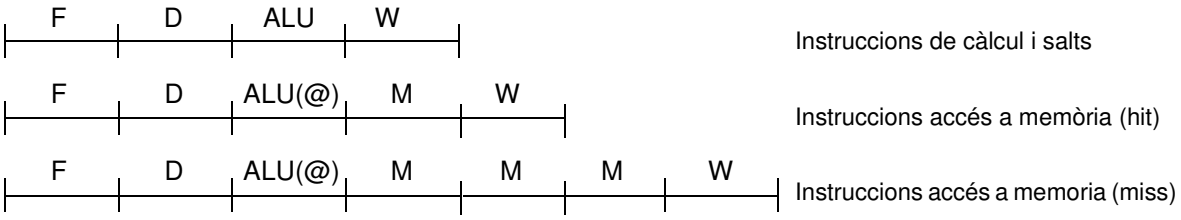
Examen Final Segmentació i Paral.lelisme
11 de gener del 2001

Problema 1 (primer parcial):

Tenim un processador segmentat multicicle amb les següents etapes:



El repertori d'instruccions és l'explicat a classe. La memòria de dades es pot considerar com una unitat funcional segmentada amb temps d'operació de 1 cicle de rellotge en cas de *hit* o 3 cicles de rellotge en cas de *miss*. El processador disposa de tots els curtcircuits (bypass) necessaris per eliminar els perills de dades. L'escriptura de resultats en el banc de registres ocupa tot un cicle de rellotge. L'escriptura en el banc de registres no té perque seguir l'ordre del programa. En cas de salt condicional, el processador es bloqueja fins que s'obté el resultat del salt (el càlcul de l'adreça destí i l'avaluació de la condició es fa en l'etapa ALU). La memòria cache es prou gran i completament associativa. Inicialment està buida. A cada linia de cache hi caben 2 paraules (enters).

Es demana:

Donat el codi següent en alt nivell:

```
for (i = 1; i <= 100; i++)
     Z(i) = X(i) + Y(i) + Z(i);
```

Dibuixa el cronograma d'execució pel següent codi generat pel compilador i determina el número total de cicles per executar cadascuna de les iteracions del bucle.

```
     Mov  r10, #100;r10 ← 100
for: Load r0,X(r10); r0 ← mem[X+r10]
     Load r1,Y(r10); r1 ← mem[Y+r10]
     Add  r2,r0,r1; r2 ← r0 + r1
     Load r3,Z(r10); r3 ← mem[Z+r10]
     Add  r4,r2,r3; r4 ← r2 + r3
     Store Z(r10),r4; mem[Z+r10] ← r4
     Sub  r10,r10,#1; r10 ← r10 - 1
     Bnez r10, for;si r10 <> 0 llavors salta a for
```

Problema 2 (segon parcial):

Dada la siguiente secuencia de instrucciones en lenguaje máquina:

```
----------------------------
          ...
          ...
          ld $6, #100
          ld.f F1 ← b($2)
----------------------------
loop:     ld.f F2 ← a($2)
          add.f F1 ← F1, F2
          mul.f F1 ← F1, F7
          add $2 ← $2, #1
----------------------------
          st.f b($2), F1
          sub $6 ← $6, #1
          jnz loop
          ...
----------------------------
```

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 SS 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 SS 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 processador dispone de una unidad funcional de aritmética entera (latencia 1 ciclo), un sumador y un multiplicador 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).

El procesador tambien dispone de un predictor de saltos ideal (es decir, que siempre acierta en la predicción), que actua en la etapa F.

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 SS instrucciones por ciclo.

Se pide:

Dibujar el cronograma de ejecución de las tres primeras iteraciones del bucle para SS=2 y para SS=4.