COGNOMS: . . . . . NOM: . . . . . .

2on Control Arquitectura de Computadors **Curs 2015-2016 Q2**

- **Temps: 13:30 a 15:30**
- **Poseu clarament amb LLETRES MAJÚSCULES a cada full els cognoms i el nom**

Problema 1. (3 puntos)

Dado el siguiente código escrito en C:

```
struct str{
  char k;
  int w[3];
};

int subru(str *ps, char vc[10]);

int examen(str sin, char vc[10], char c, int i) {
  str vs[5];
  int j;
  ...
     j = subru(&sin, vc[]);
  ...
};
```

a) **Dibuja** el struct `str` indicando claramente el tamaño de cada campo, su desplazamiento respecto al inicio de la estructura y el tamaño total del struct.

b) **Dibuja** el bloque de activación de la subrutina examen, indicando claramente los desplazamientos y el tamaño de cada uno de los campos.

c) **Traduce** a ensamblador del IA32 la sentencia `j = subru(&sin, vc[]);`

d) La sentencia `j = subru(&sin, vc[]);` se encuentra dentro de un bucle. Deseamos almacenar la variable de inducción del bucle (comúnmente conocida como contador del bucle) en un registro durante toda la ejecución del bucle. **Indica** los registros más adecuados para almacenar el contador del bucle, con el objetivo de minimizar el número de instrucciones ejecutadas.

COGNOMS: . . . . . NOM: . . . . . .

2on Control Arquitectura de Computadors **Curs 2015-2016 Q2**

- **Temps: 13:30 a 15:30**
- **Poseu clarament amb LLETRES MAJÚSCULES a cada full els cognoms i el nom**

Problema 2. (3 puntos)

a) Tenemos un sistema formado por cuatro bloques: un procesador, una memoria cache unificada con política de escritura *Write Through + Write No Allocate*, una memoria principal y un buffer de escritura de 3 entradas para reducir el tiempo de las escrituras. **Dibuja** un esquema a nivel de bloques del sistema completo indicando las conexiones entre estos cuatro componentes.

b) Se dispone de una cache directa de 64 KB con líneas de 16 bytes. Para reducir la tasa de fallos de la cache se ha decidido implementar un *prefecth*, para lo cual la cache se ha dividido en 4 bancos de 16KB. Los bancos se organizan de forma que el bloque 0 de memoria se almacena en el banco 0, el bloque 1 en el banco 1 y así sucesivamente, de forma que el bloque *i* se almacena en el banco *i módulo 4*. Esta organización permite realizar el *prefetch* sobre un banco mientras se accede a otro bloque en un banco distinto. Los bits usados para indexar la memoria cache son los 16 bits de menos peso de la dirección (A15 ... A0), indica qué bits se usarían para la selección de banco y justifica la respuesta.

Dado el siguiente código escrito en ensamblador del x86:

```
        movl $0, %ebx
        movl $0, %esi
for:
        cmpl $512*1000, %esi
        jge end

 (a)    movl (%ebx, %esi, 4), %eax
 (b)    addl 2*4*1024(%ebx, %esi, 4), %eax
 (c)    movl %eax, 3*4*1024(%ebx, %esi, 4)

        addl $2, %esi
        jmp for
end:
```

Sabemos que el código se ejecuta en un sistema con memoria cache y memoria virtual. La memoria virtual utiliza páginas de tamaño 4KB y disponemos de un TLB de 4 entradas y reemplazo LRU. La memoria cache de datos (únicos accesos a memoria que contemplaremos en este problema) es *Write Through + Write No Allocate*, de 2 vías con reemplazo LRU, tamaño 4 KB y 32 bytes por bloque. Responde a las siguientes preguntas:

c) **Calcula,** para cada uno de los accesos etiquetados como (a, b, c), el conjunto de la memoria cache al que se accede en cada una de las 9 primeras iteraciones del bucle

iteración	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
a									
b									
c									

d) **Calcula** la cantidad de aciertos y de fallos de cache, en todo el código.

e) Para cada uno de los accesos indicados (etiquetas a, b, c), **indica** a qué página de la memoria virtual se accede en cada una de las siguientes iteraciones del bucle (recuerda que los accesos son a 4 bytes).

iteración	**0**	**1*512**	**2*512**	**3*512**	**4*512**	**5*512**	**6*512**	**7*512**	**8*512**	**9*512**
a										
b										
c										

f) **Calcula** la cantidad de aciertos y de fallos de TLB, en todo el código.

COGNOMS: . . . . . NOM: . . . . . .

2on Control Arquitectura de Computadors **Curs 2015-2016 Q2**

Problema 3. (4 punts)

Tenim una CPU (C1) que te un temps de cicle (Tc) de 1 ns. A l'executar un programa P (que executa $37x10^9$ instruccions) en un simulador de C1 on tots els accessos a memòria fan *hit* a la Cache d'Instrucciones (I$) i Dades (D$) tarda $44{,}4x10^9$ cicles.

a) **Calcula** el CPI ideal (CPIideal) i el temps d'execució en segons (Texec) del programa P en aquest sistema de memòria ideal.

Mesurem el número mig de referències per instrucció (nr) i veiem 1.33refs/instrucció repartides de la següent manera (1) 1.00 refs/inst a instruccions i (2) 0.33 refs/inst a dades.

Amb una I$ i una D$ reals tenim un *miss rate* de D$ del 13% i de I$ del 3%.

En cas d'encert a la I$ i a la D$ el temps de servei es de 1 cicle. En cas de *miss* a la I$ o a la D$ el temps de penalització per accedir a la memòria es de 100 cicles.

La D$ segueix una política d'escriptura amb *Copy Back* i *Write Allocate*, tot i que en el programa P el nombre de blocs modificats es negligible.

b) **Calcula** el temps mig d'accés a memòria en cicles pels accessos a instruccions (TmaI)

c) **Calcula** el temps mig d'accés a memòria en cicles pels accessos a dades (TmaD)

d) **Calcula** el temps mig d'accés a memòria en cicles per tots els accessos (Tma)

e) **Calcula** el temps d'execució del programa P a la CPU C1 amb caches I$ i D$ reals (TexeR1)

Per a millorar el rendiment del programa P dissenyem una nova CPU (C2) a partir de la CPU C1 descrita anteriorment a la que li hem afegit una cache de segon nivell Unificada (L2$). Aquesta cache te un temps de servei en cas d'encert de 12 cicles i una penalització en cas de *miss* de 100 cicles. En el simulador mesurem un *miss rate* local de la L2$ del 53%.

f) **Calcula** el Temps d'execució del programa P a la CPU C2 amb caches I$, D$ i L2$ reals (TexeR2)

La mida de bloc (linea) de totes les caches es de 64 bytes. Els accessos a la I$ son sempre de 4 bytes (la mida de totes les instruccions), els accessos a la D$ son sempre de 8 bytes i els accessos a la L2$ i a la memòria principal (MP) son sempre de 64 bytes:

g) **Calcula** el nombre d'accessos, el nombre de bytes que es llegeixen i el ample de banda (en MBytes/segon) a tots els elements de la jerarquia: I$, D$, L2$ i Memòria Principal (MP), que utilitza el programa P a la CPU C2. Justifica les respostes.

	Accesos	**Bytes llegits**	**Ample de banda**
I$			
D$			
L2$			
MP			

h) **Calcula** el mínim *hit rate* local (h) que hauria de tenir la L2$ per a que un programa s'executi més ràpidament en la CPU C2 que en la C1?