Static Multiple Issue Processor

Son procesadores que tienen la capacidad de terminar de ejecutar mas de una instruccion por ciclo

Las cuales deben ser empaquetadas por el compilador

- **Issue Packet**

Dependencia de datos - Hazard de datos

Las dependencias de datos es una caracteristica del programa
Se detectan como hazard y si genera o no un stall, es propiedad de la organizacion del pipeline

Implica

- Posibilidad de un hazard
- El orden es que se debe calcular el resultado
- Un limite superior en cuanto se puede explotar el paralelismo
  Puede superarse
- Manteniendo la dependencia pero evitando el hazard
- Eliminando la dependencia al modificar el codigo
- Forwarding, forwarding-stall, modificar el codigo (su orden su forma, etc), NOPS (stalls)

La dependencia de datos se puede dar a traves de la memoria o registros
Registros

- Relativamente sencillo de detectar
- se ven en las instrucciones
  Memoria
- Dificil de detectar
- Dos instrucciones pueden referirse a la misma posicion pero parecer diferente
- Ejemplo -> [x4, 100] y [x6, 20]
  - La misma instruccion ejecutada en distintas partes del codigo puede apuntar a posiciones distintas
  - Como no se el valor de x4 y x6 podriamos estar accediendo a la misma posicion de memoria
  - Estos accesos se deben hacer en orden porque podria estar teniendo una dependencia de datos atraves de la memoria

- Ej: {stur x0, [x4,100]; ldur x1, [x6, 20]}

Ejemplo

```
1>  L:  ldur X0, [X1, #0]     //X0=array element, X1=element addr
2>      add X0, X0, X2        //add scalar in X2 and save in X0
3>      stur X0, [x1, #0]     //store result
4>      subi x1, x1, #8       //decrement pointer 8 bytes
5>      cbz  x1, L            //branch si x1 es cero
```

Tipo Hazard i1,i2, dato
Datos 1, 2, X0
Datos 2, 3, X0
Datos cond 1, 4, X1
Datos 4, 5, X1

RAW Dependencias real de datos (el que venimos viendo) ^

- La instruccion i es dependiente en datos con la instruccion j
- Son dependencias que no puedo evitar

WAW Dependencias de nombre

- Dos instrucciones usan el mismo registro o posicion de memoria pero no hay flujo real de datos
- Ejemplo
- **ADD X0, X1, X2**
  ADD X0, X3, X4
  - Antes aca no pasaba nada pues se ejecutaban secuencialmente y se pisaba
- Pero en el micro con recursos duplicados si hay problema (Cortex-A53)
  - **Problema**
    - ADD X0, X1, X2
      ADD X0, X3, X4 (Ejemplo webon que nadie haria pero el procesador es un problema)
    - Puede pasar que esas dos instrucciones se ejecuten en en paralelo en las dos ALU
    - Queriendo escribir en X0 en el mismo ciclo de clock
    - yo necesito que X0 acabe como la suma entre X3 y X4
  - **Solucion**
    - Simplemente renombro el registro

- ADD X0, X1, X2
- ADD X10, X3, X4

Ejemplo RAW y WAW

Ejemplo de *Hazard* de datos:

- **RAW**:

```
add X0, X1, X2          |   add X0, X1, X2
add X3, X4, X0          |   stur X3, [X0, #0]
```

- **WAW**:

```
add X0, X1, X2          |   add X0, X1, X2
add X0, X3, X4          |   ldur X0, [X3, #0]
```

Dependencias de datos condicional

- Es una dependencia que esta condicionada al resultado de un salto

```
1>        sub x2,x3,x4
2>        cbnz x2,L1
3>        ldur x1, [X2, #0]
4>   L1:  add x3, x2, x5
```

-
- Si no se toma
  - => 1, 3, X2
- Si se toma
  - => 1, 4, X2

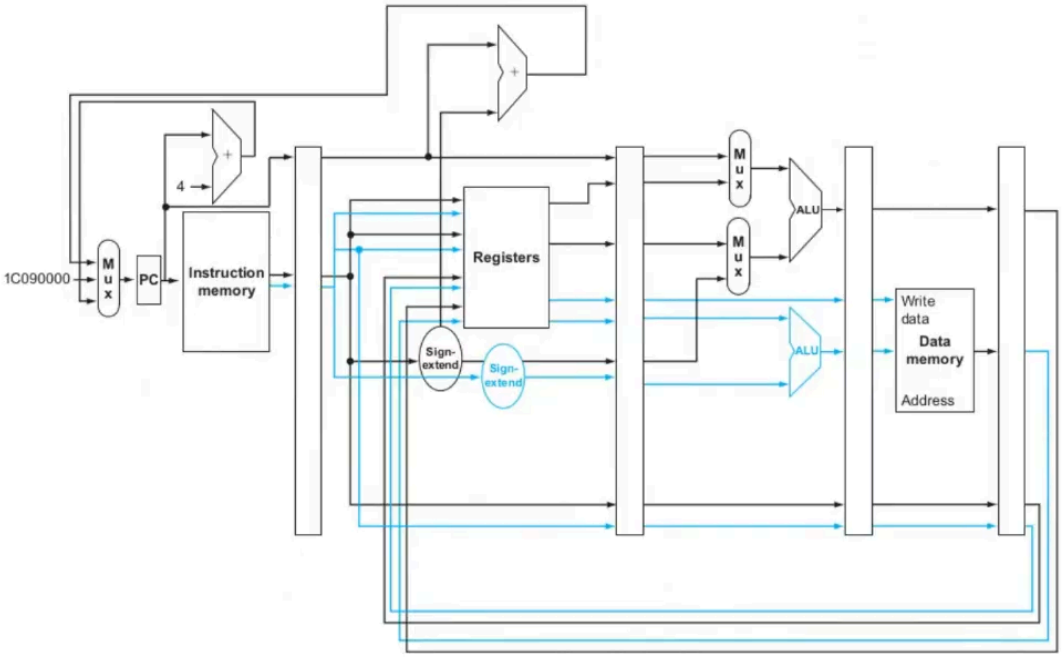
LEGv8 Static Two-Issue Datapath

El micro LEGV8 Tiene dos etapas distintas

- Execute
- Memory

  Entonces duplicando estos recursos podemos tener la capacidad de ejecutar 2 instrucciones a la vez

LEGv8 Static Two-Issue Datapath



- Duplicamos
  - **SignExtend**
  - **Alu** (Pero es un sumador)
  - **Acceso al bloque de registro** (en vez de sacar 2 sacamos 4)
- Entonces en paralelo
  - **"En la parte de arriba del micro"** Ejecutamos instrucciones que no requieran acceso a memoria (Tipos R- Branch)
  - **"En la parte de abajo del micro"** Ejecutamos las instrucciones que requieren acceso a memoria (LOAD - STORE)
- **Issue Packet**
  - Debe haber una instruccion Tipo r o Branch y una Load o Stur ()
  - Ejecutar dos instrucciones por ciclo requiere hacer fetch y decode de instr de 64 bits
    - Cada instr de 32 bits
  - => El PC se incrementa de a 8
  - **Si una de las Issue no puede utilizarse porque existe una dependencia se la debe acompañar de un nop**

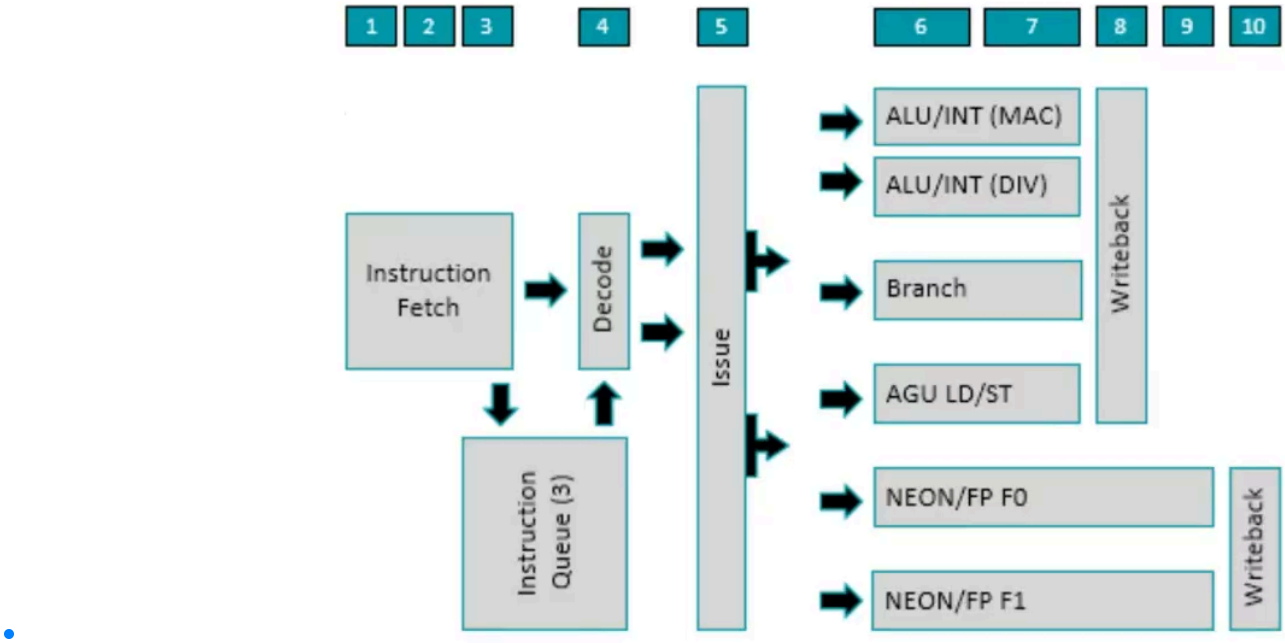
LEGv8 Static Two-Issue Pipe Stages

LEGv8 Static Two-Issue Pipe Stages

Instruction type	Pipe stages							
ALU or branch instruction	IF	ID	EX	MEM	WB			
Load or store instruction	IF	ID	EX	MEM	WB			
ALU or branch instruction		IF	ID	EX	MEM	WB		
Load or store instruction		IF	ID	EX	MEM	WB		
ALU or branch instruction			IF	ID	EX	MEM	WB	
Load or store instruction			IF	ID	EX	MEM	WB	
ALU or branch instruction				IF	ID	EX	MEM	WB
Load or store instruction				IF	ID	EX	MEM	WB

- Estamos haciendo fetcheando 2 instrucciones (TipoR/Branch y LOAD/STUR)
- 2 instrucciones de esas son un **Packet Issue**
- **Se puede usar forwarding entre Packet Issues pero no entre el mismo**
- LDUR -> Tiene el dato disponible en la etapa de memory, asi que vendria primero
  - =>se obtendria el valor de la LDUR primero que la del R-FORMAT que recien lo tiene en WB
- **STUR** -> Guarda el dato recien en memoria
  - => Se obtiene primero el valor del STUR antes que el R-format
- Dos instrucciones no pueden procesarse juntas en el mismo **Issue Packet si una depende de la otra** (No hay forwarding entre issues)

La ARM CORTEX-A53 hace esto



-
  - Hace Issue de dos instrucciones en paralelo
  - Hace fetch -> Decodifica las dos instrucciones -> hace Issue
  - Tiene 2 ALU, o sea puedo ejecutar dos instrucciones que usen ALU (**En paralelo si no hay dependencia de datos entre ellas**)
    - ALU de multiplicacion y acumulacion
    - ALU de division
  - Tiene una unidad de Branch
  - Tiene la unidad para los que acceden a memoria
  - Luego tenemos para ejecutar dos instrucciones de punto flotante
  - No se puede combinar dos instrucciones de P. flotante y P. entero

Evadir dependencia de datos en Two Issue

Sabemos que por Issue hay 2 instrucciones

- [R-Format or Branch | STUR or LDUR] <- Issue Packet

Entonces en nuestro codigo se pueden presentar Hazard de datos

- **RAW**
- **WAW**
- **Condicional RAW**

Formas de evitarlo en Two Issue

- **Forwarding** <=> No pertenece al mismo Issue Packet
- **Stall** entre Issue Packets (nop no hace falta que ambas instrucciones sea nop)
- **Modificar orden del codigo** y el codigo en si<=> La consigna no especifica lo contrario
  - Modificar offsets, nombres, etc
  - Ya que cambiar el orden puede significar cambiar el codigo
- **Loop unrolling**
  - Es una tecnica estatica que permite como **desesamblar un Loop** para tener una nueva ejecucion del codigo
  - Donde en lo que era la ejecucion de una iteracion
  - Se puede Duplicar, triplicar, cuadruplicar, etc. la ejecucion de ese codigo en esa iteracion
  - Siempre teniendo en cuenta el multiplo de las N iteraciones
    - Si N es multiplo de 2 => duplicamos lo mismo en una sola iteracion
    - => pasamos a iterar N/2 veces
  - **Problemas**
  - **Hazard de datos con el mismo codigo de antes**
    - Si la actividad permite cambiar el nombre de los registros resultado =>
    - El hecho de repetir el mismo codigo de la iteracion para tener menos iteraciones
    - Pues van a tener los mismos nombres de registro
    - **Solucion** => **Register rename**
      - Renombrar los registros que no afecten resultados y accesos a memoria
  - **Instrucciones que se pueden eliminar modificando el codigo**
    - Puede suceder que haya por ejemplo una instruccion que incrementa en 8 para acceder al siguiente elemento del array
    - Pues imaginemos que "unrolleamos 2 veces" el loop
    - Entonces se accederia al primer elemento le sumaba 8 y luego accedia al siguiente y tambien le sumaba 8 para la siguiente iteracion
    - Pero mejor seria
    - eliminamos la primera suma de 8
    - Modificamos el offset de la siguiente vez
    - y sumamos 16 a ese registro al ultimo para la siguiente iteracion
  - **Two Issue**
    - Luego de hacer la nueva ejecucion tenemos que pasarme al **Two Issue** para ver como quedaria realmente la ejecucion con las dependencias, en el pipeline

- Siempre que no nos digan lo contrario podemos modificar offsets, registros, y si cambiar la posicion de algo significa cambiar offsets tambie nlo hacemos