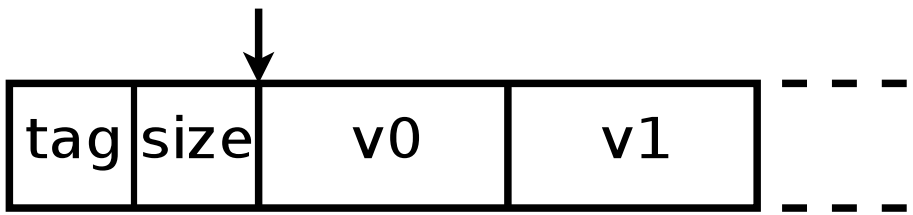


Représentation des valeurs

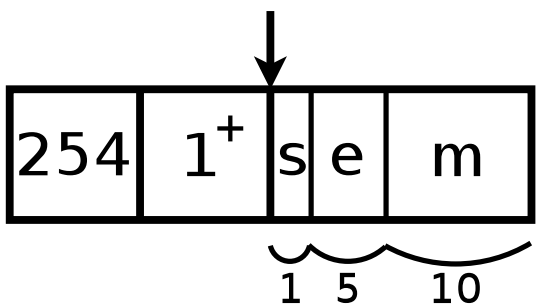
Représentation des entiers

- Codés sous la forme $2 \times n + 1$ (bit de tag)

Structure des blocs



Représentation des flottants



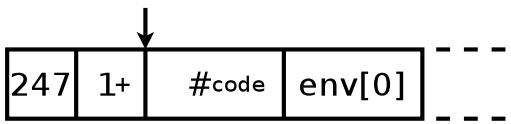
Gestion des fermetures

Qu'est-ce ?

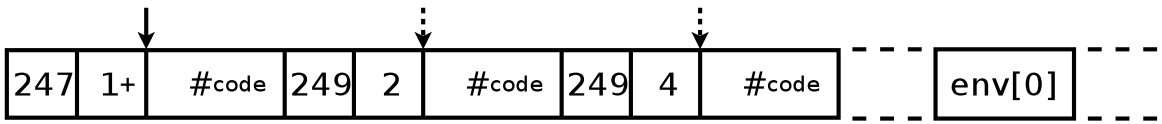
- Permet de représenter les fonctions
- Stocke l'environnement
- Stocke les arguments déjà passés en cas d'appel partiel

Deux types de fermetures

- Fermetures simples



- Fermetures récursives



Géré par 5 instructions de la machine virtuelle

- `apply` / `appterm`
- `return`
- `grab` / `restart`

Gestion automatique de la mémoire

Qu'est ce ?

- Libère la mémoire qui n'est plus utilisée
- Dans certains cas compacte la mémoire

Plusieurs types de ramasse miettes

- *Stop & Copy*
- *Mark & Sweep*
- *Mark & Compact*

Choix : *Stop & Copy*

- Deux phases :
  1. Parcours des racines
  2. Parcours du tas TO
- Optimise l'utilisation des registres
- N'améliore pas la localité spaciale

Compression simple du code-octet

Départ de l'exécutable généré par ocamlc

Compression du code-octet

- Compression instruction par instruction
- Pas de changement dynamique de la taille des arguments
- Pas de phase de décompression
- Reste facile à interpréter
- Facteur de compression : 3,5

Autres fonctionnalités

- Remapper les appels externes
- Lier avec l'interprète et la bibliothèque d'exécution
- Améliorer (précalculer) l'initialisation des variables globales
- Sauts absolus au lieu de relatifs

Élimination de code mort

Motivation

- Diminuer la taille du programme
- Diminuer l'occupation de la mémoire dynamique
- Augmenter la vitesse d'exécution
- Faire passer les objets OCaml

Algorithme

1. Repérage des blocs nettoyables
2. Analyse du flot de données
3. Analyse du flot de contrôle
4. Globalisation des fermetures

Limitations

- Arguments des foncteurs non nettoyés
- Chargement dynamique impossible

Un simulateur de PIC

Autres techniques

- Debuggage manuel par « `printf` » difficile voire impossible
- Simulateurs standards (gpsim, MPASM, PIC simulator IDE)
- Debuggers in-situ (MPASM)

Motivation

- Associer les actions du PIC au code source OCaml
- Adaptabilité vis-à-vis du circuit électronique
- Performances

Fonctionnement

- Bibliothèque d'exécution interceptant opérations de lecture/écriture
- Communication avec d'autres processus simulant les composants électroniques

Étude de performances

Vitesse d'exécution

Occupation mémoire

Vitesse d'exécution

Vitesse d'exécution moyenne

- 400 000 instructions VM par seconde
- 25 cycles machine par instruction VM

Instructions utiles / gestion du code-octet

- 7,5 cycles par instruction VM pour la gestion du code-octet
- 70% du code est utile

Comparaison avec PICOBIT

- 37000 instructions VM par seconde pour PICOBIT
- soit "11 fois moins" que pour OCaPIC
  (mais pour des code-octets différents)

Occupation mémoire

Mémoire programme

- Taille totale : 64Ko pour le PIC18F4620
- Interprète de code-octet : 4954o (7,6%)
- Bibliothèque d'exécution : 4254o (6,5%)
- Code-octet : jusqu'à 55Ko (86%)

Mémoire vive

- Proportion pile / tas ajustable
- Taille du tas par défaut : 1792o
- Taille de la pile par défaut : 174 niveaux

Exemple du jeu de gobelets

- Taille du code OCaml : 837
- Taille du code-octet (compressé) : 6570o
- Tas rempli aux 6/7 après l'initialisation
- Pile remplie parfois à 90%

Conclusion

Application fonctionnelle

- Distribuée en open-source (licence CeCILL-B)
  `http://www.algo-prog.info/ocaml_for_pic/`

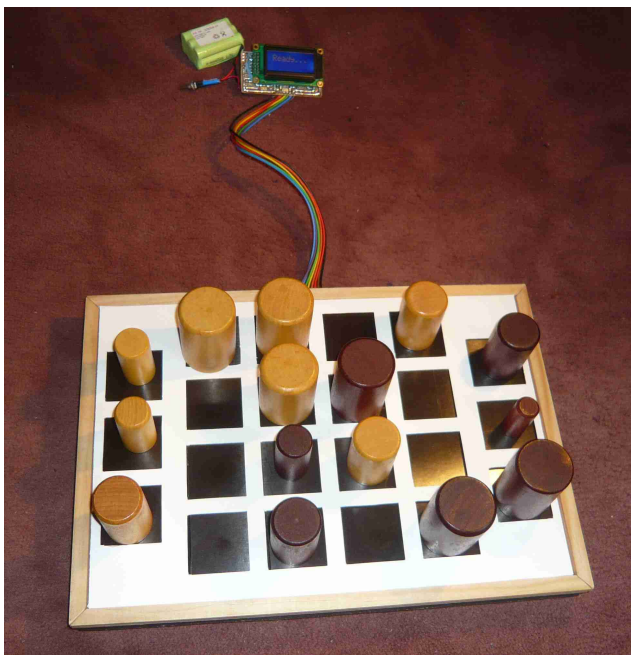
De bonnes performances

Perspectives

- Enrichir la bibliothèque standard
- Adapter à d'autres modèles de programmation
- Enrichir la bibliothèque spécifique aux PIC
- Porter sur d'autres architectures
- Écrire d'autres ramasse-miettes
- Gérer une RAM externe
- ...

Exemple d'application

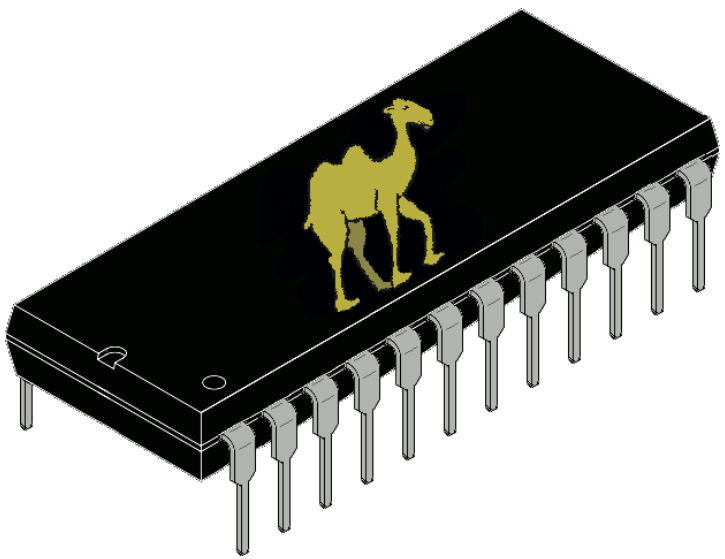
- Un jeu de réflexion se jouant à deux
- Programmation d'une IA sur PIC
- Entièrement codé, réalisé et testé



Questions

?

? ?



? ?

?

Bibliographie

Xavier Leroy, Damien Doligez, Jacques Garrigue, Didier Rémy, and Jérôme Vouillon.

The Objective Caml system (release 3.11) : Documentation and user's manual.

Institut National de Recherche en Informatique et en Automatique, 2008.

`http://caml.inria.fr/pub/docs/manual-ocaml/`.

Microchip.

PIC18F2525/2620/4525/4620 Data Sheet, 2008.

`http://ww1.microchip.com/downloads/en/devicedoc/39626b.pdf`.

Vincent St-Amour and Marc Feeley.

Picobit : A compact scheme system for microcontrollers.

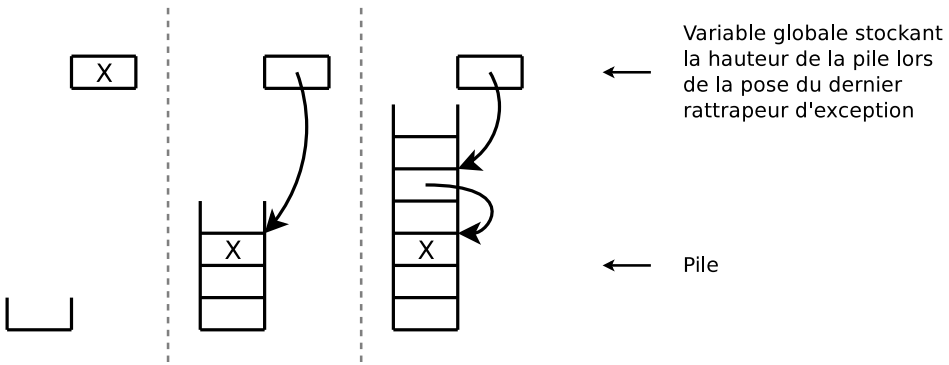
Implementation and Application of Functional Languages (IFL'09), pages 1–11, September 2009.

Gestion des exceptions

Plusieurs techniques

1. Rien lors de la pause des rattrapeurs, tout au lancement (exemple : Java)
2. Stockage des rattrapeurs dans la pile (exemple : OCaml)

Choix 2 : comme en OCaml



Gestion par trois instructions VM

- `pushtrap` / `poptrap`
- `raise`