Examen Final Segmentació i Paral.lelisme
17 de gener del 2002

Problema 1

Sigui el seguent programa:

```
for ( i = 1; i <= 100 ; i++) {
   if (A[i] ≠ 0) then X[j] = Y[i] / A[i];
}
...
```

i la seva traducció a llenguatge màquina:

```
     Mov  r10, #0        ;r10 ← 0
     Mov  r11, #100      ;r11 ← N
for: Load r0,A(r10)      ;r0 ← mem[A+r10]
     beq  r0, #0, fin    ;si r0 = 0 llavors salta a fin
     Load r1,Y(r10)      ;r1 ← mem[Y+r10]
     Div  r2,r1,r0       ;r2 ← r1 / r0
     Store X(r10),r2     ;mem[X+r10] ← r2
fin: Add  r10,r10,#8     ;r10 ← r10 + 8
     Sub  r11,r11,#1     ;r11 ← r11 - 1
     Bneq r11, #0, for   ;si r11 <> 0 llavors salta a for
     ...                 ;instruccions de despres del bucle
```

El programa es vol executar en un processador segmentat amb 5 etapes (F, D, E, M i W), com l'explicat a classe amb tots els bypass necessaris. Totes les instruccions d'aritmética entera ocupen un cicle de E. L'escriptura a l'etapa W dura tot el cicle de rellotge. Les instruccions de salt fan efectiu el salt (es a dir, coneixen la condició i l'adreça desti del salt) al final de l'etapa E. Supposseu que la probabilitat de que A[i] ≠ 0 es el 90%.

Es demana:

a) Si el processador implementa salt amb bloqueig, indiqueu quin es el temps d'execució del bucle `for` (no considereu l'execució dels dos `mov` inicials).

b) Si el processador implementa salt amb anul.lació (es a dir, és capaç d'invalidar l'execució de totes les instruccions iniciades si detecta que el salt es fa efectiu), indiqueu quin es el temps d'execució del bucle `for`.

c) Si el processador implementa salt retardat, reordena el codi per tal d'optimitzar el rendiment. (Nota: la reordenació només ha de preservar el valor de les variables existents en el programa d'alt nivell). Quin es el nou temps d'execució del bucle `for`.

Problema 2

Dados los dos fragmentos de código equivalentes:

```
     CÓDIGO 1                          CÓDIGO 2
-----------------------------    -----------------------------
                                          ...
          ...                             ld $1, #100
-----------------------------    -----------------------------
          ld $1, #100                     ld.f F1, #0
          ld.f F1, #0                     ld.f F2, #0
-----------------------------    -----------------------------
loop:     ld.f F3, a($1)         loop:    ld.f F3, a($1)
          add.f F1, F1, F3                ld.f F4, a+1($1)
-----------------------------    -----------------------------
          add $1, $1, 1                   add.f F1, F1, F3
          bneq $1, #100, loop             add.f F2, F2, F4
-----------------------------    -----------------------------
          ...                             add $1, $1, 2
                                          bneq $1, #100, loop
-----------------------------    -----------------------------
                                          add.f F1, F1, F2
                                          ...
```

donde Fx y $x representan los registros reales y enteros, respectivamente. Las instrucciones acabadas en .f son de aritmética real. El resto de instrucciones son de aritmética entera.

Suponer un procesador superescalar de ancho 2 con el mecanismo de Tomasulo, que nunca bloquea la busqueda de instrucciones porque suponemos ilimitado el número de estaciones de reserva y de registros de renombre (Rename Buffer). La segmentación incluye las etapas FDIE...EWC para las instrucciones aritméticas y FDIEMWC para las de acceso a memoria (siendo F: fetch, D: decode, I: issue, E: execute, M: memory, W: write y C: committ).

El procesador produce la búsqueda de 2 instrucciones durante la etapa de fetch. Las lineas horizontales en el código anterior representan las fronteras de las lineas de cache en las que se encuentran almacenadas las instrucciones. El procesador tambien dispone de un predictor de saltos ideal (es decir, que siempre acierta en la predicción), que actua en la etapa F.

El processador dispone de una unidad funcional de aritmética entera (latencia 1 ciclo), una unidad funcional de aritmetica real (latencia 4 ciclos, segmentados) y un puerto de acceso a memoria con su sumador para el cálculo de direcciones (1 ciclo en la etapa E y 1 ciclo en la etapa M).

La etapa de committ se ejecuta en orden, en concreto siguiendo el orden de fetch (es decir, dos instrucciones que han realizado el fetch en ciclos seguidos también realizaran el committ en ciclos seguidos). Se permite el committ de hasta 2 instrucciones por ciclo.

Con estas especificaciones, se pide:

a) Dibujar el cronograma de ejecución de las tres primeras iteraciones del bucle para cada uno de los códigos anteriores. (No hace falta dibujar la ejecución de las instrucciones previas al bucle).

b) A partir de estos cronogramas, calcular cuantos ciclos durará la ejecución completa de cada uno de los códigos?