# M1 SESI 2017-2018
# Architecture Multi-Processeurs
# TP2 : Architecture Interne du Contrôleur de Cache

Kevin Mambu

March 1, 2018

## Question C1

L'adresse de la première instruction du main est 0x00400000. La quatrième instruction du main est l'instruction du loop, à 0x0040000c.

## Question C2

- A est la première structure en mèmoire du segment data, donc
  @A : 0x01000000
- @B : $@A + 20 \times 4 + 48 =$ 0x01000080
- @C : $@B + 20 \times 4 + 48 =$ 0x01000100

## Question C3

Parce que le MIPS32 est un processeur pipeline 5 étages et qu'à cause du plan pipeline de ce processeur, lors d'un branchement, l'instruction suivante est systématiquement exécutée, sinon on ne peut pas calculer l'adresse après le branchement. On met en temps normal un delayed slot (nop) pour faire face à cette contrainte mais mettre le sw à cette endroit est une optimisation.

## Question C4

Considérons que la boucle d'itération soit suffisament longue en nombre d'instructions pour que le TEP ne congéstionne pas au *sw* de chaque itération, chaque instruction serait exécutée son cycle, car aucun MISS ne serait présent. Donc 7 instructions par itération de boucle.

## Question D1

- 1 ligne de cache est composée de 16 octets. $BYTE = log_2(16) = 4$ bits
- Le cache est à correspondance directe $\rightarrow$ 1 case accessible par ligne.
  $SET = log_2(\frac{128}{16}) = 3$ bits.
- $TAG = 32 - (3 + 4) = 25$ bits.

## Question D2

Le premier MISS d'instruction est le MISS mandataire pour le chargement du code du *main* :

```
400000: 3c080100   lui t0,0x100
400010: 8d0a0000   lw  t2,0(t0)
400020: 014b6020   add t4,t2,t3
```

app.bin.txt

| #ligne | valide? | tag | mot3 | mot2 | mot1 | mot0 |
|---|---|---|---|---|---|---|
| 7 | 0 | | | | | |
| 6 | 0 | | | | | |
| 5 | 0 | | | | | |
| 4 | 0 | | | | | |
| 3 | 0 | | | | | |
| 2 | 1 | $0x004000_0$ | 0x3c040040 | 0xad0c00fc | 0x14c7fffa | 0x014b6020 |
| 1 | 1 | $0x004000_0$ | 0x21080004 | 0x20c60001 | 0x8d0b0080 | 0x8d0a0000 |
| 0 | 1 | $0x004000_0$ | 0x24060000 | 0x24070014 | 0x25080080 | 0x3c080100 |

Soit 3 MISS d'instructions.

## Question D3

Avant la première instruction de la boucle, on peut comptabiliser deux MISS.
Parce que toutes les instructions de la boucles sont en mémoire cache après la première itération, à terme de la 20e itération, le contenu du cache d'instruction est inchangé depuis la fin de la première itération.
Au total, on comptabilise 5 MISSes pour $10 + 20 \times 7 = 150$ instructions.
$\tau_{MISS} = \frac{5}{150} \approx 0.033\%$

## Question D4

L'état MISS_SELECT est indispensable pour un cache associatif, parce qu'en dehors du cas d'un cache à correspondance directe, il est possible pour une ligne de cache d'être rangée dans une case du cache parmi plusieurs emplacements possibles.

## Question D5

| **Transition** | **Expression** |
|---|---|
| **A** | $IMISS \bullet IUNC$ |
| **B** | $IMISS \bullet \overline{IUNC}$ |
| **C** | $\overline{IMISS}$ |
| **I** | 1 |
| **F** | $IREQ \bullet VALID \bullet \overline{ERROR}$ |
| **G** | $IREQ \bullet VALID \bullet ERROR$ |
| **H** | $\overline{IREQ} + \overline{VALID}$ |
| **N** | 1 |
| **L** | $IREQ \bullet VALID \bullet \overline{ERROR}$ |
| **K** | $IREQ \bullet VALID \bullet ERROR$ |
| **J** | $\overline{IREQ} + \overline{VALID}$ |
| **M** | 1 |
| **O** | 1 |

## Question D6

Le signal RESETN doit forcer l'automate du cache d'instruction à l'état IDLE pour qu'il soit prêt à recevoir la première requête du processeur: charger le code du bootloader depuis la ROM. L'autre effet qui se coule est que le cache doit être prêt à recevoir de l'information sur l'intégralité de sa mémoire, donc l'intégralité du cache doit être invalidé.

# Question E1

Voici les champs des adresses de A[0] et B[0] :

| | Adresse | TAG | SET | BYTE |
|---|---|---|---|---|
| A | 0x01000000 | $0x010000_0$ | 0 | 0 |
| B | 0x01000080 | $0x010000_1$ | 0 | 0 |

Les deux instructions générant des MISSes sont les suivantes :

```
400010: 8d0a0000   lw   t2,0(t0)
400014: 8d0b0080   lw   t3,128(t0)
```

app.bin.txt

La première génere un MISS compulsif car la donnée n'est pas présente en mémoire, tous les MISSes suivants génèrent des MISSes de conflits, alors que les valeurs de A vont écraser les valeurs de B.

| #ligne | valide? | tag | mot3 | mot2 | mot1 | mot0 |
|---|---|---|---|---|---|---|
| 7 | 0 | | | | | |
| 6 | 0 | | | | | |
| 5 | 0 | | | | | |
| 4 | 0 | | | | | |
| 3 | 0 | | | | | |
| 2 | 0 | | | | | |
| 1 | 0 | | | | | |
| 0 | 1 | $0x0100000_1$ | 104 | 103 | 102 | 101 |

Soit deux MISSes par itération de boucle.

# Question E2

On comptabilise à terme $2 \times 20 = 40$ MISSes de données.
Voici l'état du cache à la 20e itération:

| #ligne | valide? | tag | mot3 | mot2 | mot1 | mot0 |
|---|---|---|---|---|---|---|
| 7 | 0 | | | | | |
| 6 | 0 | | | | | |
| 5 | 0 | | | | | |
| 4 | 1 | $0x0100000_1$ | 120 | 119 | 118 | 117 |
| 3 | 1 | $0x0100000_1$ | 116 | 115 | 114 | 113 |
| 2 | 1 | $0x0100000_1$ | 112 | 111 | 110 | 109 |
| 1 | 1 | $0x0100000_1$ | 108 | 107 | 106 | 105 |
| 0 | 1 | $0x0100000_1$ | 104 | 103 | 102 | 101 |

## Question E3

| Transition | Expression |
|---|---|
| **A** | $DMISS \bullet DUNC \bullet \overline{WRITE}$ |
| **B** | $DMISS \bullet \overline{DUNC} \bullet \overline{WRITE}$ |
| **C** | $\overline{DMISS} + \overline{WRITE}$ |
| **I** | 1 |
| **F** | $IREQ \bullet VALID \bullet \overline{ERROR}$ |
| **G** | $IREQ \bullet VALID \bullet ERROR$ |
| **H** | $\overline{IREQ} + \overline{VALID}$ |
| **N** | 1 |
| **L** | $IREQ \bullet VALID \bullet \overline{ERROR}$ |
| **K** | $IREQ \bullet VALID \bullet ERROR$ |
| **J** | $\overline{IREQ} + \overline{VALID}$ |
| **M** | 1 |
| **O** | 1 |
| **D** | $WRITE \bullet \overline{WOK}$ |
| **P** | 1 |
| **E** | $WRITE \bullet WOK$ |

## Question E4

[TODO]

## Question F1

Les écritures ont une priorité plus élevée pour assurer le vidage du TEP.

## Question F2

Afin d'assurer la communication client/serveur, un système de deux bascules RS est employé. Le client, après préparation de sa requête met à SET la bascule REQ, qui sera remis à RESET par le serveur pour prévenir le client de l'aquittement de la requête.
Le serveur, après le transfert, met la bascule RSP à SET pour prévenir le client de la fin de la transaction. Le client lui, la met à RESET pour prévenir le serveur de l'aquittement de la réponse.

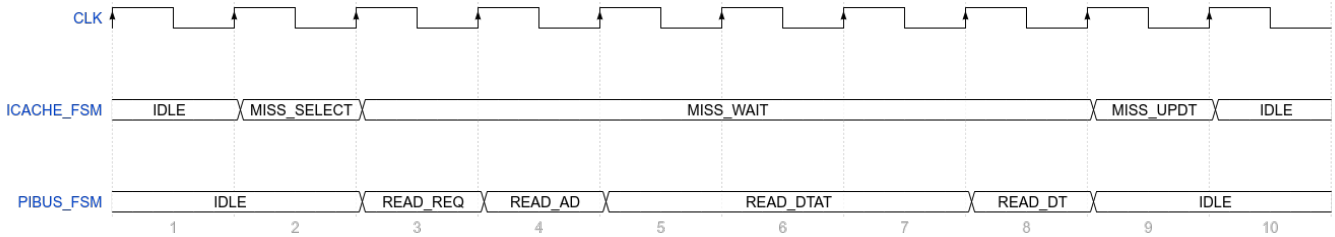
## Question F3

[TODO]

## Question F4

| Transition | Expression |
|---|---|
| Y | $(IUNC + IMISS) \bullet \overline{ROK} \bullet \overline{SC}$ |
| Y | $(DUNC + DMISS) \bullet \overline{ROK} \bullet \overline{SC}$ |
| Z | $\overline{IUNC + IMISS + DUNC + DMISS + ROK + SC}$ |
| B' | $\overline{GNT}$ |
| B | $GNT$ |
| E' | $\overline{GNT}$ |
| E | $GNT$ |
| C | 1 |
| D' | $ACK \oplus WAIT$ |
| D | $\overline{ACK \oplus WAIT}$ |
| H' | $ACK \oplus WAIT$ |
| H | $\overline{ACK \oplus WAIT}$ |
| F' | $\overline{LAST}$ |
| F | $LAST$ |
| G' | $\overline{LAST}$ |
| G | $LAST$ |

## Question F5




L'évolution des états internes du DCACHE_FSM est identique à celle représentée ci-dessus pour une lecture. 9 cycles de gel en cas de MISS dans les deux cas.

## Question F6

Au total, $7 * 20 = 140$ instructions sont exécutées pour toutes les itérations de boucle.
À la première itération on comptabilise deux MISS d'instruction compulsifs.
À toutes les itérations on comptabilise deux MISS de données dûs aux conflits.
Sachant que le poids du MISS est de 9 cycles de gel :

- $\#cycle = (\underbrace{2 \times 10}_{imiss} + \underbrace{2 \times 10}_{dmiss} + 5) + 19 \times (\underbrace{2 \times 10}_{dmiss} + 5) = 520$ cy
- $CPI = \frac{525}{140} \approx 3.714$!

# Question G1

La première instruction s'exécute au cycle 10 :

```
******* cycle = 10 **************
bcu : fsm = IDLE
proc : <InsReq     valid mode MODE_KERNEL @ 0xbfc00000>
proc : <InsRsp     valid no error ins 0x3c1d0200>
proc : <DataReq invalid mode MODE_HYPER type DATA_READ @ 0 wdata 0 be 0>
proc : <DataRsp invalid no error rdata 0>
proc : ICACHE_IDLE  DCACHE_IDLE  PIBUS_IDLE
rom : IDLE
ram : IDLE
tty : IDLE   keyboard status[0] = 0   display status[0] = 0
  -- pibus signals --
req     = 0
gnt     = 0
sel_rom = 0
sel_ram = 0
sel_tty = 0
avalid  = 0
read    = 0x1
lock    = 0
address = 0xbfc0000c
ack     = 0x2
data    = 0x341aff13
```

trace.txt

Cela correspond à la première instruction du segment *seg_reset*. Pendant les dix premiers cycles (du cycle 0 au cycle 9), le processeur est gelé. On retrouve bien la valeur théorique du nombre de cycles de gel pour un MISS d'instruction.

# Question G2

Le branchement vers la première instruction du **main** est :

```
bfc00024: 42000018  eret
```

sys.bin.txt

La commande eret passe du code système au code utilisateur via un branchement à l'adresse stockée dans le registre EPC (au démarrage de la machine, la valeur de EPC est à 0x00400000). Le branchement se fait au cycle 46 :

```
******* cycle = 46 **************
bcu : fsm = IDLE
proc : <InsReq     valid mode MODE_KERNEL @ 0xbfc00024>
proc : <InsRsp     valid no error ins 0x42000018>
proc : <DataReq invalid mode MODE_KERNEL type DATA_READ @ 0x1000000 wdata 0 be 0xf>
proc : <DataRsp invalid no error rdata 0x400000>
proc : ICACHE_IDLE  DCACHE_IDLE  PIBUS_IDLE
rom : IDLE
ram : IDLE
tty : IDLE   keyboard status[0] = 0   display status[0] = 0
  -- pibus signals --
req     = 0
gnt     = 0
sel_rom = 0
sel_ram = 0
sel_tty = 0
avalid  = 0
read    = 0x1
lock    = 0
address = 0xbfc0002c
ack     = 0x2
data    = 0
```

trace.txt

## Question G3

On remarque en observant la trace qu'une lecture de donnée non en cache au cycle 71 prend jusqu'au cycle 81 pour être menée, soit 10 cycles. En enlevant la requête, le coût du MISS est de 9 cycles. La première itération commence au cycle 71 et se termine au cycle 107, soit 36 cycles :

```
******* cycle = 71 **************
bcu : fsm = IDLE
proc : <InsReq     valid mode MODE_USER @ 0x400010>
proc : <InsRsp     valid no error ins 0x8d0a0000>
proc : <DataReq invalid mode MODE_KERNEL type DATA_READ @ 0x1000000 wdata 0 be 0xf>
proc : <DataRsp invalid no error rdata 0x400000>
proc : ICACHE_IDLE  DCACHE_IDLE  PIBUS_IDLE
rom : IDLE
ram : IDLE
tty : IDLE   keyboard status[0] = 0   display status[0] = 0
  -- pibus signals --
req     = 0
gnt     = 0
sel_rom = 0
sel_ram = 0
sel_tty = 0
avalid  = 0
read    = 0x1
lock    = 0
address = 0x40001c
ack     = 0x2
data    = 0x21080004
```

trace.txt

```
******* cycle = 107 **************
bcu : fsm = IDLE
proc : <InsReq     valid mode MODE_USER @ 0x400028>
proc : <InsRsp     valid no error ins 0xad0c00fc>
proc : <DataReq invalid mode MODE_USER type DATA_READ @ 0x1000100 wdata 0 be 0xf>
proc : <DataRsp invalid no error rdata 0x65>
proc : ICACHE_IDLE  DCACHE_IDLE  PIBUS_IDLE
rom : IDLE
ram : IDLE
tty : IDLE   keyboard status[0] = 0   display status[0] = 0
  -- pibus signals --
req     = 0
gnt     = 0
sel_rom = 0
sel_ram = 0
sel_tty = 0
avalid  = 0
read    = 0x1
lock    = 0
address = 0x40002c
ack     = 0x2
data    = 0x3c040040
```

trace.txt

## Question G4

La seconde et troisième itération prennent chaqu'une 30 cycles. Le coût des MISSes de données sont plus importants parce qu'il faut maintenant invalider la ligne de cache dans la case où on veut charger la nouvelle ligne depuis la mémoire.

## Question G5

Le taux de MISS est de 100% à la fin de l'exécution de la boucle.
La dernière instrcution de la dernière boucle est au cycle 696 :

```
******* cycle = 697 **************
bcu : fsm = IDLE
proc : <InsReq    valid mode MODE_USER @ 0x40002c>
proc : <InsRsp    valid no error ins 0x3c040040>
proc : <DataReq   valid mode MODE_USER type DATA_WRITE @ 0x10001cc wdata 0x8c be 0xf>
proc : <DataRsp   valid no error rdata 0>
proc : ICACHE_IDLE  DCACHE_WRITE_REQ  PIBUS_IDLE
rom : IDLE
ram : IDLE
tty : IDLE   keyboard status[0] = 0   display status[0] = 0
  -- pibus signals --
req     = 0
gnt     = 0
sel_rom = 0
sel_ram = 0
sel_tty = 0
avalid  = 0
read    = 0x1
lock    = 0
address = 0x100014c
ack     = 0x2
data    = 0x78
```

trace.txt

Sachant que la première instruction du **main** est exécutée au cycle 57, l'exécution du main a pris 639 cycles.

## Question H

Le problème que l'on a actuellement est que les adresses sont toutes deux alignées entre elles, et que chaque requête de lecture de tableaux engendre des MISSes de conflit. Il faudrait assigner le tableau B à une autre adresse afin que les lignes de caches dans lesquelles elles figurent n'entre pas en collision sur la même case.
On peut désaligner l'adresse de B, avec du bourrage d'espace pour éviter les collisions. Par exemple en positionnant la base de B de telle sorte qu'elle soit chargée dans la case suivant celle de B:

- Par exemple, positionner la base du tableau B à l'adresse 0x01000090 ferait que la première ligne de cache contenant une portion de B sera chargée à la case d'index numéro 1

On aurait alors 2 MISSes de données toutes les 5 itérations :

- $\#cycles = 15 \times 7 + 4 \times 25 + 1 \times 45 = 250$
- $CPI = \frac{250}{140} \approx 1.786$, soit plus de deux fois plus performant.