Nom : AIDER
Prénom : Smail
N°Etudiant : 3603379
Parcours : SAR
Responsable : M Alain Greiner

Compte-Rendu TP2
Déploiement de code sur processeur programmable

C) *Modélisation de l'architecture matérielle*

Question C1 :

Cache size : 1 Kbytes.
Cache block size : 4 words = 16 bytes.
Pas d'associativité : 1-way set associative (direct-mapped cache).

Donc : 1Kbytes / 16bytes = 64 / (number_of_ways) = 64 sets .

Daprés le pibus_mips32_xcache.h :

Nombre de sets : puisscance de 2 (<=1024).
Nombre de mots/lignes : puissance de 2 (≤ 32).
Nombre de ways : puissance de 2 (≤ 8).

- icache_ways : 1
- icache_sets : 64
- icache_words : 4
- dcache_ways : 1
- dcache_sets : 64
- dcache_words : 4
- wbuf_depth : 8

Question C2 :

le segment seg_reset est assigné à la ROM et non pas à la RAM parce que c'est le premier segment que le processeur va executer au démarrage, et à cette instant(boot), la RAM n'est pas encore intitialisée. La RAM ne peut pas contenir le boot-block (seg_reset) vu qu'elle est volatile.

Question C3 :

Le segement seg_tty doit etre non cachable pour une raison de **cohérence.**
Toute région accédée par le materiel(hardware) et le logiciel(software) doit etre non cachable.
Le segment TTY peut etre modifier sans avoir à notifer le processeur.
Donc si le contenu d'une adresse changerait, le processeur peut retourner une valeur dont il se souvient(du cache) qui peut etre fausse.

Question C4 :

Les segements protégés sont :
- segment seg_kcode, seg_kunc, seg_kdata, seg_tty, seg_reset.

Cette protection est realisée via le bit de poids fort (MSB), 1: Kernel, 0: User.
Dans cette exemple, on a un processeur 32 bits, donc un espace d'adressage 2^32(de 0x00000000 -0xFFFFFFFF).
On découpant cette espace en deux(user-kernel), on obtient :
- Kernel : 0x**8**0000000 – 0xFFFFFFFF
- User : 0x00000000 – 0x**7**FFFFFFF

On voit bien que les segments qui doivent etre executer en mode superviseur commencent tous par une adresse de base ≥ 0x80000000.

D) *Systeme d'exploitation : GIET*

Question D1:

Les informations que doit fournir un programme utilisateur lors de l'éxecution d'un appel systeme sont :
- Le numéro de l'appel systeme
- Les arguments nécessaires(PC, SP, args ...)

Ces informations sont transmises via des registres.

Question D2:

Le GIET utilise deux tableaux de fonctions :
1) _cause_vector[16] :
- Definie les 16 causes (exceptions) pour entrer dans le GIET
- Initialiser dans exc_handler.c
- Indexer par des registres

2) _syscal_vector[32] :
- Definie les adresses des 32 handlers correspondant aux appels systemes
- Initialiser dans sys_handler.c
- Indexer par l'indice de chaque appel systeme

Question D3:

Les appels de fonctions déclenchée par l'appel systeme proctime() :
- stdio.c -- sys_call() : Met le numéro de l'appel systeme (SYSCALL_PROCTIME) dans un registre et déclenche une trap.
- giets.s -- sys_handler : Analyse le numéro de l'appel systeme et appelle le handler associé
- drivers.c -- _proctime() : Accede au registre CP0 et retourne le nombre de cycles d'horloge écoulés depuis le démarrage(boot-up).

Question D4:

Une estimation du cout de l'appel systeme « proctime » : ~90 cycles.

E) *Génération du code binaire*

Question E1:

Le code de boot doit nécessairement s'exécuter en mode superviseur car on accede aux registres protéges du processeur.

Question E2:

La convention est :
la premiere adresse du segment data doit contenir le point le code d'entrée dans le code applicatif.

Question E3:

Si les adresses définies dans ces deux fichiers ne sont pas égales, le processeur va invalider toutes les adresses ne correspondant pas aux adresses definies pour le matériel.

Question E4:

Les segments logiciels placés dans :
- seg_reset : reset.o
- seg_kcode : giet.o, common.o, ctx_hander.o, irq_handler.o, sys_handler.o, exc_handler.o, driver.o

Question E5:

La longueur effective des segments :
- seg_reset : 8 lignes
- seg-kcode : 2215 lignes

Question E7:

La fonction qui contient la boucle d'attente est : tty_getc() (stdio.c) car :
On ne veut pas qu'un processus monopolise le processeur.
Si on avait fait la boucle d'attente dans le code de la fonction systeme _tty_read() (driver.c), le processeur ne pourait pas executer d'autres commandes tty_getc() d'autres processus.

Question E8:

La longueur effective du segment :
- seg_code : 1304 lignes

Question E9:

Makefile : OK

F) *Exécution du code binaire sur le prototype virtuel*

Question F1:

La premiere transation sur le bus correspond à un : SEL ROM.

La premiere instruction du code de boot est exécutée au cycle : 2
- la premiere adresse (du segment seg_reset) est chargée au deuxieme cycle, la donnée est disponible au troisieme cycle 3.

La deuxieme transaction sur le bus consiste à lire le premiere mot du segment data :
- Deuxieme transation (cycle 20) : la derniere adresse exécutée est « 0xbfc0001c » qui correspand à l'instruction « lw » dans « sys.bin.txt »
- Troisieme transation (cycle 30) : la premiere adresse chargée est « 0x1000000 » qui correspand à l'adresse du premiere octet du ségment data
→ Donc, la transaction 2 consistait à lire le premiere octet du segment data.

Question F2:

Dans le fichier « app.bin.txt », on voit que la premiere adresse du main est « 0x4012dc ».
Dans la trace d'exécution, cette adresse est chargée au cycle 31.
→Donc la premiere instruction de la fonction main(à s'exécute au cycle 31

Question F4:

La premiere écriture d'un caractere vers le terminal TTY se fait au cycle : 1108