UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament d'Arquitectura de Computadors

Repàs final

Estructura de Computadors (EC)
2023 - 2024 Q2
Adrià Armejach (adria.armejach@upc.edu)

18-19Q2 - Exercici 3

Durant l'execució d'un cert programa la seva taula de pàgines és en el següent estat:

VPN (hex)	PPN (hex)	P	D
...			
1010	1A	1	1
...			
2800	3F	1	0
...			
57C0	25	1	1
...			
77FF	20	1	0
...			

Els darrers accessos que ha realitzat aquest programa han estat en el següent ordre de VPNs: 0x1010, 0x77FF, 0x2800 i 0x57C0.

b) Omple la següent taula per a cada referència a memòria que es realitza posteriorment a l'estat indicat anteriorment.

adr. lògica (hex)		VPN (hex)	fallada de pàgina? (SI/NO)	PPN (hex)	lectura de disc (SI/NO)	escriptura a disc (SI/NO)	VPN pàgina reemplaçada (hex)
E	0x1011C5F4	1011	SI	1A	SI	SI	1010
L	0x10100008	1010	SI	20	SI	NO	77FF
L	0x2800FFFC	2800	NO	3F	NO	NO	
E	0x77FFA400	77FF	SI	25	SI	SI	57C0
E	0x1011A274	1011	NO	1A	NO	NO	
L	0x101010A0	1010	NO	20	NO	NO	

Rendiment i consum

19-20Q1-Parcial - Exercici 2

Considera el següent prototip de la funció crc32, que calcula el Codi de Redundància Cíclica de 32 bits per a un missatge `M` compost de `N` bytes, fent servir una taula auxiliar `LUT` de 256 words.

```
unsigned int crc32(unsigned char M[], int N, unsigned int LUT[]);
```

El codi corresponent en assemblador MIPS és el següent:

```
crc32:
        nor     $v0, $zero, $zero
for:
        beq     $a1, $zero, fifor   # surt si N==0
        lbu     $t2, 0($a0)         # carrega un element de M
        andi    $t3, $v0, 0xFF
        xor     $t3, $t3, $t2
        sll     $t3, $t3, 2
        addu    $t4, $a2, $t3
        lw      $t5, 0($t4)         # carrega un element de LUT
        srl     $v0, $v0, 8
        xor     $v0, $v0, $t5
        addiu   $a0, $a0, 1
        addiu   $a1, $a1, -1        # N = N-1
        b       for
fifor:
        nor     $v0, $v0, $zero
        jr      $ra
```

Suposem que el missatge `M` consta de `N`=100 bytes, i que la funció `crc32` s'executa en un processador que dissipa 100W de potència, que funciona amb un rellotge de 2GHz i que té els següents CPI segons el tipus d'instrucció:

TIPUS	salt (salta)	salt (no salta)	load/store	altres
CPI	5	1	8	1

19-20Q1- Parcial - Exercici 2

a) (0,6 pts) Calcula quantes instruccions de cada tipus s'executen en la funció crc32, i quants cicles tarden. Calcula també el total d'instruccions i el total de cicles.

TIPUS	salt (salta)	salt (no salta)	load/store	altres	TOTAL
Núm. d'instruccions					
Núm. de cicles					

b) (0,6 pts) Calcula el temps d'execució de crc32 en segons

t_{exe} =

c) (0,3 pts) Calcula l'energia total consumida durant l'execució de `crc32`, en Joules

E =

Traducció

19-20Q1- Parcial - Exercici 6

Donada la següent declaració de variables globals d'un programa escrit en llenguatge C:

```
short a[3] = {1, 31, -2};
long long int b = -31
char c[5] = "ACDC";
short *d = &a[2];
```

a) (0,5 pts) Tradueix-la al llenguatge assemblador del MIPS.

b) (0,5 pts) Completa la següent taula amb el contingut de memòria en hexadecimal (sense el prefix 0x). Recorda que el codi ASCII de la 'A' és el 0x41. Les variables s'emmagatzemen a partir de l'adreça 0x10010000. Les posicions de memòria sense inicialitzar es deixen en blanc.

19-20Q1- Parcial - Exercici 6

c) (0,6 pts) Donat el següent codi en assemblador MIPS, indica quin és el valor final en hexadecimal del registre `$t0`:

```
lui     $t0, 0x1001
addiu   $t0, $t0, 8
la      $t1, d
lw      $t1, 0($t1)
subu    $t0, $t0, $t1
```

d) (0,6 pts) Tradueix a llenguatge assemblador del MIPS la següent sentència en C:

```
*(d - 2) = *d + 5;
```

19-20Q1- Parcial - Exercici 4

Completa la traducció a assemblador del MIPS de la funció g:

```
int g(short *q, short val) {
        return (*q == val);     /* Retorna 1 si són iguals, 0 altrament */
}
```

Subrutines

19-20Q1- Parcial - Exercici 5

Donades les següents declaracions de funcions en C:

```
int f1(short *ps1, short *ps2,
       int *pi);

int f2(short *x, int y) {
       short sv[5];
       int a[2], res;
       short s;
       s = *x + 3;
       res = y + f1(&sv[2], &s, a);
       return res;
}
```

Contesta els següents apartats que hi fan referència:

a) (0,5 pts) Completa la taula següent indicant on s'han d'emmagatzemar cadascun dels elements de f2: a la pila, a un registre temporal o a un registre segur. Posa una X a la columna corresponent, només pots triar una opció per a cada element de f2.

element de `f2 (en C)`	pila	registre temporal	registre segur
`sv`			
`a`			
`res`			
`s`			
`x`			
`y`			

19-20Q1- Parcial - Exercici 5

b) (0,5 pts) Dibuixa el bloc d'activació de `f2`, especificant-hi la posició on apunta el registre `$sp` un cop reservat l'espai corresponent a la pila, així com el nom de cada registre i/o variable, i la seva posició (desplaçament relatiu al `$sp`).

c) (0,8 pts) Tradueix a assemblador de MIPS la següent sentència del cos de la subrutina `f2`:

```
res = y + f1(&sv[2], &s, a);
```

Accés aleatori/seqüencial

19-20Q1- Parcial - Exercici 1

Considera les següents declaracions en C:

```
int G[64][64];
void f(int i, int j, int M[][64]) {
        while (j<64) {
                M[j][i] = G[0][63-j];
                j++;
        }
}
```

El següent fragment de codi conté la traducció de la funció f, aplicant-li l'optimització d'*accés seqüencial*. En concret, s'han usat els registres `$t0` i `$t1` com a punters per a recórrer els accessos a `G[0][63-j]`, i `M[j][i]`, respectivament. Completa les instruccions que falten als requadres per tal que la traducció sigui correcta:

19-20Q1-
Parcial - Exercici 1

```
f:      # Inicialitza el punter $t0 <- @G[0][63-j]
        la      $t0, [          ]
        sll     $t4, $a1, 2
        subu    $t0, $t0, $t4
        # Inicialitza el punter $t1 <- @M[j][i]
        [                       ]
        li      $t2, 64
while:
        bge     $a1, $t2, fi
        lw      $t4, 0($t0)
        sw      $t4, 0($t1)
        addiu   $t0, $t0, [     ]
        addiu   $t1, $t1, [     ]
        addiu   $a1, $a1, 1
        b       while
        jr      $ra
fi:
```

Subrutines Part 2

21-22Q1- Final - Exercici 4

Considera les següents declaracions de funcions en C:

```
short f(int *a, short b, char *c);
short examen(short *x, int y[], char *z) {
      short w[3];
      int p;

      w[0] = f(&p, *(x+1), z);
      w[1] = f(y+2, *x, z);
      w[2] = 3;

      return w[0] + w[1] + w[2];
}
```

Per a totes les variables i arguments de la rutina `examen`, indica si els guardaries a la pila, en registres segurs o en registres temporals. Justifica la teva resposta.

Tradueix la subrutina `examen` a MIPS, seguint les decisions de l'apartat anterior.

Excepcions i interrupcions

18-19Q2 - Exercici 4

	Afirmació	V	F
1.-	Si el bit EXL val 1, les interrupcions seran ignorades.		
2.-	Una excepció no pot ser atesa fins que la instrucció que l'ha causada hagi finalitzat.		
3.-	En un sistema amb memòria virtual, la mida total d'un programa i les seves dades poden excedir la capacitat de la memòria física.		
4.-	La divisió d'enters codificats en el format de Ca2 no pot produir overflow.		
5.-	En format de simple precisió IEEE-754 (32 bits), la codificació 0x00F00000 representa un número normalitzat.		
6.-	En una memòria cache amb política d'escriptura immediata sense assignació, un accés a la memòria cache pot implicar dos accesos a memòria principal.		
7.-	En una subrutina, una variable local de tipus enter sempre es guardarà en un registre.		
8.-	La rutina RSE de tractament d'excepcions del MIPS segueix les regles de l'ABI que s'estableixen per programar les subrutines.		
9.-	Al MIPS es detecta que un accés a memòria causa una fallada de pàgina consultant el bit V en el TLB.		
10.-	La codificació en excés de l'exponent en el format de coma flotant simplifica les operacions de comparació.		