Master SESI

Systèmes Multi-processeurs

Cours du professeur Alain Greiner

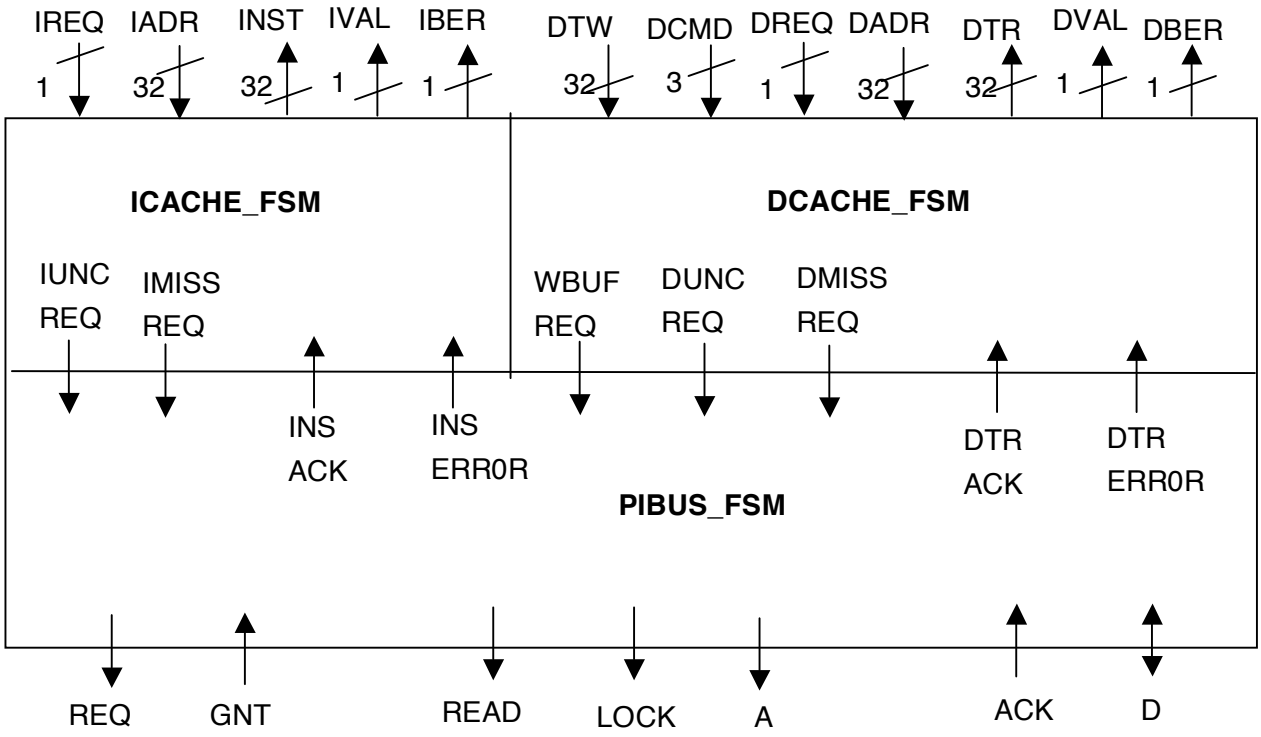
Examen de juin 2011

Cet examen est sans documents

PARTIE A : Contrôleur de cache (10 points)

On s'intéresse dans cette première partie à l'architecture interne d'un contrôleur de cache contenant deux caches séparés pour les données et pour les instructions. On suppose que la largeur d'une ligne de cache est de 32 octets. La capacité totale de chacun des deux caches est de 16 Koctets. Chaque cache possède 4 niveaux d'associativité. La stratégie d'écriture est de type WRITE-THROUGH, et il possède un tampon d'écritures postées de profondeur égale à 8 mots de 32 bits.

Ce contrôleur s'interface d'un côté avec un processeur MIPS32 à architecture pipe-line, capable de démarrer une nouvelle instruction à chaque cycle. Le processeur peut effectuer deux requêtes à chaque cycle : une requête de lecture instruction, et une requête de lecture ou écriture de donnée. Le contrôleur de cache s'interface de l'autre côté avec le PIBUS pour accéder à la mémoire en cas de lecture faisant miss, en cas de lecture non cachable, ou en cas d'écriture. Il contient trois automates ICACHE_FSM, DCACHE_FSM et PIBUS_FSM, qui contrôlent respectivement chacun des trois interfaces.



QA1) (1 point) Rappeler la différence entre un cache à correspondance directe et un cache associatif par ensembles.

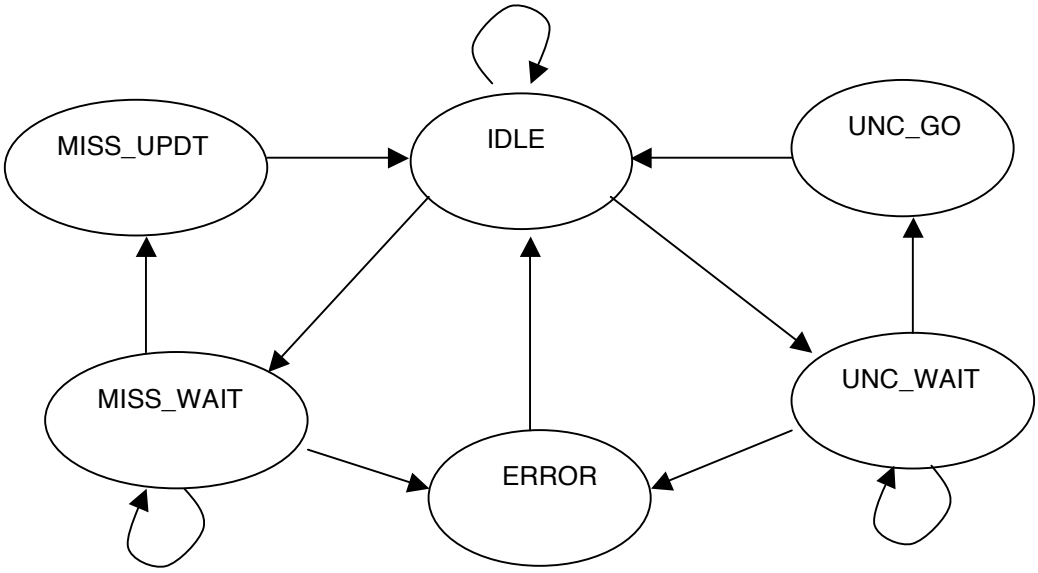
QA2) (1 point) Lorsque le processeur effectue une requête de lecture vers l'un des deux caches, le contrôleur du cache doit analyser l'adresse 32 bits en la décomposant en trois champs. Rappelez la signification des trois champs OFFSET, INDEX et TAG, et donnez le nombre de bits pour chacun de ces trois champs pour le cache défini ci-dessus.

On s'intéresse maintenant à l'automate ICACHE_FSM, qui contrôle les accès au cache instructions, et répond aux requêtes de lecture d'instruction en provenance du processeur. Cet automate est capable de traiter des requêtes de lecture cachables (états MISS_WAIT et MISS_UPDT, et non cachables (états UNC_WAIT et UNC_GO). Il doit également reporter au processeur les éventuelles erreurs d'adressage signalées par l'automate PIBUS_FSM. Les conditions de transition dépendent des signaux d'entrée suivants :

- IREQ : l'adresse transmise par le processeur est une requête instruction valide,
- IUNC : le décodeur qui analyse les bits de poids fort de l'adresse IADR signale que cette adresse appartient à un segment non cachable,
- IMISS : le répertoire du cache instruction signale que l'adresse instruction correspond à une ligne de cache non présente dans le cache
- INS_ACK : l'automate PIBUS signale que la réponse à la transaction effectuée sur le bus est disponible (cette réponse peut être un succés ou un échec en cas d'erreur d'adressage)
- INS_ERROR : l'automate PIBUS signale que la réponse correspond à une erreur d'adresse.

L'automate ICACHE FSM génère lui-même, à chaque cycle, différents signaux de sortie , soit vers le processeur, soit vers l'automate PIBUS chargé de contrôler les transactions sur le bus :

- IVAL : signal indiquant au processeur que l'instruction qui lui est transmise est valide.
- IBER : signal indiquant une erreur d'adresse instruction.
- IUNC_REQ : signal demandant la lecture d'une instruction non cachable en mémoire.
- IMISS_REQ : signal demandant la lecture d'une ligne de cache en mémoire.



QA3) (2 points) Complétez le graphe de l'automate ICACHE_FSM ci-dessus, en attachant à chaque transition sa condition de franchissement.

QA4) (1 point) Dans quel(s) état(s) le signal IVAL peut-il prendre la valeur true ? Donnez l'expression Booléenne du signal IVAL, dépendant de l'état de l'automate, ainsi que des signaux IMISS et IUNC.

QA5) (1 point) Pour ce qui concerne les écritures de données, rappelez la différence entre une stratégie WRITE-THROUGH et une stratégie WRITE-BACK. A quoi sert le tampon d'écritures postées, et que se passe-t-il quand on diminue la profondeur de ce tampon ?

On s'intéresse à l'automate PIBUS_FSM, qui contrôle les transactions sur le bus.

Les conditions de transition dépendent des signaux suivants :
- WBUF_REQ : Le tampon d'écritures postées est non vide et demande une écriture en mémoire
- IUNC_REQ : L'automate ICACHE_FSM demande une lecture d'un mot non cachable en mémoire
- IMISS_REQ : L'automate ICACHE_FSM demande une lecture d'une ligne de cache en mémoire
- DUNC_REQ : L'automate DCACHE_FSM demande une lecture d'un mot non cachable en mémoire
- DMISS : L'automate DCACHE_FSM demande une lecture d'une ligne de cache en mémoire
- GNT : Le contrôleur de bus alloue le bus au composant PIbusMips32Xcache
- LAST : le compteur de mots interne signale que c'est la dernière adresse d'une rafale
- ACK : Réponse de la cible adressée (3 valeurs possibles : READY, WAIT, ERROR)

L'automate PIBUS_FSM est un « serveur » qui peut donc recevoir 5 requêtes. Si il y a plusieurs requêtes simultanées, il implémente la priorité fixe suivante :

WBUF_REQ > DUNC_REQ > DMISS_REQ > IUNC_REQ > IMISS_REQ

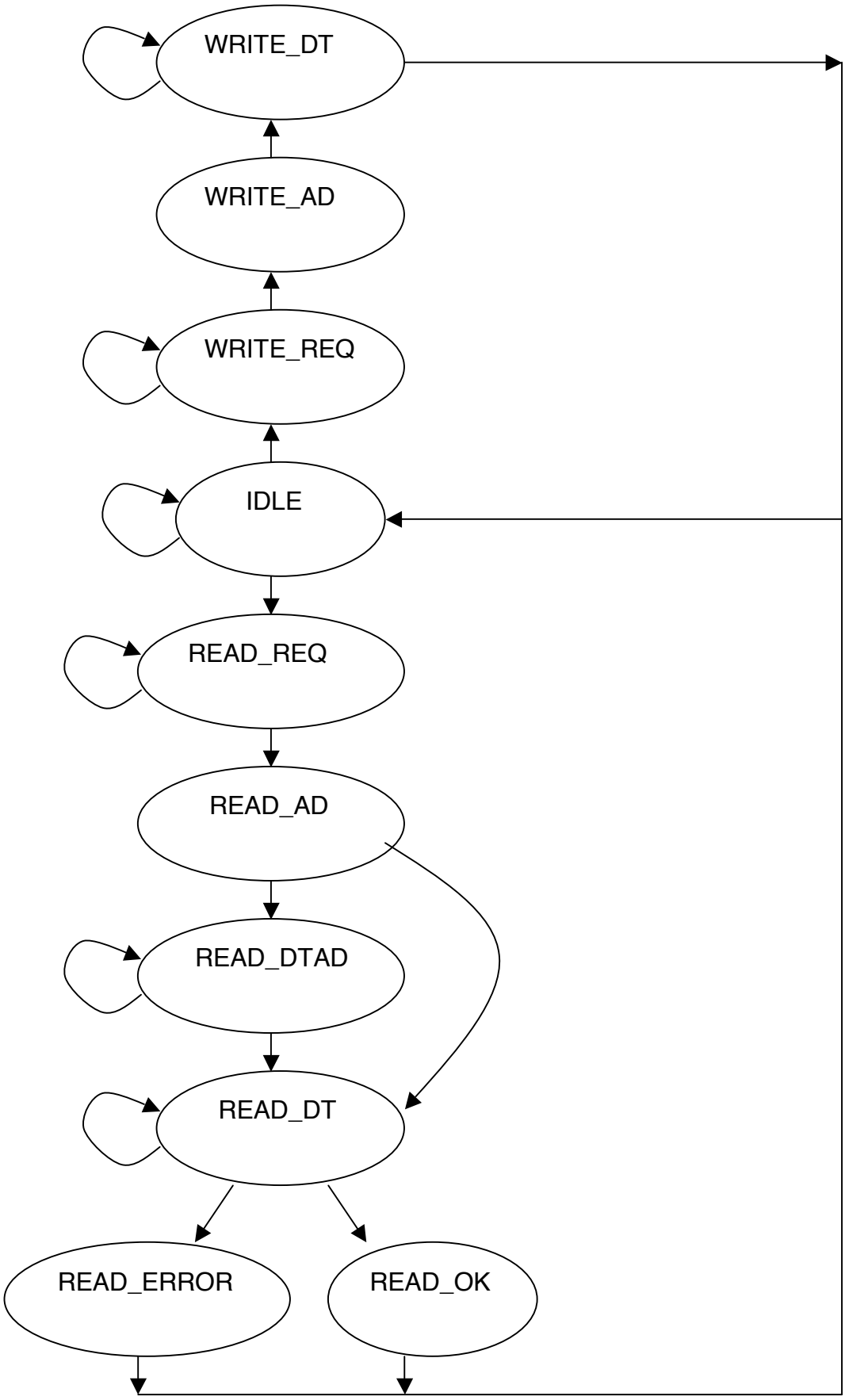
L'automate PIBUS_FSM génère lui-même différents signaux soit vers le bus, soit vers les automate DCACHE_FSM et ICACHE_FSM :

- INS_ACK : signal indiquant à l'automate ICACHE_FSM que la transaction est terminée.
- INS_ERR0R : signal indiquant à l'automate ICACHE_FSM une erreur d'adresse.
- DTR_ACK : signal indiquant à l'automate DCACHE_FSM que la transaction est terminée.
- DTR_ERROR ; signal indiquant à l'automate DCACHE_FSM une erreur d'adresse.
- REQ : signal demandant l'allocation du PIBUS
- READ : signal définissant le sens du transfert sur le bus de données.
- LOCK : signal indiquant que l'adresse sur le bus n'est pas la dernière d'une rafale.

QA6) (1 point) Pourquoi a-t-on choisi que les écritures soient prioritaires par rapport aux requêtes de lectures (données ou instruction) ?

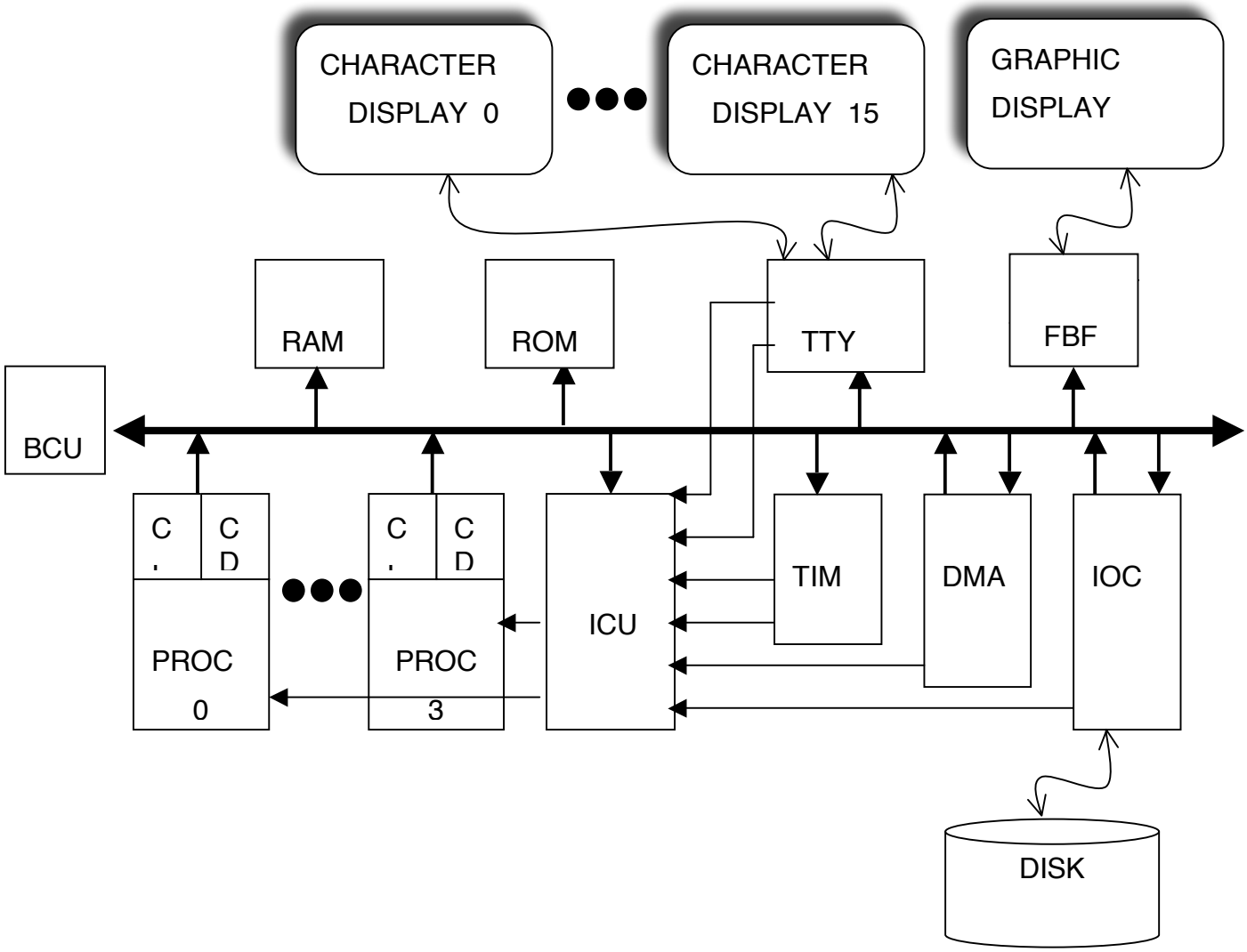
QA7) (2 points) Complétez le graphe de l'automate ICACHE_FSM ci-dessous, en attachant à chaque transition sa condition de franchissement.

QA8) (1 point) Donnez les expressions Booléennes des signaux REQ et READ en fonction des états de l'automate.


WRITE_DT
WRITE_AD
WRITE_REQ
IDLE
READ_REQ
READ_AD
READ_DTAD
READ_DT
READ_ERROR
READ_OK

PARTIE B : Interruptions (10 points)

On considère l'architecture matérielle multi-cores ci-dessous, comportant quatre processeurs, une mémoire (RAM), une mémoire morte (ROM), un contrôleur de terminaux alpha-numériques (TTY), un timer programmable (TIM), un contrôleur de disque (IOC), un cotrôleur graphique (FBF), un contrôleur DMA, et un contrôleur d'interruptions (ICU).



Chaque processeur peut exécuter au plus 4 tâches en pseudo-parallélisme, par multiplexage temporel. Chaque tâche est un programme indépendant, qui dispose de son propre terminal TTY. On a donc 16 programmes (et 16 terminaux) repérés par le couple (p,t), ou *p* est l'index du processeur, et *t* est l'index de la tâche. On a également 16 lignes d'interruption IRQ_TTY(p,t) associées à chacun des 16 terminaux.

QB1) (1point) Rappelez le principe du fonctionnement multi-tâches par multiplexage temporel préemptif. Comment un seul processeur peut-il exécuter 4 programmes «en même temps ?

On rappelle que chaque terminal TTY possède 4 registres adressables par le logiciel :

- DISPLAY (write only / offset = 0) : code ASCII du caractère à afficher
- STATUS (read only / offset = 4) : flags décrivant l'état des tampons DISPLAY et KEYBUF
- KEYBUF (read only / offset = 8) : code ASCII du caractère saisi au clavier
- CONFIG (write only / offset = 12) : inutilisé

QB2) (1point) Quelle est la longueur du segment mémoire associé au contrôleur TTY ? Comment le système d'exploitation qui souhaite lire ou écrire une valeur dans un registre d'un terminal (p,t) adresse-t-il ce registre sachant que le numéro de terminal est égal à (4*p + t).

Puisqu'on a 4 processeurs, on a besoin de quatre interruptions IRQ_TIMER(p) provenant du timer programmable pour déclencher les changements de contexte. On a également une ligne d'interruption IRQ_IOC provemant du contrôleur IOC et une ligne d'interruption IRQ_DMA provenant du contrôleur DMA. On a donc au total 16 (TTY) + 4 (TIMER) + 1 (IOC) + 1 (DMA) = 22 lignes d'interruption entrantes sur le contrôleur ICU, qui sont connectées dans cet ordre (16 TTY, puis 4 TIMER, puis IOC, puis DMA) sur les entrées IRQ_IN [0] à IRQ_IN[21] du contrôleur ICU.

QB3 (1 point) Qu'est-ce qu'une ISR (Interrupt Service Routine) ? Combien d'ISRs différentes sont nécessaires dans cette architecture ?

QB4) (1 point) Qu'est-ce qu'un vecteur d'interruption ? Quel doit être le contenu du vecteur d'interruptions dans le cas de cette architecture ?

Pour répartir la charge, on souhaite router ces interruptions de la façon suivante vers les 4 processeurs :

- Processeur 0 : TTY(0,0) / TTY(0,1) / TTY(0,2) / TTY(0,3) / TIMER(0) / IOC
- Processeur 1 : TTY(1,0) / TTY(1,1) / TTY(1,2) / TTY(2,3) / TIMER(1) / DMA
- Processeur 2 : TTY(2,0) / TTY(2,1) / TTY(2,2) / TTY(2,3) / TIMER(2)
- Processeur 3 : TTY(3,0) / TTY(3,1) / TTY(3,2) / TTY(3,3) / TIMER(3)

QB5) (1 point) Quelles valeurs hexadécimales faut-il écrire dans les registres de masque du composant ICU pour réaliser ce routage ?

On suppose que le programme (p,t) exécute l'appel système *tty_getc_irq(char* byte)* qui a pour effet lire un caractère saisi au clavier sur son terminal, et de ranger le code ASCII de ce caractère dans une variable locale (*byte)* de la fonction contenant l'appel système.

QB6) (1 point) Quelle erreur peut-elle être détectée et reportée par cet appel système ? Du point de vue du programme utilisateur, cet appel système est-il bloquant ou non-bloquant ? Dans quel segment mémoire sera rangé la variable *byte* ?

Lorsque l'utilisateur frappe une touche du clavier correspondant au terminal (p,t), le programme (p,t) qui attend ce caractère n'a qu'une chance sur quatre d'être en cours d'exécution sur le processeur p.

QB7) (1 point) Détaillez le mécanisme qui garantit que le caractère frappé sur le clavier du terminal (p,t) sera bien acheminé vers le programme (p,t) et rangé dans la variable *byte*. Précisez quelle fonction effectue la lecture du caractère dans le registre matériel du contrôleur TTY(p,t), et quelle fonction effectue l'écriture dans la variable byte.

Lorsqu'un programme souhaite effectuer un transfert depuis le disque vers un tampon mémoire, il utilise les deux appels système i*oc_read(size_t lba, void* buffer, size_t length)*, et *ioc_completed().* L'argument *lba* est le numéro du premier bloc à lire sur le disque, l'argument *buffer* est l'adresse du tampon mémoire dans l'espace utilisateur, et l'argument *length* est le nombre de blocs à transférer.

QB8) (1 point) Quelle est la signification de l'interruption associée au contrôleur IOC ? Que fait l'ISR associée à cette interruption ?

QB9) (1 point) Que font les deux appels système ioc_read() et ioc_comleted() ? Quand rendent-elles la main au programme utilisateur ?

Lorsqu'un programme utilisateur effectue une lecture sur le disque, il peut commettre deux types d'erreur d'adressage : Le premier type d'erreur est de choisir une adresse *buffer* appartenant à la zone réservée au système d'exploitation (*segmentation violation*). Le second type d'erreur est de fournir une adresse *buffer* ne correspondant à aucun segment défini dans l'architecture (*bus error*). Ces erreurs sont signalées par un code d'erreur retourné par l'appel système.

QB10) (1 point) Dans une architecture sans mémoire virtuelle, quel type d'erreur est signalé par l'appel système ioc_read() ? Quel type d'erreur est signalé par l'appel système ioc_completed() ? Justifiez la réponse.