Assembleur MIPS – Incrément 4

Notes de BE

Date de rendu livrable final :
cf. le site web
14/12/2018

Génération du code binaire des instruction

TEXT et DATA (big endian)

```
.text
  ADD $2, $3, $4
  boucle : ADDI $2, $3, boucle
  BEQ $2, $3, boucle
.data
  .word tab
.bss
  .space 0x10
  tab : .space 8
```

TEXT et DATA (big endian)

	31 26	25 21	20 16	15 11	10 6	5 0
R	opcode	rs	rt	rd	sa	func
	6	5	5	5	5	6
I	opcode	rs	rt	imm		
	6	5	5	16		
J	opcode	target				
	6	26				

Exemple : ADD $2, $3, $4 (au tableau)

⇒ Utilisation de champs de bits

TEXT et DATA (big endian)

Champs de bits

```
struct R {
    unsigned int func:6,
    sa:5,
    rd:5 ,
    rt:5,
    rs:5,
    opcode:6;
} ;
```

```
int main() {
   struct R add_inst ;
   add_inst.func = 0x0 ;
   add_inst.sa = 0x0 ;
   add_inst.rd = 0x2 ;
   add_inst.rt = 0x3 ;
   add_inst.rs = 0x4 ;
   add_inst.opcode = 0x20 ;
   Ecrire_dans_fichier(add_inst) ;
}
```

Remarque : Ajout des informations binaires dans le dictionnaire pour chaque instruction

Instructions

Également possible de faire du masquage et décalage.

Exemple :

```
unsigned int rd = 5;
unsigned int code_type_R = 0;
code_type_R = code_type_R | ((rd << 11) & 0x0000F800);
```

Les champs de bits sont plus lisibles

Comment savoir quel opérande correspond à quel champ ?

Opérandes des instructions pas toujours dans le même ordre

Exemple :

```
Add $1,$2,$3 # $1→ rd, $2 → rs $3 → rt
Div $1,$2      # $1→ rs, $2 → rt
MFHI $1        # $1→ rd
```

Encoder la syntaxe dans le dictionnaire

```
Add R 3 Reg Reg Reg rd rs rt
…
DIV R 2 Reg Reg rs rt
...
MFHI R 1 Reg  rd
```

Notez que l'information Reg devient redondante

Comment savoir quel valeur pour opcode, func etc ?

Dans le dictionnaire, soit en attribut-valeur

Add R 3 Reg Reg Reg rd rs rt opcode 0x0 func 0x20 sa 0x0

…

Soit en valeur initiale par défaut (en init les champs à zéro avant la génération binaire)

Add R 3 Reg Reg Reg rd rs rt 0x00000020

…

→ attention au swap

TEXT et DATA (big endian)

Little Indian vs Big Indian

① Instruction : 10000000 01000011 00100000 00000000

Ordre de lecture

Écriture dans le fichier par votre PC en little Endian sur 32 bits :

② Fichier :

00000000 00100000 01000011 10000000

Ordre de lecture en little endian

Lecture du fichier par la machine MIPS en big Endian sur 32 bits :

③ Lu par la machine MIPS :

00000000 00100000 01000011 10000000

Ordre de lecture en big endian

---

④ Résultat lu par la machine MIPS :

00000000 00100000 01000011 10000000

TEXT et DATA (big endian)

Swap sur 32 bits

Little Indian vs Big Indian

① Instruction : 00000000 00100000 01000011 10000000

Ordre de lecture

Écriture dans le fichier par votre PC en little Endian sur 32 bits :

② Fichier :

10000000 01000011 00100000 00000000

Ordre de lecture en little endian

Lecture du fichier par la machine MIPS en big Endian sur 32 bits :

③ Lu par la machine MIPS :

10000000 01000011 00100000 00000000

Ordre de lecture en big endian

---

④ Résultat lu par la machine MIPS :

10000000 01000011 00100000 00000000

TEXT et DATA (big endian)

Swap:

```
union inst_poly {
    struct R r_inst ;
    struct I i_inst ;
    struct J j_inst ;
    char code[4], ;
} ;
```

```
void swap_code(inst_poly inst) {
   swap(inst.code, inst.code+3) ;
   swap(inst.code+1, inst.code+2) ;
}
```

⇒ Exemple de swap au tableau

Remarque :

- Même processus pour les opérandes dans la section data.
- Attention aux chaînes de caractères

Swap de mot de 32 bits

Également possible de faire du masquage et décalage (rapide quand on sait ce que l'on fait).

Exemple :

code = 0xAABBCCDD;

code = ((code >> 24)& 0x000000FF) | ((code << 24)& 0xFF000000) | ((code >> 8)& 0x0000FF00) | ((code << 8)& 0x00FF0000);

Q: comment swapper les asciiz les .byte et les .space ?

Fichier ELF *Executable and Linkable Format* (cf. sujet Annexe B)

La bibliothèque ELF

Pour écrire un fichier ELF, utiliser la bibliothèque maison pelf.

cf. siteweb. La bibliothèque fournit :

- les structures de données section, symbole et reloc

- la fonction elf_write_relocatable qui permet d'écrire un elf à partir de toutes les sections

==> Exemple :

- make_mips_elf.c:270 (section),

- pelf.h:172 (elf_write_relocatable)

Nom, machine et nororder

name: Nom du fichier objet final

machine : Nom de la machine qui va lire le fichier (“mips”)

noreorder : Flag de compilation (pas de rearrangement de code)

==> Exemple : make_mips_elf.c:288

Sections importantes

```
En-têtes de section :
  [Nr] Nom              Type            Adr      Décala.Taille ES Fan LN Inf Al
  [ 0]                  NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text            PROGBITS        00000000 000040 000028 00  AX  0   0 16
  [ 2] .rel.text        REL             00000000 0001ac 000020 08   I  9   1  4
  [ 3] .data            PROGBITS        00000000 000070 000008 00  WA  0   0 16
  [ 4] .rel.data        REL             00000000 0001cc 000008 08   I  9   3  4
  [ 5] .bss             NOBITS          00000000 000080 000018 00  WA  0   0 16
  [ 6] .reginfo         MIPS_REGINFO    00000000 000080 000018 18   A  0   0  4
  [ 7] .MIPS.abiflags   MIPS_ABIFLAGS   00000000 000098 000018 18   A  0   0  8
  [ 8] .gnu.attributes  GNU_ATTRIBUTES  00000000 0000b0 000010 00      0   0  1
  [ 9] .symtab          SYMTAB          00000000 0000c0 0000c0 10     10  11  4
  [10] .strtab          STRTAB          00000000 000180 00002b 00      0   0  1
  [11] .shstrtab        STRTAB          00000000 0001d4 00005c 00      0   0  1
Clé des fanions :
  W (écriture), A (allocation), X (exécution), M (fusion), S (chaînes), I (info),
  L (ordre des liens), O (traitement supplémentaire par l'OS requis), G (groupe),
  T (TLS), C (compressé), x (inconnu), o (spécifique à l'OS), E (exclu),
  p (processor specific)
```

- shstrtab : table des noms de section
- text, data : données (PROGBITS)
- bss : espace à alloué au lancement du programme
- strtab : autres chaînes de caractères (symboles)
- Symtab : symboles
- rel.text, rel.data : sections de relocation

String table des sections (shstrtab)

```
Vidange hexadécimale de la section « .shstrtab » :
  0x00000000 002e7379 6d746162 002e7374 72746162 ..symtab..strtab
  0x00000010 002e7368 73747274 6162002e 72656c2e ..shstrtab..rel.
  0x00000020 74657874 002e7265 6c2e6461 7461002e text..rel.data..
  0x00000030 62737300 2e726567 696e666f 002e4d49 bss..reginfo..MI
  0x00000040 50532e61 6269666c 61677300 2e676e75 PS.abiflags..gnu
  0x00000050 2e617474 72696275 74657300          .attributes.
```

Ensemble des noms de section mise bout à bout

La table commence par une sentinelle ‘\0’ toutes les chaînes sont séparées par un ‘\0’ (==0)

→ dans le projet ces sections seront prédéfinies et constantes. L’exemple de code fourni sera suffisant.

==> Exemple : make_mips_elf.c:280

TEXT et DATA (big endian)

```
.text
   Lw $t0 , lunchtime
   LW $6, -200($7)
   ADDI $t1,$zero,8
boucle:
   BEQ $t0 , $t1 , byebye
   NOP
   addi $t1 , $t1 , 1
   J boucle
   NOP
byebye:
   JAL viteviteauru
```

```
Vidange hexadécimale de la section « .text » :
  0x00000000 3c080000 8d080000 8ce6ff38 20090008 <..........8 ...
  0x00000010 11090004 00000000 21290001 08000004 ........!).....
  0x00000020 00000000 0c000000                    ........
```

```
   .data
   lunchtime:
      .word 12
      .word menu
      .asciiz "ils disent : \"au ru!\""
```

```
Vidange hexadécimale de la section « .data » :
  0x00000000 0000000c 00000000 696c7320 64697365 ........ils dise
  0x00000010 6e74203a 20226175 20727521 2200     nt : "au ru!".
```

Swap de tous les mots de 32 bits

JAL = 0000000C → 0C000000 en big endian

.word 12 = 0C000000 → 0000000C en big endian

Octets et chaînes de caractères restent inchangés

⇒ Explication au tableau

TEXT et DATA (big endian)

```
.text
   Lw $t0 , lunchtime
   LW $6, -200($7)
   ADDI $t1,$zero,8
boucle:
   BEQ $t0 , $t1 , byebye
   NOP
   addi $t1 , $t1 , 1
   J boucle
   NOP
byebye:
   JAL viteviteauru

   .data
   lunchtime:
      .word 12
      .word menu
      .asciiz "ils disent : \"au ru!\""
```

```
Vidange hexadécimale de la section « .text » :
  0x00000000 3c080000 8d080000 8ce6ff38 20090008 <..........8 ...
  0x00000010 11090004 00000000 21290001 08000004 ........!).....
  0x00000020 00000000 0c000000                    ........
```

```
Vidange hexadécimale de la section « .data » :
  0x00000000 0000000c 00000000 696c7320 64697365 ........ils dise
  0x00000010 6e74203a 20226175 20727521 2200     nt : "au ru!".
```

⇒ Exemple : make_mips_elf.c:283,295

BSS

Utilisation de la fonction *make_bss_section* :

- La taille de la section correspond à la taille attribué dans les .space
- Représente une section avec que des 0

⇒ Exemple : make_mips_elf.c:285, 307

String table (strtab)

```
Vidange hexadécimale de la section « .strtab » :
  0x00000000 006c756e 63687469 6d650062 6f75636c .lunchtime.boucl
  0x00000010 65006279 65627965 006d656e 75007669 e.byebye.menu.vi
  0x00000020 74657669 74656175 727500             teviteauru.
```

Ensemble des chaînes de caractères des symboles mises bout à bout

Doivent impérativement être triées par ligne de définition (non définies à la fin)

La table commence par une sentinelle (‘\0’) toutes les chaînes sont séparées par un ‘\0’ (==0)

==> Exemple : make_mips_elf.c:287,315

Symtab → structure Elf32_Sym

```
Struct {
Elf32_Word      st_name ; /* Indice du nom du symbol dans la string table des symboles */
Elf32_Addr  st_value ;/* adresse relative du symbole dans sa section*/
Elf32_Word      st_size ;/*taille de l'objet visé, toujours 0, inutilisée*/
unsigned char   st_info ;/*Attributs sur le type et le binding du symbole*/
unsigned char   st_other ;/* Encode la visibilité -> toujours 0 */
Elf32_Half   st_shndx ;/* Indice de la section contenant le symbole */
}
```

Dans elf.h

```
#define ELF32_ST_BIND(i) ((i)>>4)   /* de info vers bind (STB_LOCAL ou STB_GLOBAL) */
#define ELF32_ST_TYPE(i) ((i)&0xf) /* de info vers type (STT_NOTYPE, STT_OBJECT,
STT_FUNC, STT_SECTION)*/
#define ELF32_ST_INFO(b,t) (((b)<<4)+((t)&0xf)) /* de (bind,type) vers info */
```

Liste de Symboles

```
La table de symboles « .symtab » :
Num:Valeur      Tail  Type      Lien      Vis          Ndx Nom
    0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
    1: 00000000     0 SECTION LOCAL  DEFAULT    1
    2: 00000000     0 SECTION LOCAL  DEFAULT    3
    3: 00000000     0 SECTION LOCAL  DEFAULT    5
    4: 00000000     0 NOTYPE  LOCAL  DEFAULT    3 lunchtime
    5: 00000010     0 NOTYPE  LOCAL  DEFAULT    1 boucle
    6: 00000024     0 NOTYPE  LOCAL  DEFAULT    1 byebye
    7: 00000000     0 NOTYPE  LOCAL  DEFAULT    5 menu
...
   11: 00000000     0 NOTYPE GLOBAL DEFAULT  UND viteviteauru
```

1) Démarre toujours par une sentinelle :
2) Puis les 3 sections text,data,bss
3) Puis les symboles définis triés par ordre de ligne de déclaration
4) Puis les symboles indéfinis/globaux

Symbole

- Attention, lors de la création des structures (Elf32_Sym)
- Lorsque le symbole indique une section (type STT_SECTION) il n'est pas nécessaire de saisir le champ st_name (laisser 0)
  → st_shndx est suffisant pour retrouver le nom du symbole
- Lorsque le symbole est de type NOTYPE, la valeur de st_name doit contenir l'indice du premier caractère de la chaîne dans strtab (p.ex., lunchtime est en 1 et boucle en 11)

⇒ Exemple : make_mips_elf.c:322

Elf32_Rel

```
Elf32_Rel{
Elf32_Addr      r_offset ; /*@ relative au début de la section*/
Elf32_Word      r_info; /*mode de calcul de la relocation*/
}
```

Dans elf.h

```
#define ELF32_R_SYM(i) ((i)>>8) /* info vers sym (numéro du symbole dans la table des symboles)*/
#define ELF32_R_TYPE(i) ((unsigned char)(i)) /* info vers type (R_MIPS32, etc.)*/
#define ELF32_R_INFO(s,t) (((s)<<8)+(unsigned char)(t)) /* (sym,type) vers info */
```

Sections de relocation (.rel.text, rel.data)

Section de réadressage '.rel.text':

Décalage	Info	Type	Val.-sym	Noms-symboles
00000000	00000205	R_MIPS_HI16	00000000	.data
00000004	00000206	R_MIPS_LO16	00000000	.data
0000001c	00000104	R_MIPS_26	00000000	.text
00000024	00000b04	R_MIPS_26	00000000	viteviteauru

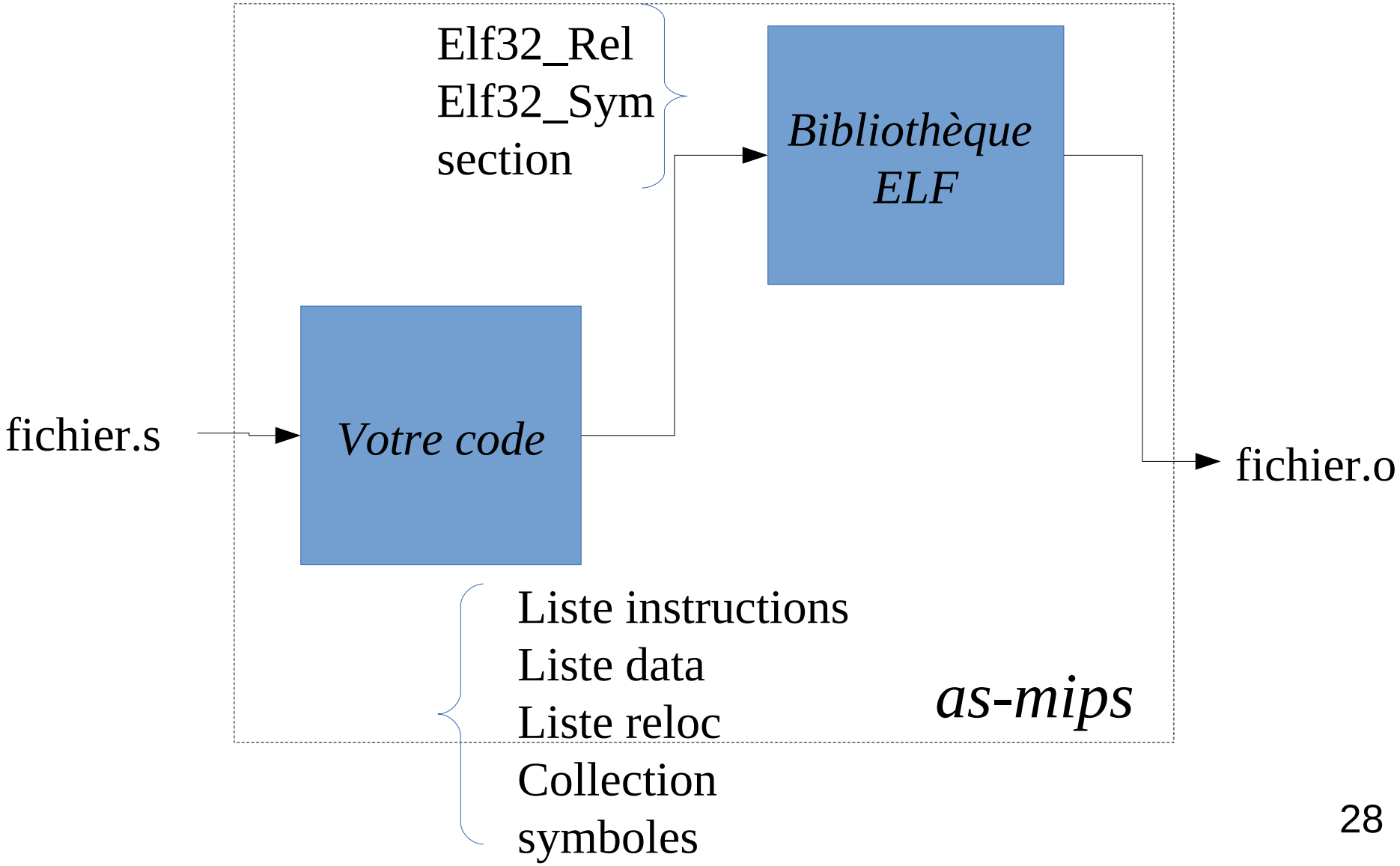
Section de réadressage '.rel.data' :

Décalage	Info	Type	Val.-sym	Noms-symboles
00000004	00000302	R_MIPS_32	00000000	.bss

- Toujours triée par adresse relative

⇒ Exemple : make_mips_elf.c:344

PELF : interface



Exemple d’utilisation de la bibliothèque PELF

- Cf. siteweb
- Compilation de la bibliotheque (lib dynamique)
- Make
- Compilation de l’exemple
- Make exemple_elf
- Execution
- ./exemple_elf → créé exemple.o

Tests automatiques

- Il faut :
  - fichier.s (avec entete : #TEST_RETURN_CODE = {PASS,FAIL,SKIP})
  - fichier.o.ref : sortie binaire attendue
  - votre exécutable (as-mips) qui produit un fichier.o à partir d'un fichier.s
- ./simpleUnitTest.sh -e as-mips fichier.s
- Exemple sur pelf :
  - ./simpleUnitTest.sh -e exemple_elf exemple.s

Tests automatiques

- Lorsqu'une erreur est détectée lors du traitement de l'assembleur (e.g., .word add $1$2$4) votre programme doit absolument sortir avec
  - exit(EXIT_FAILURE) (stdlib.h)
  - ou
  - ERROR_MSG (notify.h)
- Retourner 0 en cas de succès
  - Return 0
  - ou
  - exit(EXIT_SUCCESS) (stdlib.h)

Travail à faire

- Génération
  - s’assurer que les pseudo sw reg, symbole et lw reg, symbole sont gérées
  - Structure de données des instructions binaires
  - Fonctions de base de génération des instructions
  - Modification du dictionnaire
  - Génération des données (attention à l’alignement des .word !)
- ELF
  - Comprendre l’exemple
  - Adapter l’exemple au cas général et l’intégrer à votre code (copier le répertoire include, et la libpelf.so)
  - Ne JAMAIS modifier la bibliothèque PELF
  - Les fonctions make_ ???_section de make_mips_elf doivent être modifiées