RISC: Reduced Instruction Set Computing (set reducido de instrucciones)

Mips: conjunto reducido de instrucciones

El hardware se organiza en Segmentación de cauce (Pipeline), donde se realiza más de una instrucción en cascada, de
forma paralela.

Hasta ahora se vio las instrucciones de forma secuencial, siguiendo un orden específico:

-Instrucción 0
-Instrucción 1
-Instrucción 2
...
-Instrucción N

En la ejecución segmentada, las instrucciones se ejecutan en fases (5) y a medida que ocurre un ciclo se ejecuta otra
instrucción en simultáneo:

-Instrucción 0: FASE 1, FASE 2, FASE 3, FASE 4, FASE 5
-Instrucción 1: FASE 1, FASE 2, FASE 3, FASE 4, FASE 5
-Instrucción 2: FASE 1, FASE 2, FASE 3, FASE 4, FASE 5
-Instrucción 3: FASE 1, FASE 2, FASE 3, FASE 4, FASE 5
-Instrucción 4: FASE 1, FASE 2, FASE 3, FASE 4, FASE 5

Es decir a medida que se avanza en las fases se va ejecutando en paralelo otra instrucción (max 5).

Fases:

1-Amarillo-IF: Instruction Fetch (toma de la instrucción, se accede a memoria por la instrucción, incrementa el PC).

2-Azul-ID: Instruction Decoding (decodifica la instrucción, se accede al banco de registros por los operandos, se calcula el valor del operando inmediato, si es un salto se calcula el destino del mismo (se hacen según lo requerido, pueden no ocurrir)).

3-Rojo-EX: Execution (si es una operación se usa la ALU, si es acceso a memoria se calcula la dirección, si es un salto se ejecuta (se modifica el PC)).

4-Verde-ME: Memory Access (Si es un acceso a memoria se lee/escribe el dato).

5-Violeta-WB: Write Back (se almacena el resultado (si hay) en los registros).

Todas las etapas tardan lo mismo excepto la EX pues dependerá de cuál operación se realice.

Atascos: se pueden dar situaciones que impiden que la instrucción se ejecute en el ciclo que le corresponde.

-Estructurales: provocado por conflictos de recursos.
-Dependencia de datos: RAW (Read After Write) WAR (Write After Read) WAW (Write After Write).
-Dependencia de control: Provocados al esperar la decisión de otra instrucción anterior. (Branch taken).

RAW: Se produce cuando una instrucción necesita leer un dato que todavía no está disponible.

Es decir, un registro necesita otro registro que está escribiendo una instrucción anterior, hasta que el registro anterior no llegue a fase WB (write back) el registro actual permanecerá en espera.

Soluciones al RAW:

Software:

- **NOP:** se agrega una instrucción que no haga nada, solo dé tiempo a la etapa (nop). El contra es que esto aumenta los ciclos de instrucción.

- **Ordenar sentencias:** reordenar las líneas para que no ocurra el RAW.

Hardware:

- **Forwarding:** Entre EX - ME - WB existen buffers para hacer adelantos de datos. De esta forma no hace falta esperar las etapas MEM y WB para usar los datos de registros anteriores. Permite postergar la necesidad de los operandos, es decir, no es necesario usar NOP en ningún momento.

Dependencias de control:

Atasco por saltos.

Incondicionales: salta siempre.

Condicionales: salta dependiendo de que se cumpla una condición.

Branch Target Buffer(BTB): predice el resultado de la condición (1 o 0) y decide si salta (1) o no (0). Según como falle va a contar uno de estos atascos. Cuando falle cambia el bit que indica si salta o no.

- **Branch Taken Stall:** la ejecución de una instrucción se atasca, cuando esta depende de una condición previa que aún no ha emitido un resultado.

- **Branch Misprediction Stalls:** cuando la predicción falla, cuenta como atasco.

Delay Slot: Consiste en ejecutar siempre la siguiente instrucción a un salto. Se busca ordenar las instrucciones que no dependen del salto de forma que se aprovecha la instrucción sin tener atascos de salto. Como último recurso se puede usar NOP.

Mips:

-32 registros de uso general(R0..R31) - R0: siempre vale 0, no modificable.
-32 registros de punto flotante (F0..F31).
-Definición de variables: .data (establece el bloque de variables).
    -Tipos de variables: .byte (1 byte) / .word16 (2 bytes) / .word32 (4 bytes) / .word (8 bytes).

-Definición de código del programa: .code (bloque de lo que es código).

-Maquina de 3 direcciones: [COP] - Dir Result - Dir Op1 - Dir Op2.
    +ejemplo: DADD R1,R2,R3 (sumará R2 y R3, el resultado irá a R1).

-En vez de H para diferenciar un Hexadecimal, se usa 0x previo al numero
    +ejemplo: 0xFF.

-Todas las operaciones aritmetico-lógicas siempre se hacen con registros.
-Las variables antes de usarse deben cargarse siempre al registro.
-Se deben especificar los desplazamientos (Rn) siendo N el número de desplazamientos deseado.
    +ejemplo: LD R1,VAR(R0) -> Load en R1, var + 0 desplazamientos (o sea cargar solo var).
-Formato de archivo (.s).
-Direcciones de registros y valores expresados en hexadecimal.

1) Muchas instrucciones comunes en procesadores con arquitectura CISC no forman parte del repertorio de instrucciones del MIPS64, pero pueden implementarse haciendo uso de una única instrucción. Evaluar las siguientes instrucciones, indicar qué tarea realizan y cuál sería su equivalente en lenguaje assembly del x86.

a) dadd r1, r2, r0 (r0 es necesario porque si o si debe haber una tercera dirección en la instrucción)

sumar r2 y r0 y guardar el resultado en r1
en X86: MOV r1,r2

b) daddi r3, r0, 5

guarda en r3, la suma de r0 y el valor directo 5
en x86: ADD r3,5

c) dsub r4, r4, r4

guarda en r4 el resultado de restar r4 por sí mismo
en x86: SUB r4,r4

d) daddi r5, r5, -1

guardar en r5 el resultado de sumar -1 a r5
en x86: DEC r5

e) xori r6, r6, 0xffffffffffffffff

guardar en r6 el resultado de un XOR entre el registro r6 y el valor inmediato ff..ff
en x86: NOT r6 (en mips no existe NOT)