ESTRUCTURA DE COMPUTADORES
MEMORIA PRÁCTICA 2

Titulación: Ingeniería informática y Administración de empresas

Hecha por:

Paulo Álvarez Da Costa

- NIA ➜ 100475757
- Dirección de correo electrónico ➜ 100475757@alumnos.uc3m.es

Juan Marín Olloqui

- NIA ➜ 100475063
- Dirección de correo electrónico ➜ 100475063@alumnos.uc3m.es

Índice de contenidos

EJERCICIO 1

Nombre de la instrucción	Diseño de las instrucciones solicitadas en lenguaje RT	Señales de control que se han de activar en cada ciclo de cada instrucción	Decisiones de diseño
la R1, U32	**CICLOS** **1:** MAR ‹= U32 **2:** MBR ‹= MEM[MAR] **3:** R1 ‹= U32	**CICLOS** **1:** T2, C0 **2:** TA, R, BW, M1, C1 **3:** M2, C2, T1, LC, RC, A0, B	Hemos guardado el address, almacenado en U32, en el registro de dirección de memoria(MAR), para luego acceder a memoria y cargar el dato en MBR. Luego, en el registro r1 hemos guardado la dirección que hay en U32 y activando "A0" "B" y "C" = 0 volvemos al fetch. (La vuelta al fetch se hace al final de todas las instrucciones)
sc R1, R2, (R3)	**CICLOS** **1:** MAR ‹= r3 **2:** MBR ‹= r1 **3:** MEM[MAR] ‹= MBR	**CICLOS** **1:** RA, T9, C0 **2:** RA, T9, C1 **3:** BW, TA, TD, W	**Parte real** Cargamos en MAR el address de r3 y en MBR el contenido de

	4: MAR <= r3 + 4 **5:** MBR <= r2 **6:** MEM[MAR] <= MBR	**4:** RA, MB, COP, T6, C0 **5:** RA, T9, C1 **6:** BW, TA, TD, W, A0, B	r1. Luego guardamos el contenido de r1 en el address de memoria que es igual al contenido de r3. **Parte imaginaria** Igual que la parte real, solo que cambiando el registro r1 por r2 y sumando 4 al registro r3 en la ALU antes de cargar la dirección en MAR.
lc R1, R2, (R3)	**CICLOS** **1:** MAR <= r3 **2:** MBR <= MEM[MAR] **3:** r1 <= MBR **4:** MAR <= r3 + 4 **5:** MBR <= MEM[MAR] **6:** R2 <= MBR	**CICLOS** **1:** RA, T9, C0 **2:** TA, R, BW, M1, C1 **3:** T1, RC, LC, SE **4:** RA, MB, COP, T6, C0 **5:** TA, R, BW, M1, C1 **6:** T1, RC, LC, SE, A0, B	**Parte real** Cargamos en MAR la dirección de memoria de r3, luego leemos la dirección y cargamos el contenido de r3 en MBR. El contenido de r3 se carga en r1 activando la extensión de signo. **Parte imaginaria** Igual que la parte real, pero cambiando el registro r1 por el r2 y sumando 4 al contenido de r3 en la ALU antes de cargar

			la dirección en MAR.
addc R1, R2, R3, R4	**CICLOS** **1:** R1 <= R1 + R3 **2:** R2 <= R2 + R4	**CICLOS** **1:** RA, RB, COP, T6, RC, LC, SELP, M7, C7 **2:** RA, RB, COP, T6, RC, LC, A0, B	**Parte real** Sumamos el contenido del registro r1 al contenido de r3 y actualizamos los valores C, V, N, Z. **Parte imaginaria** Hacemos lo mismo que en la parte entera, pero sin actualizar los valores de C,V,N,Z y sumando el contenido de los registros r2 y r4.
mulc R1, R2, R3, R4	**CICLOS** **1:** RT1 <= r1 * r3 **2:** RT2 <= r2 * r4 **3:** MBR <= RT1 - RT2 **4:** RT1 <= r1 * r4 **5:** RT2 <= r2 * r3 **6:** r2 <= RT1 + RT2 **7:** r1 <= MBR	**CICLOS** **1:** RA, RB, COP, T6, C4, SE **2:** RA, RB, COP, T6, C5, SE **3:** MA, MB, COP, T6, C1, SELP, M7, C7 **4:** RA, RB, COP, MC, T6, C4, SE **5:** RA, RB, COP, T6, C5 SE **6:** MA, MB, COP, T6, RC, LC, SELP, M7, C7 **7:** T1, RC, LC, A0, B	**Parte imaginaria** Primero hemos multiplicado el contenido de r1 y r3 guardando el resultado en el registro temporal RT1. Luego hemos hecho lo mismo con los registros r2 y r4, guardando el resultado en el temporal RT2. Restamos los registros RT1 y RT2 en la ALU guardando

			el resultado en MBR. Como las operaciones modificando antes de tiempo el registro r1 serían distintas, hemos actualizado su valor al final de la parte imaginaria **Parte imaginaria** Igual que en la parte real, pero multiplicando primero el contenido de r1 y r4 seguido del de r2 y r3. La utilización de los registros temporales es la misma; y la operación en la ALU es de suma en vez de resta. Por último se transfiere el contenido de RT3 al registro r2. Finalmente transferimos el contenido de MBR a r1 (del banco de registros) a través del bus interno, activando T1.
beqc R1, R2	**CICLOS**	**CICLOS**	Primero, hemos

R3, R4, S6	**1:** MBR <= SR **2:** SR <= r1 - r3 **3.1:** if SR.Z != 1: SR <= MBR **3,2:** If SR.Z ==1: SR <= R2 - R4 **4.1:** if SR.Z != 1: SR <= MBR **4.2:** If SR.Z ==1 **5:** RT1 <= PC **6:** RT2 <= IR(OFFSET) **7:** PC <= RT1 + RT2 **8:** SR <= MBR	**1:** T8, C1 **2:** RA, RB, COP, SELP, M7, C7 **3:** B, C, MADDR **4:** RA, RB, COP, SELP, M7, C7 **5:** B, C, MADDR **6:** T2, C4 **7:** SE, SIZE, T3, C5 **8:** MA, MB, COP, T6, C2, A0, B **9:** T1, C7 **10:** A0, B	guardado el estado actual (SR) en MBR. La estrategia en la parte real es restar el contenido de r1 al de r3 y verificar si es igual a 0, lo que indicaría que ambos son iguales. En caso de que no lo sean (es decir Z no esté activo), saltamos a la etiqueta beqc1 y el programa termina, devolviendo el contenido guardado en MBR a SR. Si Z ==1, entonces hacemos la misma comprobación pero con la parte imaginaria, restando el contenido de r2 al de r4 y haciendo lo mismo que al principio. Aquí si Z==0, saltamos a la etiqueta beqc1 y actualizamos SR como antes; pero si es igual a 1, haremos lo siguiente: - Guardamos el valor del PC en el temporal RT1 - Guardamos el contenido de

			la etiqueta (offset) desde IR, guardándolo en RT2 - Hacemos la suma de RT1 y RT2, guardando el resultado en el PC Por último devolvemos el contenido de MBR a SR.
call U20	**CICLOS** **1:** BR[ra] <= PC **2:** PC <= U20	**CICLOS** **1:** T2, MR, RC, LC **2:** SIZE, T3, C2, A0, B	El funcionamiento de esta función es la mismo que el de **jal ra u20,** con la que estamos más familiarizados. Se carga la dirección actual del PC en el registro ra, que no sale del IR, sino que se encuentra en el registro 1 como indican en la práctica, para ello activamos MR. Por último, especificando el SE, SIZE, y OFFSET, sacamos del IR el contenido de u20, que

			cargamos en el PC.
ret	**CICLOS** **1:** PC ‹= BR[ra]	**CICLOS** **1:** MR, RA, T9, C2, A0, B	Esta función es igual a un **jr ra.** Pasamos por el bus interno mediante T9 el valor de ra que se encuentra en el registro 1, para cargarlo en el PC, con C2.
hcf	**CICLOS** **1:** SR ‹= 0 **2:** PC ‹= 0	**CICLOS** **1:** T11, C2 **2:** T2, C7	Con ExCode hemos cargado el valor inmediato 0 y llevado al bus interno con T11. Ese valor lo hemos cargado en SR y luego lo hemos llevado al PC.

EJERCICIO 2

Diferencias ventajas e inconvenientes y propuesta de mejora

RISC-V

Ventajas:

El código RISC-V es más simple y directo, lo que puede conducir a un mejor rendimiento y a una mayor comprensión, además estas instrucciones se ejecutan de manera eficiente en hardware dedicado, como hemos visto en clase

Incovenientes:

Puede haber menos flexibilidad para realizar operaciones específicas a nivel de microinstrucciones y la realización de cambios puede ser más difícil ya que está limitada por las instrucciones disponibles.

Propuestas de mejora:

Se podrían explorar técnicas de optimización de código para mejorar la eficiencia de las instrucciones RISC-V.

EXTENSIONES PROPUESTAS DEL EJERCICIO 1: MICROPROGRAMACIÓN

Ventajas:

Permite un control detallado a nivel de microinstrucciones, lo que genera una mayor flexibilidad en la implementación de operaciones específicas.

Incovenientes:

La microprogramación es más difícil de entender y mantener y puede haber un pérdida de rendimiento debido a la ejecución secuencial de microinstrucciones en comparación con las instrucciones nativas del RISC-V.

Propuestas de mejora:

Probablemente se podrían realizar mejoras en la eficiencia de las instrucciones para mejorar el rendimiento y se podrían explorar técnicas de compilación de alto nivel para generar automáticamente secuencias eficientes de microinstrucciones.

Tabla con los resultados de medición de número de ciclos

	Ciclos de reloj sin extensión	Ciclos de reloj con la extensión	Mejora
A == B	0x00000070	0x000000F5	No hay mejora
A != B	0x00000073	0x000000F8	No hay mejora

Si va a trabajar con números complejos , ¿Merece la pena disponer de la extensión del ejercicio 1?

Nuestros resultados no reflejan ninguna mejora significativa, y honestamente, esperábamos obtener resultados completamente diferentes. No estamos completamente seguros de los resultados, y reconocemos humildemente la posibilidad de haber pasado por alto algún pequeño detalle en la práctica, lo que podría haber llevado a una solución incorrecta. A pesar de estas posibles fallas, mantenemos confianza en nuestro planteamiento y aproximación.

Dejando a un lado las excusas, podemos concluir que "with_ext" podría ser preferible en términos de concisión y posiblemente rendimiento. Sin embargo, si la portabilidad y la simplicidad del código son consideraciones importantes, la implementación sin instrucciones microprogramadas (no_ext) podría ser más adecuada.

Conclusiones, problemas encontrados y estimación de tiempo empleado

Las conclusiones son bastante satisfactorias (por lo menos, antes de recibir la nota de la práctica). Al principio nos parecio muchísima información en muy poco tiempo pero poco a poco hemos ido progresando. La microprogramación nos ha ayudado a entender lo que verdaderamente está ocurriendo cuando ejecutas el código.

La mayoría de problemas estaban en las pruebas y en la revisión de tantísimo código. Nos hemos vuelto completamente locos.

La práctica nos ha llevado unas 45 horas aproximadamente.