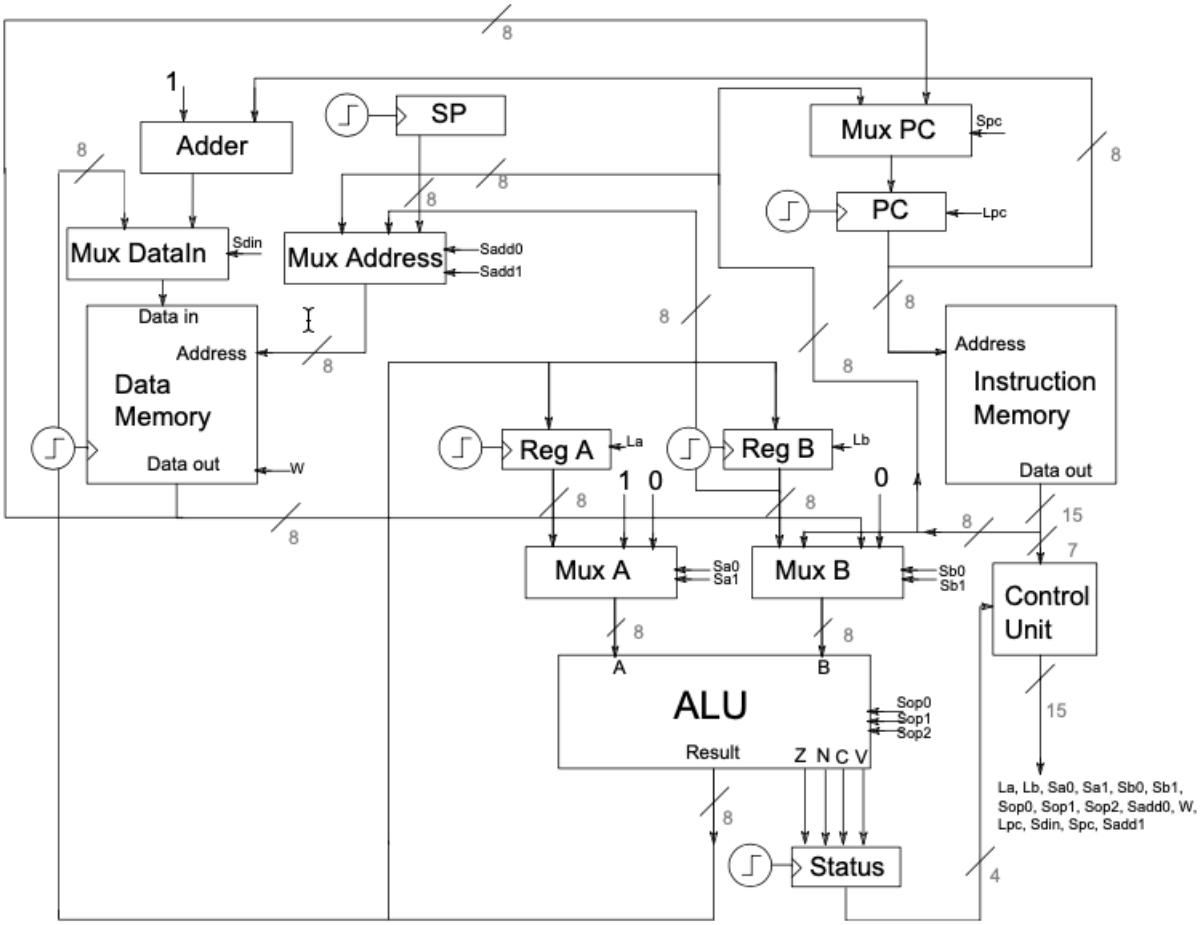
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
ESCUELA DE INGENIERÍA
PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

IIC2343 - Arquitectura de Computadores (II/2025)

Ayudantía 6

Ayudantes: Daniela Ríos (danielaarp@uc.cl), Alberto Maturana (alberto.maturana@uc.cl), Mario Rojas (mario.denzel@estudiante.uc.cl)

Computador básico



Pregunta 1: Subrutinas - Modificación de código

Imagina que te encuentras trabajando como programador(a) en la oficina del famoso juego desarrollado completamente en Assembly: *Rollercoaster Tycoon*. Luego de unas horas, te das cuenta que la letra T del teclado de tu computador ha dejado de funcionar, la que es de suma importancia al momento de escribir subrutinas por el uso de la instrucción `RET`. Adicionalmente, no cuentas con la opción de copiar y pegar texto, por lo que deberá desistir de su uso. Por el motivo anterior, se diseñan las siguientes tres nuevas instrucciones que te permitan simular parte del comportamiento de la instrucción `RET`.

1. `MOV B, SP`: Almacena el valor del registro `SP` en el registro `B`.
2. `JMP B`: Salta a la instrucción con dirección igual al valor almacenado en el registro `B`.
3. `JMP (B)`: Salta a la instrucción con dirección igual al valor almacenado en la memoria de datos en la dirección `B`.

Asuma que se implementan las instrucciones anteriores y modifique el siguiente fragmento de código de forma que no utilice la instrucción `RET`, pero que se llegue al mismo resultado. Luego, comente sobre su resultado: ¿Cumple la misma función que el original? ¿Existe alguna consideración a tener en cuenta?

```
DATA:
    number_of_rides 0

CODE:
    JMP main

    // Incrementa la cantidad de veces a la que se sube a la montaña rusa
    increase_number_of_rides:
        MOV A, (number_of_rides)
        ADD A, 1
        MOV (number_of_rides), A
        RET

    main:
        PUSH A
        CALL increase_number_of_rides
        POP A
```

Solución: Para simular el comportamiento de `RET` con las instrucciones nuevas, existen dos alternativas:

a) Reemplazar `RET` por:

1) `POP B`
2) `JMP B`

En este caso, el código cumple la función original y no se requiere ningún arreglo adicional.

b) Reemplazar `RET` por:

1) `MOV B, SP`
2) `INC B`
3) `JMP (B)`

En este caso, la ejecución no es exactamente igual a la original, dado que si bien se accede correctamente a la dirección de retorno, el valor de `SP` no es ajustado mediante un incremento. Esto genera que la ejecución de `POP A` almacene la dirección de retorno de la subrutina en `A` y no el valor respaldado al comienzo del programa.

Pregunta 2: Subrutinas - Computador básico

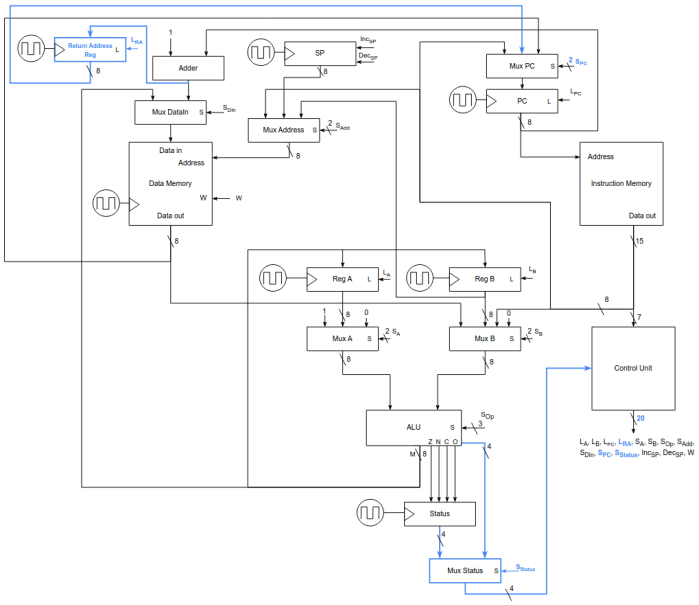
Ha sido contratado/a como programador/a en una bolsa de valores internacional. Al revisar la infraestructura tecnológica, nota que todos los procesadores se basan en el computador básico del curso, lo que limita el rendimiento necesario para reaccionar ante cambios del mercado. Su misión es diseñar una versión mejorada, llamada *FastBasicComputer*, que permita ejecutar programas con mayor velocidad y eficiencia. Para ello, deberá incorporar **instrucciones nuevas que disminuyan los ciclos de ejecución**. Deberá modificar la microarquitectura del computador básico para implementar las instrucciones nuevas y su funcionamiento. Para cada instrucción nueva, deberá incluir la combinación **completa** de señales que la ejecutan. Por cada señal de carga/escritura/incremento/decremento, deberá indicar si se activan (1) o no (0); en las señales de selección, deberá indicar el **nombre** de la entrada escogida ("-" si no afecta). Puede realizar todas las modificaciones en un solo diagrama.

(a) Implemente la instrucción `CALLFAST label`: Almacena `PC+1` en un **nuevo** registro llamado *Return Address Register* y **no modifica** el *stack*. Además, salta a la dirección `label`.

(b) Implemente la instrucción `RETFAST`: Carga en el contador `PC` el valor almacenado en el registro *Return Address Register* y **no modifica** el *stack*.

(c) Implemente la instrucción `JEQFAST A, Lit, label`: Ejecuta la operación `A - Lit` en la ALU y salta a la dirección asociada a `label` si, y solo si la *flag* `Z` **computada en el mismo ciclo** es igual a 1.

Solución: A continuación, se muestra el diagrama con las conexiones necesarias para ejecutar todas las instrucciones solicitadas, incluyendo una descripción de cada conexión añadida y lo que habilita.



- Se añade el registro *Return Address Register* con su señal de carga L_{RA}.
- La conexión entre la salida del *Adder* que computa `PC + 1` y la entrada de datos de *Return Address Register* permite cargar el valor en el registro con la señal L_{RA}.
- La conexión entre la salida de *Return Address Register* y una de las entradas del multiplexor `Mux PC` permite seleccionar el valor del registro como dato a cargar en el contador `PC`. Es importante destacar que, al tener tres valores a seleccionar en el multiplexor, es necesario incrementar a 2 bits la señal de selección S_{PC}.
- Se añade el multiplexor `Mux Status` y su señal de selección S_{Status} para seleccionar las *flags* a ser evaluadas por la Unidad de Control. Si provienen del registro `Status`, entonces se utilizan las *flags* computadas en el ciclo anterior; en otro caso, se utilizan las computadas en **el ciclo actual**.

A continuación, la tabla de señales resultante:

Instrucción	L_A	L_B	L_{PC}	L_{RA}	Inc_{SP}	Dec_{SP}	W	S_A	S_B	S_{OP}	S_{Add}	S_{DIn}	S_{PC}	S_{Status}
`CALLFAST label`	0	0	1	1	0	0	0	-	-	-	-	-	LIT	-
`RETFAST`	0	0	1	0	0	0	0	-	-	-	-	-	RA	-
`JEQFAST A, Lit, label`	0	0	Z == 1	0	0	0	0	A	LIT	SUB	-	-	LIT	ALU

Algunas consideraciones importantes:

- En la columna S_{PC}, `RA` representa la selección de la salida de *Return Address Register*.
- En la columna S_{Status}, `ALU` representa la selección de las *flags* proveniente de ella y no del registro `Status`.

- En la columna L_{PC}, no importa la notación, pero sí debe quedar claro que su valor no es por defecto 1, sino que depende de la *flag* `Z`.
- En la instrucción `JEQFAST A, Lit, label` existe una limitante importante: se debe usar el mismo literal tanto para la comparación como para el *label* dado que en la memoria de instrucciones solo se define uno. A modo de simplificación, se considerará correcto "asumir" que pueden ser distintos sin modificaciones adicionales de *hardware*, pero en la práctica su implementación requiere de la definición de dos literales por instrucción, lo que implicaría más cambios en la microarquitectura.

(d) Para evaluar la velocidad de *FastBasicComputer*, se realiza una comparativa entre un programa escrito en el *Assembly* del computador básico `(a)` y otro equivalente en la nueva arquitectura `(b)`. Indique los valores de los registros `A` y `B` al finalizar la ejecución del código en cada caso junto con la cantidad de ciclos de ejecución. Asuma que no se agregan instrucciones para inicializar la memoria RAM.

(a)

```
DATA:
    n 8
    is_even 0
CODE:
    MOV A, (n)
    CALL check_is_even
    JMP end
    check_is_even:
        MOV B, 1
        AND A, B
        CMP A, 0
        JEQ set_is_even
        end_check_is_even:
            RET
        set_is_even:
            MOV (is_even), B
            JMP end_check_is_even
    end:
        MOV A, (is_even)
```

(b)

```
DATA:
    n 8
    is_even 0
CODE:
    MOV A, (n)
    CALLFAST check_is_even
    JMP end
    check_is_even:
        MOV B, 1
        AND A, B
        JEQFAST A, 0, set_is_even
        end_check_is_even:
            RETFAST
        set_is_even:
            MOV (is_even), B
            JMP end_check_is_even
    end:
        MOV A, (is_even)
```

Solución: Los programas `(a)` y `(b)` determinan si la variable `n` es par (`is_even` = 1) o no (`is_even` = 0) a partir de la revisión de su bit menos significativo (`AND n, 1`). En ambos casos, el valor final de `A` será igual a 1, ya que es correspondiente al resultado de `is_even` para `n` igual a 8; mientras que el valor de `B` también será 1, dado que se le asigna este literal y no se modifica de nuevo. En términos de ejecución, el programa `(a)` toma un total de 12 ciclos ya que se debe considerar que se realiza el salto condicional `JEQ set_is_even` y que `RET` ejecuta en dos ciclos; `(b)`, en cambio, toma 10 ciclos al no requerir el uso de `CMP` y al tener un retorno de un solo ciclo con `RETFAST`.

(e) Las instrucciones `CALLFAST label` y `RETFAST` presentan un problema al ejecutar llamados anidados, situación que **no ocurre** con `CALL` y `RET`. En particular, se observa que no se realiza de forma correcta el retorno. Explique por qué ocurre esto.

Solución: Si se tienen llamados anidados, existirá más de una dirección de retorno (una por llamado). Al usar `CALL` y `RET`, las direcciones se respaldan en el *stack*, que simplemente incrementa su tamaño. Al usar `CALLFAST` y `RETFAST`, en cambio, se tiene un único registro que se **sobreescribe con cada llamado**, manteniendo así solo la última dirección de retorno.

Feedback ayudantía

Escanee el QR para entregar feedback sobre la ayudantía.