

Syntaxe d'un fichier

Les commentaires commencent par ; et s'étendent jusqu'à la fin de la ligne.

Les sections disponibles sont les suivantes :

.text signale le début d'une section d'instructions assembleur.

.data signale le début d'une section de déclarations de données.

Chaque instruction ou déclaration peut être localisée par un *label* qui sera converti en une adresse par l'assembleur. Un label respecte la *regex* suivante : $[a-z, A-Z, _, 0-9]^+$.

On localise une instruction/déclaration à l'aide de la syntaxe :

```
label:
    instruction ou déclaration
```

On note que l'adressage de la mémoire se fait par mots de 32 bits (et pas moins!). Pour faire appel à des fonctions ou des sections définies dans un autre fichier on utilise la syntaxe suivante à n'importe quel endroit du fichier : **.include file**. Dans ce cas, le code du fichier référencé est inséré à cette position.

Les dossiers utilisés pour la recherche des fichiers à insérer est configurable via la ligne de commande.

Données

Chaines de caractères

Les chaines de caractères sont composées des caractères suivants :

- Les caractères ASCII affichables (codes 32 à 126 inclus).
- Le caractère nul noté \0 (code 0).

Les caractères " et les \ sont échappés : \" pour le guillemet et \\ pour la barre oblique inversée. La figure 1 expose comment les caractères sont codés sur des mots de 32 bits.

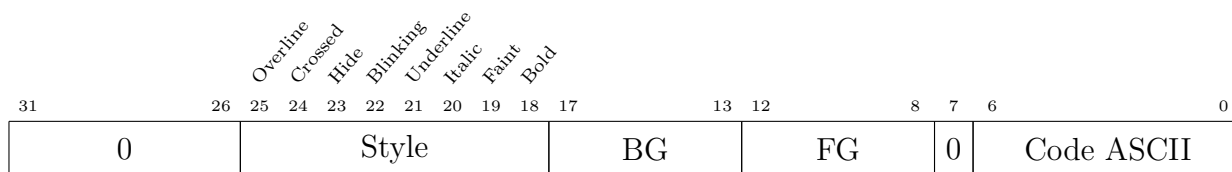


FIGURE 1 – Encodage d'un caractère.

Avec :

- Code ASCII : Le code ASCII du caractère représenté.
- FG : La couleur du texte comme décrit dans le tableau 1.
- BG : La couleur du derrière du texte comme décrit dans le tableau 1.








Identifiant	Nom	
0	Noir	
1	Rouge	
2	Vert	
3	Jaune	
4	Bleu	
5	Magenta	
6	Cyan	
7	Blanc	
8	Noir Clair	
9	Rouge Clair	
10	Vert Clair	
11	Jaune Clair	
12	Bleu Clair	
13	Magenta Clair	
14	Cyan Clair	
15	Blanc Clair	
16	Défaut	

TABLE 1 – Couleurs disponibles et leurs identifiants.

Le texte est par défaut donné entre guillemet : "**texte**". Les fonctions suivantes sont disponibles afin de spécifier les couleurs ainsi que le style du texte :

- `#textcolor(couleur, "texte")` pour spécifier la couleur du texte.
- `#backcolor(couleur, "texte")` pour spécifier la couleur du fond.
- `#bold("texte")`
- `#faint("texte")`
- `#italic("texte")`
- `#underline("texte")`
- `#blinking("texte")`
- `#hide("texte")`
- `#crossed("texte")`
- `#overlined("texte")`
- `#default("texte")`

Le symbole + pourra être utilisé afin de concaténer du texte. Par exemple :

```
#bold(#textcolor(red, "Hello") + " " + #italic(#textcolor("green", "World")))
```

↓

Hello World

Entiers

Les entiers peuvent être donnés dans plusieurs bases :

- En base décimale, base par défaut.
- En base binaire, lorsque préfixés par 0b.
- En base hexadécimale, lorsque préfixés par 0x.

Déclaration des données

On utilise les notations suivantes pour déclarer des données :

- .string** *text* déclare une chaîne de caractère. La chaîne est éventuellement stylisée. Cette chaîne n'est pas automatiquement terminée par `\0`.
- .zstring** *text* déclare une chaîne de caractère terminée par `\0`. La chaîne est éventuellement stylisée sauf le zéro final.
- .int** déclare un entier signé sur 32 bits signé ou non.

Instructions

Registres

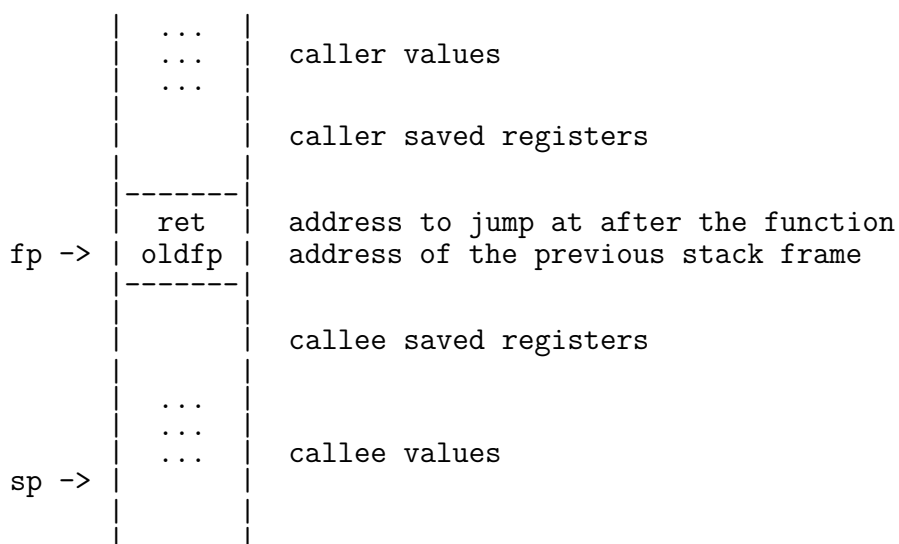
L'assembleur possède 31 registres de travail : de `r0` à `r28`, `rout`, `sp`, `fp`. Les registres suivants ont un sens particulier :

- r0** n'est pas modifiable et a comme valeur 0.
- r1** n'est pas modifiable et a comme valeur 1.
- rout** est supposé être utilisé afin de stocker la valeur de retour des fonctions.
- sp** a comme valeur la prochaine adresse libre du tas.
- fp** a comme valeur l'adresse du tableau d'activation de la fonction en cours.

Le registre 31 est réservé à l'assembleur/compilateur. Avertissement dans l'assembleur si utilisé. Nom : `rpriv`

Convention d'appel

La pile est organisée de cette manière lors d'un appel de fonction :



Les 9 premiers arguments d'une fonction sont donnés dans les registres `r20` à `r28`. Les autres sont passés sur la pile de la droite vers la gauche. C'est la responsabilité de l'appelleur de nettoyer les arguments sur la pile après l'appel.

La valeur de retour de la fonction est passé dans le registre `rout`. Dans le cas d'une structure ou de donnée trop grande, le registre `rout` contient une adresse mémoire vers ces données.

Les registres de **r15** à **r28** sont dits *caller-saved* (volatile) : une fonction est susceptible d'écraser la valeur de ces registres sans les restaurer. Les registres de **r0** à **r14** sont dits *callee-saved* (non-volatile) : lors de l'appel à une fonction, cette dernière ne doit pas modifier ces registres.

Labels

Les *labels* ne peuvent être déclarés qu'une seule fois.

Format des Sauts

Lors de saut les formats d'adresse suivants sont autorisés :

- Saut à une adresse absolue. L'adresse est donnée sans signe dans n'importe quelle base. Exemple : `jmp 12` saute à l'adresse 12 du programme.
- Saut à une adresse relative. L'adresse est donnée avec un signe (+ ou -) dans n'importe quelle base. Exemple : `jmp +12` saute 12 instruction après, `jmp -12` saute 12 instruction avant.
- Saut à un *label*. L'adresse absolue est calculée par l'assembleur. Exemple : `jmp lbl` saute au label *lbl* du programme.

Instructions

Instructions de Calculs Logiques

Instruction	Destination	Arguments		Description
<code>and</code>	r_1	r_2	r_3	Calcule $r_2 \wedge r_3$ dans r_1 .
<code>or</code>	r_1	r_2	r_3	Calcule $r_2 \vee r_3$ dans r_1 .
<code>nor</code>	r_1	r_2	r_3	Calcule $\neg(r_2 \vee r_3)$ dans r_1 .
<code>xor</code>	r_1	r_2	r_3	Calcule $r_2 \oplus r_3$ dans r_1 .
<code>not</code>	r_1	r_2		Calcule $\neg r_2$ dans r_1 .

Instructions de Calculs Arithmétiques

Instruction	Destination	Arguments		Description
<code>add</code>	r_1	r_2	r_3	Calcule $r_2 + r_3$ dans r_1 .
<code>sub</code>	r_1	r_2	r_3	Calcule $r_2 - r_3$ dans r_1 .
<code>mul</code>	r_1	r_2	r_3	Calcule $r_2 \times r_3$ dans r_1 .
<code>div</code>	r_1	r_2	r_3	Calcule $r_2 \div r_3$ dans r_1 .
<code>neg</code>	r_1	r_2		Calcule $-r_2$ dans r_1 .
<code>inc</code>	r_1	r_2		Calcule $r_2 + 1$ dans r_1 .
<code>dec</code>	r_1	r_2		Calcule $r_2 - 1$ dans r_1 .

Instructions de Décalages

Les décalages sont opérés modulo 32 bits.

Instruction	Destination	Arguments		Description
<code>asr</code>	r_1	r_2	r_3	Décale r_2 vers la droite de r_3 bits en dupliquant le bit de signe dans r_1 .
<code>lsr</code>	r_1	r_2	r_3	Décale r_2 vers la droite de r_3 bits en ajoutant les 0 nécessaires dans r_1 .
<code>lsl</code>	r_1	r_2	r_3	Décale r_2 vers la gauche de r_3 bits en ajoutant les 0 nécessaires dans r_1 .

Instructions pour la manipulation du Tas

Instruction	Destination	Argument	Description
<code>push</code>		r_1	Empile r_1 sur le tas.
<code>pop</code>	r_1		Dépile le sommet du tas dans r_1 .

Instructions pour manipuler la Mémoire

Instruction	Destination	Arguments		Description
<code>mov</code>	r_1	r_2		Copie le registre r_2 dans r_1 .
<code>store</code>	r_1	r_2		Copie le registre r_2 à l'adresse mémoire de r_1 .
<code>load</code>	r_1	r_2		Copie la valeur de la mémoire à l'adresse r_2 dans r_1 .
<code>loadi</code>	r_1	i		Copie i (entier signé ou non sur 32 bits) dans r_1 .
<code>loadi</code>	r_1	$\$l$		Copie l'adresse associée à l dans r_1 .
<code>loadi</code>	r_1	i	r_2	Copie $i + r_2$ dans r_1 avec i un entier signé ou non sur 32 bits.
<code>loadi</code>	r_1	$\$l$	r_2	Copie l'adresse associée à l plus r_2 dans r_1 .

Instructions de Branchement

Instruction	Destination	Argument	Description
<code>test</code>		r_1	Remplit les flags \mathbf{N}^1 et \mathbf{Z}^2 à partir de la valeur de r_1 .
<code>jmp</code>		r_1	Continue l'exécution du programme à l'adresse r_1 .
<code>jmp</code>		$\$l$	Continue l'exécution du programme à l'adresse associée au label l .
<code>jmp</code>		i	Continue l'exécution du programme à l'adresse i (entier codé sur 16 bits).
<code>jmp</code>		$+i$	Continue l'exécution du programme i adresses (entier codé sur 16 bits) plus tard.
<code>jmp</code>		$-i$	Continue l'exécution du programme i adresses (entier codé sur 16 bits) avant.
<code>jmp.F</code>		r_1	Continue l'exécution du programme à l'adresse r_1 lorsque le <i>flag</i> F est activé.
<code>jmp.F</code>		$\$l$	Continue l'exécution du programme à l'adresse associée au label l lorsque le <i>flag</i> F est activé.
<code>jmp.F</code>		i	Continue l'exécution du programme à l'adresse i (entier codé sur 16 bits) lorsque le <i>flag</i> F est activé.
<code>jmp.F</code>		$+i$	Continue l'exécution du programme i adresses (entier codé sur 16 bits) plus tard lorsque le <i>flag</i> F est activé.
<code>jmp.F</code>		$-i$	Continue l'exécution du programme i adresses (entier codé sur 16 bits) avant lorsque le <i>flag</i> F est activé.
<code>halt</code>			Saute à l'adresse $2^{32} - 1$. Les valeurs des registres r_3 et r_4 sont écrasées.

Instructions pour les Fonctions

Instruction	Destination	Argument	Description
<code>call</code>		$\$l$	Appelle la fonction l .
<code>call</code>		r_1	Appelle la fonction à l'adresse r_1 du programme.
<code>ret</code>			Termine la fonction courante et retourne à l'exécution du programme.

1. *Flag* négatif
2. *Flag* zéro