# Zorglub33 – Aide mémoire

## Registres

A registre général
B registre général
PC compteur ordinal / *program counter*
SP pointeur de pile / *stack pointer*
SR registre d'état / *status register*

Le registre d'état comporte les bits suivants :

| | ... | 9 | 8 | ... | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| SR | ... | S | IE | ... | O | N | Z | C |

Bits S et IE : accessibles en mode superviseur
S (*supervisor*) 1 : mode superviseur, 0 : mode utilisateur
IE (*interrupt enable*) 1 : interruptions activées, 0 : masquées

Bits O, N, Z, C : accessibles en mode utilisateur ou superviseur
Valeur = fonction de la dernière opération `cmp` ou arithmétique
O (*overflow*) dépassement de capacité
N (*negative*) résultat négatif
Z (*zero*) résultat nul
C (*carry*) l'opération a provoqué une retenue

## Interruptions et exceptions

En cas d'interruption ou d'exception, le processeur :
- sauvegarde le registre PC à l'adresse 100
- sauvegarde le registre SR à l'adresse 101
- sauvegarde le code de l'événement à l'adresse 102
- met le bit SR:S à 1
- met le bit SR:IE à 0 s'il s'agit d'une interruption
- met la valeur 200 dans PC

Codes d'événements :

| | |
|---|---|
| 0 | interruption matérielle |
| 1 | division par zéro |
| 2 | tentative d'exécution d'une instruction invalide |
| 3 | tentative d'exécution d'une instruction privilégiée |
| 4 | exécution de l'instruction `trap` |
| 5 | accès mémoire invalide |

Lorsque le processeur exécute l'instruction `rti`, il :
- restaure le registre PC à partir de l'adresse 100
- restaure le registre SR à partir de l'adresse 101

## Contrôleur de clavier

Registre d'état (R/-)

| | ... | 2 | 1 | 0 |
|---|---|---|---|---|
| état (20) | ... | - | I | R |

I (*interrupt*) 1 : le contrôleur a généré une interruption
R (*ready*) 1 : donnée prête à être lue
La lecture du registre d'état provoque la remise à 0 du bit I.

Registre de contrôle (-/W)

| | ... | 1 | 0 |
|---|---|---|---|
| contrôle (20) | ... | I | L |

I (*interrupt*) 1 : génère une interrupt. si donnée disponible
L (*LED*) 1 : allumer ou éteindre une LED du clavier

Registre de donnée (R/-)

| | ... | 9 | 8 | 7 ... 4 | 3 ... 0 |
|---|---|---|---|---|---|
| donnée (21) | ... | C | S | col | lig |

C (*control*) 1 : touche Ctrl enfoncée
S (*shift*) 1 : touche Shift enfoncée
col (*colonne*) numéro de colonne de la touche enfoncée
lig (*ligne*) numéro de ligne de la touche enfoncée

Registre de donnée (-/W)

| | ... | 3 | 2 ... 0 |
|---|---|---|---|
| donnée (21) | ... | S | led |

S (*switch*) 1 : allumer la LED, 0 : éteindre
led (*numéro*) numéro de la LED à allumer ou éteindre

## Contrôleur de disque

Registre d'état (R/-)

| | ... | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|
| état (50) | ... | E | I | R | A |

E (*error*) 1 : une erreur a été détectée
I (*interrupt*) 1 : le contrôleur a généré une interruption
R (*ready*) 1 : donnée prête à être lue
A (*available*) 1 : disque inactif
La lecture du registre d'état provoque la remise à 0 des bits I et E.

Registre de contrôle (-/W)

| | ... | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|
| contrôle (50) | ... | I | L | R | W |

I (*interrupt*) 1 : génère une interruption après transfert
L (*location*) 1 : registres de données contiennent C/H/S
R (*read*) 1 : lance une requête de lecture
W (*write*) 1 : lance une requête d'écriture

Registres de données (R/W)

| | 7 | ... | 0 |
|---|---|---|---|
| données (51 à 100) | octet | | |

octet emplacement (C/H/S), ou données lues ou à écrire

## Zorglub33-M

Par rapport au Zorglub33, le Zorglub33-M dispose de deux registres LIM et BASE supplémentaires :

- mode superviseur : registres non utilisés
- mode utilisateur : accès mémoire à une adresse $\lambda$ :
  - si $\lambda \in [0, \texttt{\%lim}[$, alors
    - convertir l'adresse : $\varphi = \lambda + \texttt{\%base}$
  - sinon générer l'exception « accès mémoire invalide »

## Zorglub33-S

Par rapport au Zorglub33, le Zorglub33-S ajoute 10 couples de registres <LIM$_s$, BASE$_s$> ($0 \leq s < 10$) :

- mode superviseur : registres non utilisés
- mode utilisateur : accès mémoire à une adresse $\lambda$ :
  - segment $s = \lfloor \frac{\lambda}{1000} \rfloor$, offset $o = \lambda \bmod 1000$
  - si $o \in [0, \texttt{\%lim}_s[$, alors
    - convertir l'adresse : $\varphi = \lambda + \texttt{\%base}_s$
  - sinon générer l'exception « accès mémoire invalide »

## Zorglub33-V

Par rapport au Zorglub33, le Zorglub33-V ajoute un registre PT (*page table*) indiquant l'adresse de la table des pages, utilisée pour tout accès (en mode superviseur ou utilisateur) :

Chaque entrée de la table des pages a le format suivant :

| | ... | 11 | 10 | 9 | 8 | 8 | 7 | 6 | ... | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| | ... | P | R | X | S | A | D | adrcadre | | |

Les flags de chaque entrée sont :

P (*present*) 1 : page présente, 0 : page absente
R (*read-only*) 1 : lecture seule, 0 : lecture/écriture
X (*execute*) 1 : exécution possible, 0 : pas d'exécution
S (*supervisor*) 1 : accès en mode superviseur seulement
A (*accessed*) 1 : page accédée, 0 : page non accédée
D (*dirty*) 1 : page modifiée : 0 : page non modifiée

Lors d'un accès mémoire (en mode U ou S) à l'adresse $\lambda$ :
- numéro de page $p = \lfloor \frac{\lambda}{100} \rfloor$, offset $o = \lambda \bmod 100$
- lecture de la case mémoire d'adresse $\texttt{\%pt} + p$
- si flags {P, R, X, S} ok, alors
  - convertir l'adresse : $\varphi = \text{adrcadre} \times 100 + o$
  - actualiser les flags {A, D} si nécessaire
- sinon générer l'exception « accès mémoire invalide »

# Zorglub33 – Aide mémoire

## Modes d'adressage

*imm* Immédiat — `ld 100,%a`
valeur indiquée

*reg* Registre — `ld %b,%a`
contenu du registre

*dir* Direct — `ld [100],%a`
adresse fournie

*ind* Indirect — `ld [%b],%a`
adresse pointée par le registre

*idx* Indirect indexé — `ld [%b-5],%a`
adresse pointée par registre +/- déplacement

**Exemple : instruction `ld`**

Syntaxe : `ld` *imm/reg/dir/ind/idx*`,` *reg*

| | |
|---|---|
| `ld 100,%a` | a ← valeur 100 |
| `ld %b,%a` | a ← contenu du registre b |
| `ld [100],%a` | a ← contenu de la mémoire à l'adresse 100 |
| `ld [%b],%a` | a ← c. de la mém à l'adresse indiquée par b |
| `ld [%b+3],%a` | a ← c. de la mém à l'adresse indiquée par b+3 |

**Exemple : instruction `st`**

Syntaxe : `st` *reg*`,` *dir/ind/idx*

| | |
|---|---|
| `st %a,[100]` | mémoire à l'adresse 100 ← a |
| `st %a,[%b]` | mémoire à l'adresse indiquée par b ← a |
| `st %a,[%b-5]` | mémoire à l'adresse indiquée par b-5 ← a |

## Préprocesseur

`#define` *symbole val* — `#define MAX 0x50`
définit le symbole à la valeur indiquée

`#include <` *fichier* `>` — `#include <defs.h>`
inclut les définitions du fichier

`#if` *expression* / `#else` / `#endif` — `#if defined(MAX)`
traitement conditionnel des instructions suivantes

## Directives assembleur

*symbole*`:` — `debut:`
associe l'adresse actuelle au symbole

`.addr` *val* — `.addr 200`
modifie l'adresse actuelle

`.space` *val* — `.space 4`
réserve *val* octets à l'adresse actuelle

`.word` *val* — `.word 0x1000`
réserve un mot à l'adresse actuelle et y place *val*

`.string` *chaîne* — `.string "hello"`
place la chaîne (avec un octet nul) à l'adresse actuelle

## Instructions

`add` *imm/reg/dir/ind/idx*`,` *reg* — `add 5,%a`
ajoute une valeur à un registre

`and` *imm/reg/dir/ind/idx*`,` *reg* — `and 0xf8,%a`
*et* bit à bit avec la valeur indiquée

`call` *imm/reg/dir/ind/idx* — `call 8399`
empiler `%pc` et aller à l'adresse indiquée

`cmp` *imm/reg/dir/ind/idx*`,` *reg* — `cmp 0,%a`
compare une valeur à un registre (résultat = bits de SR)

`div` *imm/reg/dir/ind/idx*`,` *reg* — `div [100],%a`
divise un registre par une valeur

`fas` *dir/ind/idx*`,` *reg* — `fas [2500],%a`
charge le contenu d'une case mémoire dans un registre et met cette case à 1

`in` *dir/ind/idx*`,` *reg* — `in [50],%b`
lit une valeur depuis un contrôleur d'entrée/sortie

`jmp` *imm/reg/dir/ind/idx* — `jmp %b`
saut inconditionnel

`jxx` *imm/reg/dir/ind/idx* — `jeq 3700`
saut conditionnel, avec *xx* en fonction des bits de SR :
- `jeq` : saut si égal
- `jne` : saut si différent
- `jle` : saut si inférieur ou égal
- `jlt` : saut si inférieur strict
- `jge` : saut si supérieur ou égal
- `jgt` : saut si supérieur strict

`ld` *imm/reg/dir/ind/idx*`,` *reg* — `ld [100],%a`
charge un registre avec une valeur

`mul` *imm/reg/dir/ind/idx*`,` *reg* — `mul %b,%a`
multiplie un registre par une valeur

`neg` *reg* — `neg %a`
négation d'un registre

`nop` — `nop`
instruction n'effectuant aucune opération

`not` *reg* — `not %a`
*non* bit à bit du registre

`or` *imm/reg/dir/ind/idx*`,` *reg* — `or %a,%b`
*ou* bit à bit avec la valeur indiquée

`out` *imm/reg*`,` *dir/ind/idx* — `out 0x80,[50]`
écrit une valeur vers un contrôleur d'entrée/sortie

`pop` *reg* — `pop %b`
dépiler dans un registre et incrémenter `%sp`

`push` *imm/reg* — `push 5`
décrémenter `%sp` et placer la valeur au sommet de la pile

`reset` — `reset`
réinitialise le processeur

`rti` — `rti`
retour d'interruption ou d'exception

`rtn` — `rtn`
retour d'appel provoqué par `call`

`shl` *imm/reg/dir/ind/idx*`,` *reg* — `shl 2,%a`
décalage à gauche des bits du registre

`shr` *imm/reg/dir/ind/idx*`,` *reg* — `shr %b,%a`
décalage à droite des bits du registre

`st` *reg*`,` *dir/ind/idx* — `st %a,[%b-4]`
stocke le contenu d'un registre en mémoire

`sub` *imm/reg/dir/ind/idx*`,` *reg* — `sub 0x20,%a`
soustrait une valeur à un registre

`swap` *reg/dir/ind/idx*`,` *reg* — `swap [100],%a`
permute une valeur et un registre

`trap` — `trap`
provoque une exception de type *trappe*

`xor` *imm/reg/dir/ind/idx*`,` *reg* — `xor [100],%b`
*ou exclusif* bit à bit avec la valeur indiquée

## Exemple : factorielle

```
// calcul de n! (n passé en argument)
factorielle:
    ld   [%sp+1],%a
    cmp  1,%a
    jge  casparticulier  // saut si 1 ≥ n
// cas général
    sub  1,%a            // a ← n-1
    push %a
    call factorielle     // a ← (n-1)!
    add  1,%sp           // dépile l'argument n-1
    push %b              // sauvegarder b
    ld   [%sp+2],%b      // b ← n (argument original)
    mul  %b,%a           // a ← n * (n-1)!
    pop  %b              // restaurer b
    rtn
casparticulier:
    ld   1,%a
    rtn
```

## Exemple : hello, world

```
#define NBYTES  14       // nb d'octets dans la chaîne
#define P_WRITE 7        // numéro de la primitive write
#define P_EXIT  2        // numéro de la primitive exit
chaine:                  // valeur du symbole = adr de la chaîne
    .string "hello, world!\n"
main:
    push NBYTES          // nombre d'octets à transférer
    push chaine          // empiler l'adresse de la chaîne
    push 1               // sortie standard
    ld   P_WRITE,%a      // numéro de la primitive
    trap                 // appel à write
    add  3,%sp           // dépiler les arguments
    ld   P_EXIT,%a
    trap                 // appel à exit
    rtn                  // on ne devrait jamais arriver ici
```