Microarquitectura del CPU

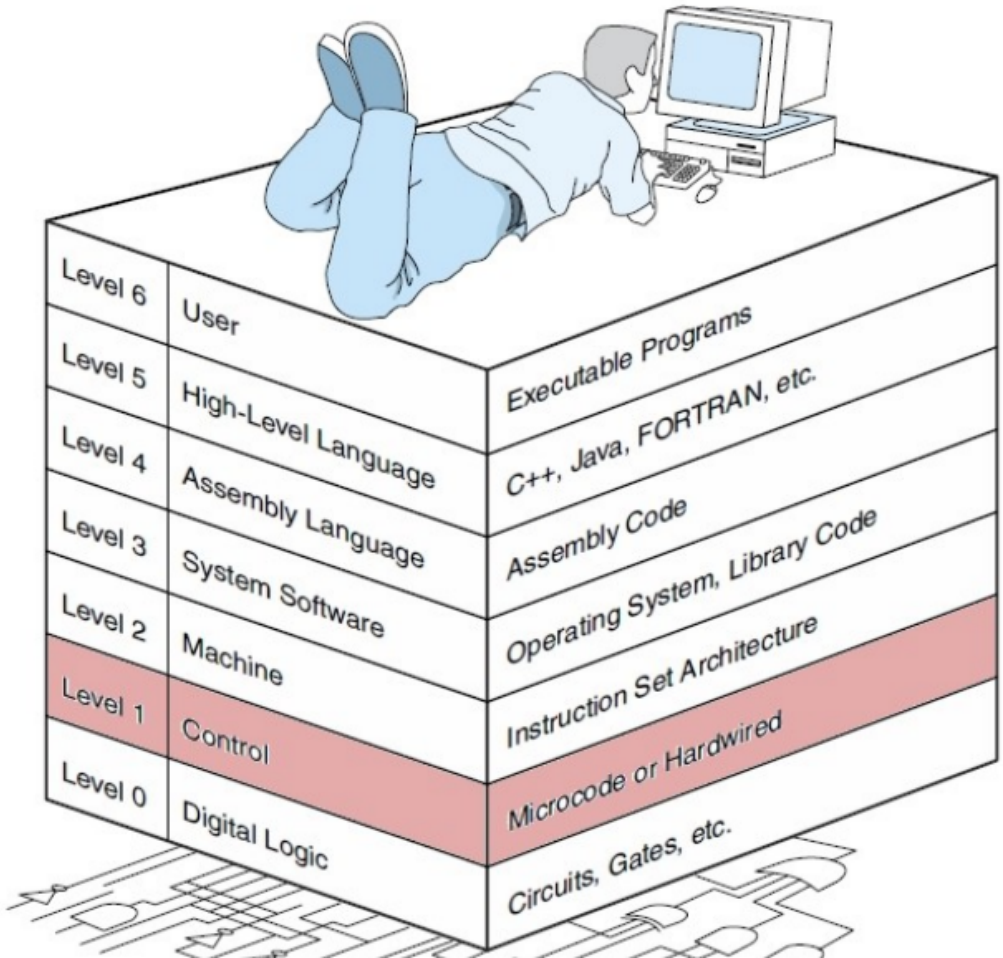
Práctica 7

Paula Verghelet
en base al material de Gus(tavo) Hurovich
y Esteban Mocskos et al. (1c 2017)

Organización del Computador I
Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

¿Dónde estamos?

- Circuitos Combinatorios - Sumadores, ALU...
- Circuitos Secuenciales - Registros...
- Unidad de Control (Microcode or Hardwired)
- Máquina de Orga1/Formato de Instrucción.



A modo de repaso

- Instrucción $\rightarrow$???? $\rightarrow$ Circuito

A modo de repaso

- Instrucción $\rightarrow$???? $\rightarrow$ Circuito

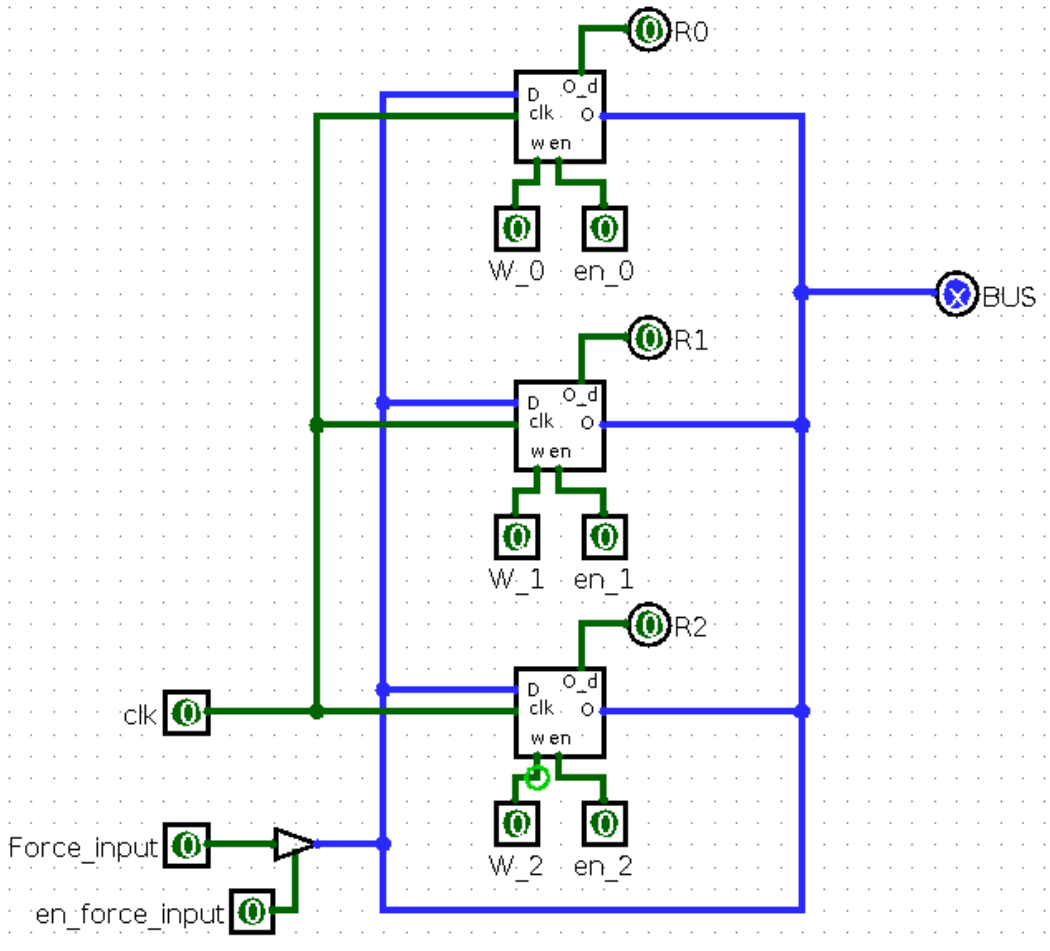
Microinstrucciones

- Cada instrucción está compuesta por microinstrucciones. Al igual que las partes del ciclo de instrucción.
- Estas microinstrucciones realizan pasos muy simples.
- Poner cosas en registros, activar circuitos, prender/apagar líneas.
- Entonces, la UC tiene que ejecutar estas microinstrucciones.
- Ejemplo: *Ciclo de Captación*

t_1: `MAR` $\leftarrow$`(PC)`
t_2: `MBR` $\leftarrow$`Memoria`
t_3: `PC` $\leftarrow$`(PC) + I`
`IR` $\leftarrow$ `(MBR)`

Un ejemplo que ya vimos

- ¿Y cómo hace?
- Mediante señales de control.



Con en_x y W_x
R1:= R0
R2:= R1

UC Cableada (Hardwired)

Unidad de Control (Control Unit)

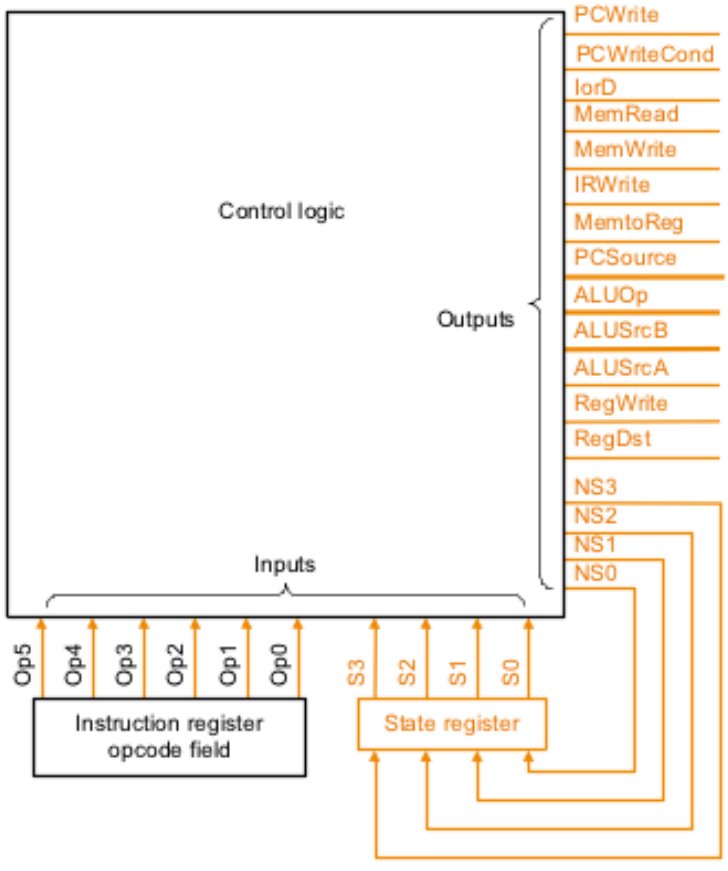
- inicialmente *cableada*.

UC Cableada (Hardwired)

Unidad de Control (Control Unit)

- inicialmente *cableada*.

Una UC con estados y señales



- En esta imagen las señales de control tienen nombres simbólicos y la UC retroalimenta su estado.
- Una UC cableada es más rápida.
- Pero poco flexible.
- ¿Y si queremos hacer algún cambio?
- ¿O corregir algo?

UC microprogramada[1]

- Una alternativa a la unidad de control cableada es la **unidad de control microprogramada**
- UC microprogramada: La lógica de la unidad de control viene dada por un *microprograma*
- Un microprograma consiste en una secuencia de instrucciones en un *lenguaje de microprogramación*
- Instrucciones muy elementales que especifican *microoperaciones*

UC microprogramada

Es un circuito lógico capaz de:

1. realizar el secuenciamiento de las microinstrucciones
2. generar las señales de control para ejecutar cada microinstrucción

- Las señales de control generadas por una microinstrucción se usan para producir transferencias entre registros y operaciones de la ALU (idem UC cableada).

[1] Organización y Arquitectura de Computadores - W. Stallings

UC microprogramada

- Es una buena solución para microarquitecturas que deben soportar cientos de instrucciones, modos, etc.
- Las señales se especifican simbólicamente usando microinstrucciones.
- Se define el formato de la microinstrucción, estructurado en campos.
- Luego, cada campo se asocia a un conjunto de señales.

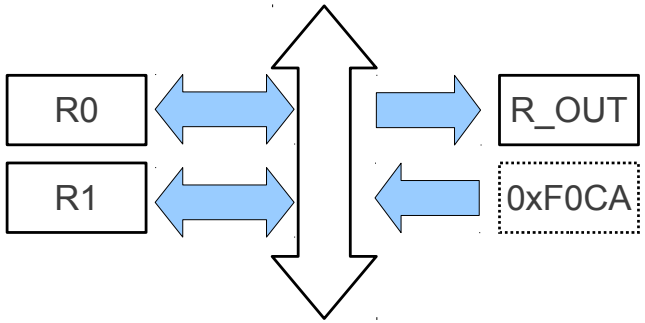
- A la hora de ejecutar una intrucción, una vez que está en IR:
- 1. Se toma el opcode como índice en una memoria de microinstrucciones.
- 2. Se carga esa dirección en un especie de "micro" PC.
- 3. Se ejecuta el microprograma.

Ejemplo de formato de microinstrucción

Campo	Función
ALU control	Qué debe hacer la ALU en este ciclo
SRC1	1er operando ALU
SRC2	2do operando ALU
Register Ctrl	Lectura/escritura de registros.
Memoria	Lectura/escritura de mem.
PC Write Ctrl	Escritura del PC.
Secuencia	Cómo elegir la próxima microinstrucción.

Lenguaje de microprogramación

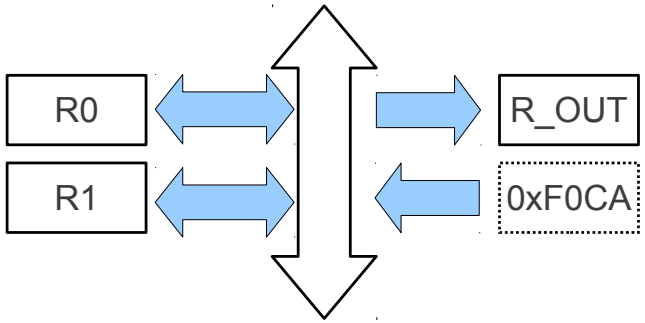
- Existen caminos (líneas) por donde van los datos.
- Podemos mover un dato de un registro a otro si hay un camino directo entre ellos
- Podemos asignar un valor **constante** (almacenada) a un registro.



VALE	NO VALE
R1:= R0	R0:=R_OUT
R_OUT:=R0	R1:=R_OUT
⋮	

Lenguaje de microprogramación

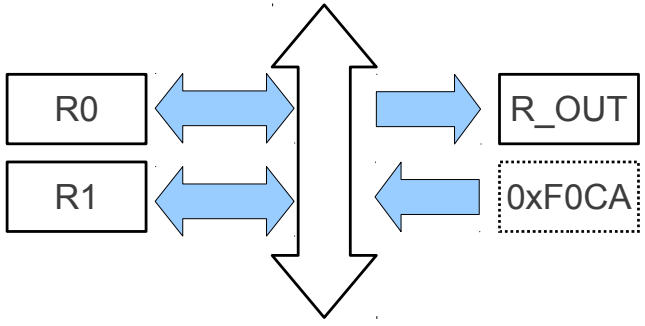
- Existen caminos (líneas) por donde van los datos.
- Podemos mover un dato de un registro a otro si hay un camino directo entre ellos
- Podemos asignar un valor **constante** (almacenada) a un registro.



VALE	NO VALE
R1:= R0	R0:=R_OUT
R_OUT:=R0	R1:=R_OUT
⋮	

Lenguaje de microprogramación

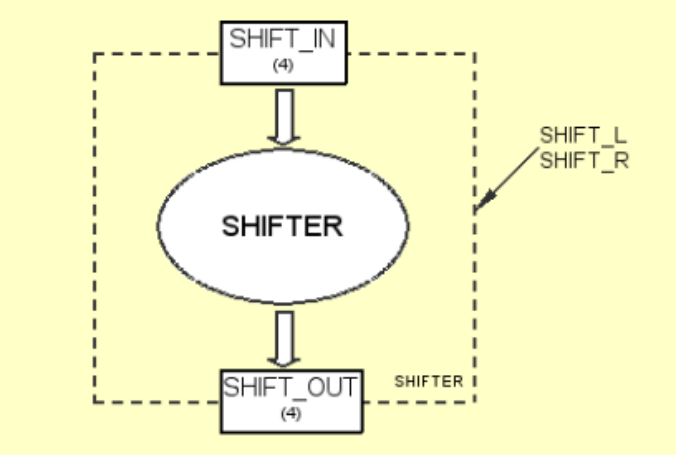
- Existen caminos (líneas) por donde van los datos.
- Podemos mover un dato de un registro a otro si hay un camino directo entre ellos
- Podemos asignar un valor **constante** (almacenada) a un registro.



VALE	NO VALE
R1:= R0	R0:=R_OUT
R_OUT:=R0	R1:=R_OUT
⋮	
R0:=0xF00A	R1:=x

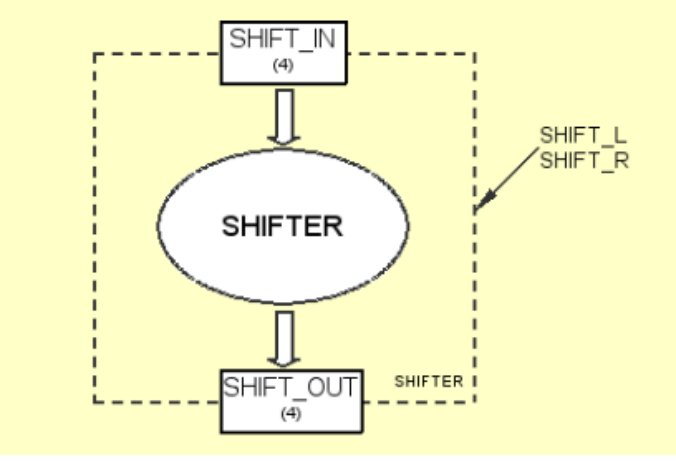
Lenguaje de microprogramación

- Activar señales
- Se pueden crear componentes

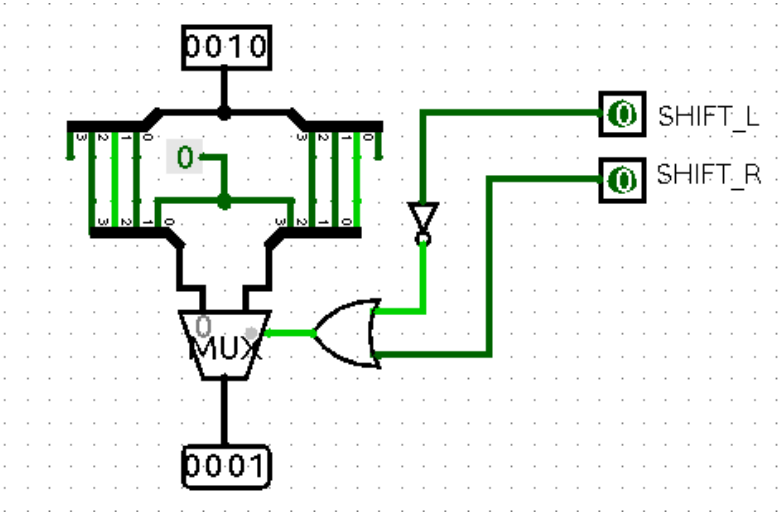


Lenguaje de microprogramación

- Activar señales
- Se pueden crear componentes



- ¡Hay que especificar su circuito!



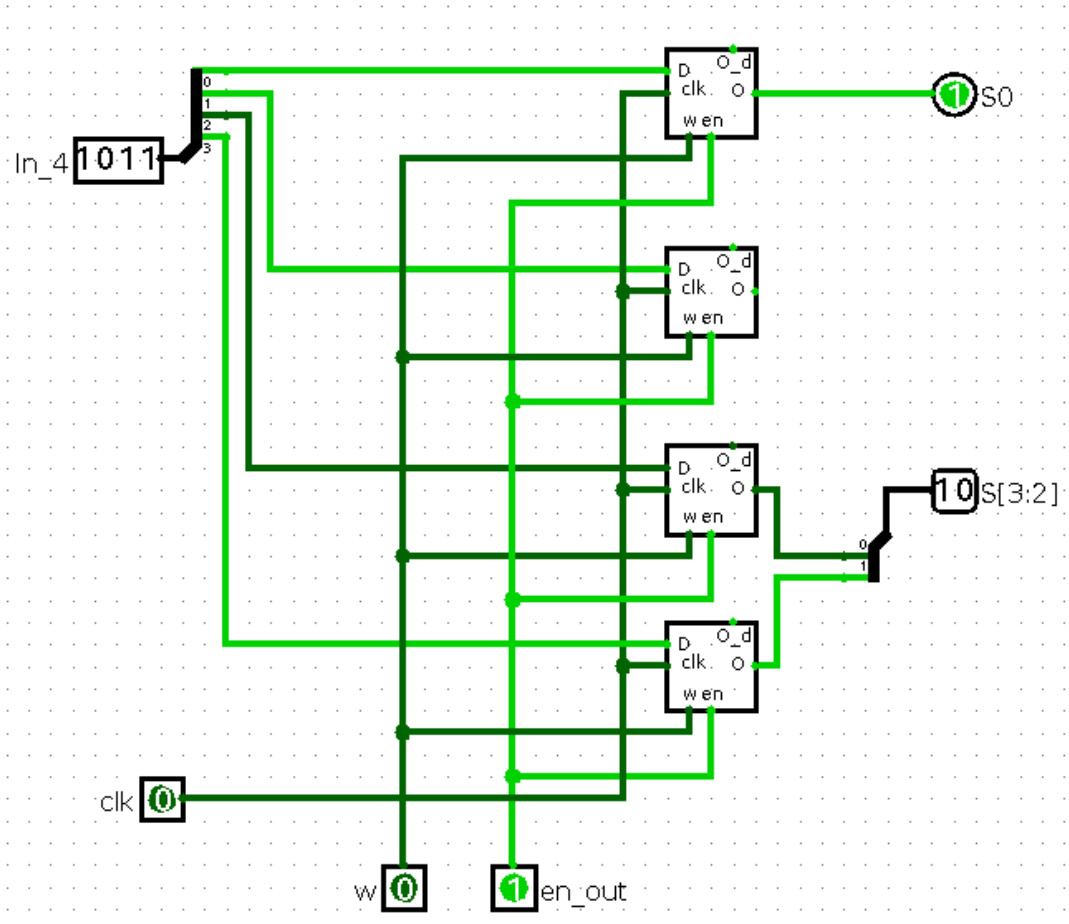
`SHIFT_L: Desplaza bits hacia izquierda`

`SHIFT_R: Desplaza bits hacia derecha`

Lenguaje de microprogramación

- Podemos ver alguna parte específica de un registro y usarla para algo...

- VALE
  R0[0]
  R2[3:2]



Lenguaje de microprogramación

- Podemos analizar un bit y actuar en consecuencia

```
if R0[0]=0
    R_OUT:=R3
else
    R_OUT:=R2
endif
```

Ejercicios

Ejercicio 1 - Máquina Orga1 - "Una nueva máquina"

Se cuentan con los siguientes circuitos:

- Una ALU con 2 registros de 16 bits (`ALU_IN1` y `ALU_IN2`) que usa de entradas y 5 registros que usa de salida: `ALU_OUT` de 16 bits y (`ALU_Z`, `ALU_N`, `ALU_C` y `ALU_V`) de 1 bit. Sus señales de control son:

Señal	Efecto
ALU_{add}	ALU_OUT := ALU_IN1 + ALU_IN2
ALU_{sub}	ALU_OUT := ALU_IN1 - ALU_IN2
ALU_{neg}	ALU_OUT := - ALU_IN1
ALU_{and}	ALU_OUT := ALU_IN1 AND ALU_IN2
ALU_{not}	ALU_OUT := NOT ALU_IN1

  Donde los flags se actualizan de forma usual.

- Un extensor de signo complemento a 2 (SIGN_EXT) con un registro de entrada de 8 *bits* (EXT_IN) y un registro de salida de 16 *bits* (EXT_OUT).
  Sus señales de control son:

Señal	Efecto
$SIGN\_EXT_{on}$	activa la operación de extensión de signo de 8 *bits* a 16 *bits*

Ejercicio 1 - Máquina Orga1 - "Una nueva máquina"

Se cuentan con los siguientes circuitos:

- Una ALU con 2 registros de 16 bits (`ALU_IN1` y `ALU_IN2`) que usa de entradas y 5 registros que usa de salida: `ALU_OUT` de 16 bits y (`ALU_Z`, `ALU_N`, `ALU_C` y `ALU_V`) de 1 bit. Sus señales de control son:

Señal	Efecto
ALU_{add}	ALU_OUT := ALU_IN1 + ALU_IN2
ALU_{sub}	ALU_OUT := ALU_IN1 - ALU_IN2
ALU_{neg}	ALU_OUT := - ALU_IN1
ALU_{and}	ALU_OUT := ALU_IN1 AND ALU_IN2
ALU_{not}	ALU_OUT := NOT ALU_IN1

  Donde los flags se actualizan de forma usual.

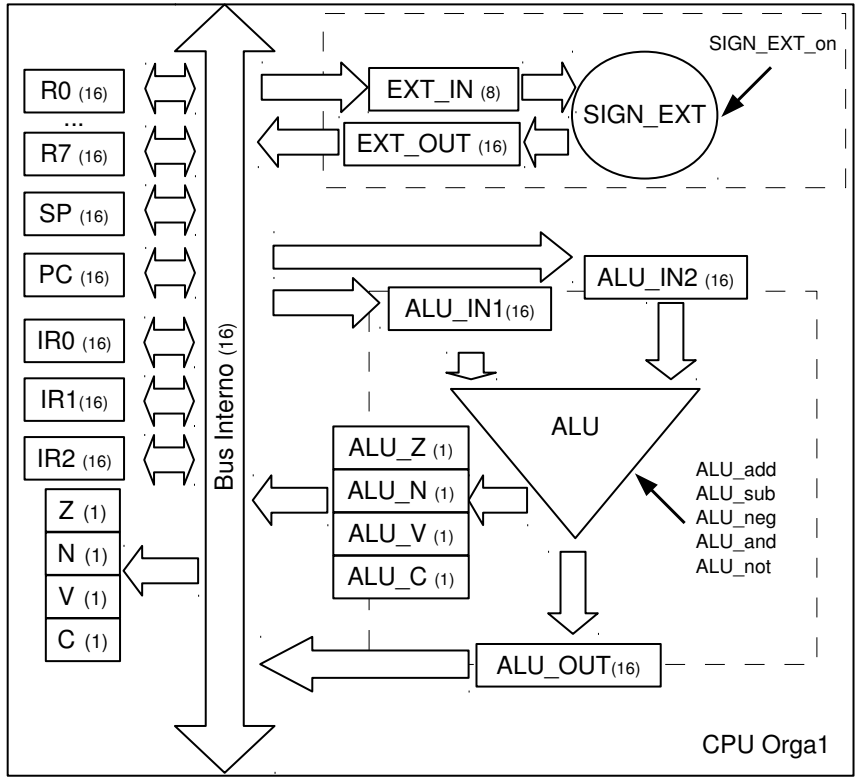
- Un extensor de signo complemento a 2 (SIGN_EXT) con un registro de entrada de 8 *bits* (EXT_IN) y un registro de salida de 16 *bits* (EXT_OUT).
  Sus señales de control son:

Señal	Efecto
$SIGN\_EXT_{on}$	activa la operación de extensión de signo de 8 *bits* a 16 *bits*

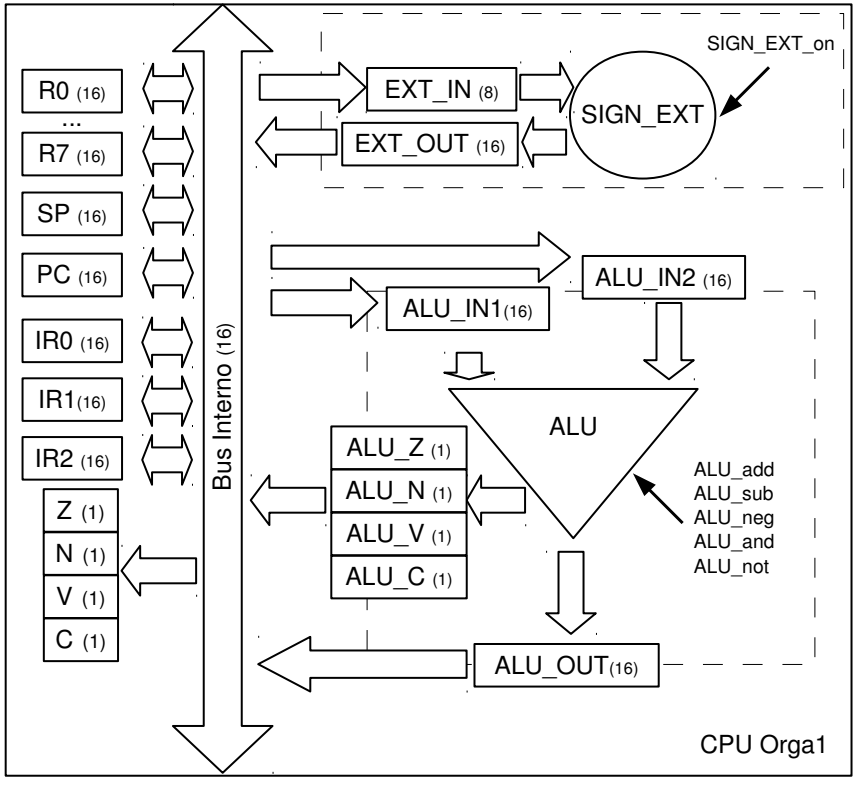
1) Suponiendo que se encuentra resuelta la decodificación y el acceso a memoria de la máquina, diseñar el camino de datos de la arquitectura de la máquina ORGA1. No dibujar la unidad de control para simplificar el diagrama.[2]

[2] Vamos a usar la solución de este ejercicio

Ejercicio 1 - Máquina Orga1 - Solución



Ejercicio 1 - Máquina Orga1 - Solución



2) Indicar cuál es la secuencia de señales (o microoperaciones) que debe realizar la unidad de control para ejecutar las siguientes instrucciones:

   - a) MOV R5, R1 b) AND R7, R1 c) JE 0xFF

Ejercicio 1 - Solución: Secuencias de microoperaciones

- **MOV R5,R1**
  1. R5 := R1

Ejercicio 1 - Solución: Secuencias de microoperaciones

- **MOV R5,R1**
  1. R5 := R1
- **AND R7, R1**
  1. ALU_IN1 := R7
  2. ALU_IN2 := R1
  3. ALU_{and}
  4. R7 := ALU_OUT
  5. Z := ALU_Z
  6. N := ALU_N
  7. C := ALU_C
  8. V := ALU_V

Ejercicio 1 - Solución: Secuencias de microoperaciones

- **MOV R5,R1**
  1. R5 := R1
- **AND R7, R1**
  1. ALU_IN1 := R7
  2. ALU_IN2 := R1
  3. ALU_{and}
  4. R7 := ALU_OUT
  5. Z := ALU_Z
  6. N := ALU_N
  7. C := ALU_C
  8. V := ALU_V
- **JE 0xFF**
  1. IF Z = 1
  2. EXT_IN := IR0 [7:0]
  3. SIGN_EXT_on
  4. ALU_IN_1 := PC
  5. ALU_IN_2 := EXT_OUT
  6. ALU_add
  7. PC := ALU_OUT

Ejercicio 2 - Máquina Orga1 - "La memoria contraataca"

Se cuenta con una memoria con palabra, direccionamiento y direcciones de 16 *bits*. Para operar con ella, se cuenta con un controlador de memoria, con las siguientes características. Posee 2 registros de entrada (ADDR, WRT_DATA) y 1 de salida (RD_DATA). Sus señales de control son:

- MEM_WRITE: Activa la microoperación de escritura del contenido del registro WRT_DATA en la dirección de memoria indicada por el ADDR
- MEM_READ: Activa la microoperación de lectura del contenido de la dirección de memoria indicada por el ADDR, colocando el valor en el registro RD_DATA.

1. Extender el camino de datos de la arquitectura de la máquina ORGA1. No dibujar la unidad de control para simplificar el diagrama[3].

[3] Vamos a usar la solución de este ejercicio

Ejercicio 2 - Máquina Orga1 - "La memoria contraataca"

Se cuenta con una memoria con palabra, direccionamiento y direcciones de 16 *bits*. Para operar con ella, se cuenta con un controlador de memoria, con las siguientes características. Posee 2 registros de entrada (ADDR, WRT_DATA) y 1 de salida (RD_DATA). Sus señales de control son:

- MEM_WRITE: Activa la microoperación de escritura del contenido del registro WRT_DATA en la dirección de memoria indicada por el ADDR
- MEM_READ: Activa la microoperación de lectura del contenido de la dirección de memoria indicada por el ADDR, colocando el valor en el registro RD_DATA.

1. Extender el camino de datos de la arquitectura de la máquina ORGA1. No dibujar la unidad de control para simplificar el diagrama[3].
2. ¿Qué componentes del camino de datos se encuentran dentro del CPU y fuera de él?

[3] Vamos a usar la solución de este ejercicio

Ejercicio 2 - Máquina Orga1 - "La memoria contraataca"

Se cuenta con una memoria con palabra, direccionamiento y direcciones de 16 *bits*. Para operar con ella, se cuenta con un controlador de memoria, con las siguientes características. Posee 2 registros de entrada (ADDR, WRT_DATA) y 1 de salida (RD_DATA). Sus señales de control son:

- MEM_WRITE: Activa la microoperación de escritura del contenido del registro WRT_DATA en la dirección de memoria indicada por el ADDR
- MEM_READ: Activa la microoperación de lectura del contenido de la dirección de memoria indicada por el ADDR, colocando el valor en el registro RD_DATA.

1. Extender el camino de datos de la arquitectura de la máquina ORGA1. No dibujar la unidad de control para simplificar el diagrama[3].
2. ¿Qué componentes del camino de datos se encuentran dentro del CPU y fuera de él?
3. Indicar cuál es la secuencia de señales (o microoperaciones) que debe realizar la unidad de control para ejecutar las siguientes instrucciones:
   - MOV R2, R5
   - MOV R2, [R5]
   - MOV R2, [0xFF00]
   - MOV [0xFF00], [0xFF01]

[3] Vamos a usar la solución de este ejercicio

Ejercicio 2 - Máquina Orga1 - "La memoria contraataca"

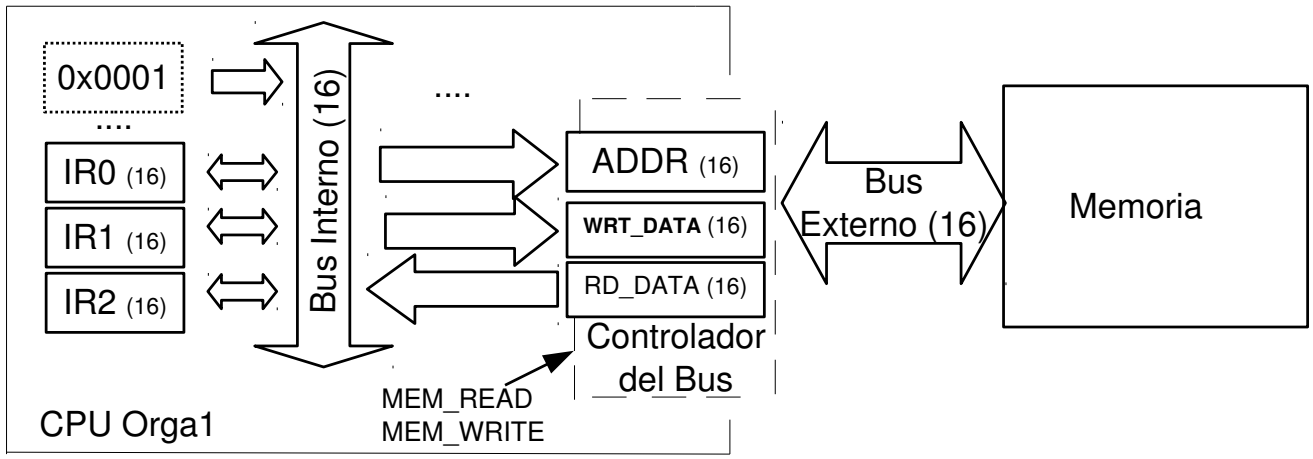
Se cuenta con una memoria con palabra, direccionamiento y direcciones de 16 *bits*. Para operar con ella, se cuenta con un controlador de memoria, con las siguientes características. Posee 2 registros de entrada (ADDR, WRT_DATA) y 1 de salida (RD_DATA). Sus señales de control son:

- MEM_WRITE: Activa la microoperación de escritura del contenido del registro WRT_DATA en la dirección de memoria indicada por el ADDR
- MEM_READ: Activa la microoperación de lectura del contenido de la dirección de memoria indicada por el ADDR, colocando el valor en el registro RD_DATA.

1. Extender el camino de datos de la arquitectura de la máquina ORGA1. No dibujar la unidad de control para simplificar el diagrama[3].
2. ¿Qué componentes del camino de datos se encuentran dentro del CPU y fuera de él?
3. Indicar cuál es la secuencia de señales (o microoperaciones) que debe realizar la unidad de control para ejecutar las siguientes instrucciones:
   - MOV R2, R5
   - MOV R2, [R5]
   - MOV R2, [0xFF00]
   - MOV [0xFF00], [0xFF01]
4. Como Tarea: Describa la secuencia de microoperaciones que realiza la unidad de control para realizar un *fetch* de una instrucción de la máquina Orga1.

[3] Vamos a usar la solución de este ejercicio

Ejercicio 2 - Máquina Orga1 - “La memoria contraataca”: Solución



Ejercicio 2 - Solución: secuencias de microoperaciones

- **MOV R2, R5**
  1. R2 := R5
- **MOV R2, [R5]**
  1. ADDR := R5
  2. MEM_READ
  3. R2 := RD_DATA
- **MOV R2, [0xFF00]**
  1. ADDR := IR1
  2. MEM_READ
  3. R2 := RD_DATA
- **MOV [0xFF00], [0xFF01]**
  1. ADDR := IR2
  2. MEM_READ
  3. WRT_DATA := RD_DATA
  4. ADDR := IR1
  5. MEM_WRITE

Ejercicio 3 - "El retorno de la Pila"

La computadora STACK1 es una máquina de pila con direccionamiento a byte, tamaño de palabra de 16 bits y direcciones de memoria de 12 bits. Trabaja con aritmética complemento a 2 de 16 bits. Posee el siguiente set de instrucciones:

Instrucción	CodOp	Significado
`PUSH [M]`	0000	push [M]
`POP [M]`	0001	[M] := pop
`ADD`	0010	push(pop+pop)
`SUB`	0011	push(pop-pop)
`JUMP`	0100	PC := pop (sólo los 12 bits menos significativos)
`SKIP_N`	0101	ignora la próxima instrucción si top es < 0
`SKIP_Z`	0110	ignora la próxima instrucción si top es 0
`SKIP_GE`	0111	ignora la próxima instrucción si top es $>= 0$

El formato de instrucción de STACK1 es el que sigue:

CodOp	Dirección
4 bits	12 bits

1) Definir el camino de datos y la organización del CPU de STACK1 para soportar la implementación de, al menos, estas instrucciones. Puede utilizar una única ALU con las operaciones ADD y SUB, sin *flags*.

Ejercicio 3 - "El retorno de la Pila"

La computadora STACK1 es una máquina de pila con direccionamiento a byte, tamaño de palabra de 16 bits y direcciones de memoria de 12 bits. Trabaja con aritmética complemento a 2 de 16 bits. Posee el siguiente set de instrucciones:

Instrucción	CodOp	Significado
`PUSH [M]`	0000	push [M]
`POP [M]`	0001	[M] := pop
`ADD`	0010	push(pop+pop)
`SUB`	0011	push(pop-pop)
`JUMP`	0100	PC := pop (sólo los 12 bits menos significativos)
`SKIP_N`	0101	ignora la próxima instrucción si top es < 0
`SKIP_Z`	0110	ignora la próxima instrucción si top es 0
`SKIP_GE`	0111	ignora la próxima instrucción si top es $>= 0$

El formato de instrucción de STACK1 es el que sigue:

CodOp	Dirección
4 bits	12 bits

1) Definir el camino de datos y la organización del CPU de STACK1 para soportar la implementación de, al menos, estas instrucciones. Puede utilizar una única ALU con las operaciones ADD y SUB, sin *flags*.
2) Describa la secuencia de microoperaciones que realiza la unidad de control para realizar un *fetch* de una instrucción.

Ejercicio 3 - "El retorno de la Pila"

La computadora STACK1 es una máquina de pila con direccionamiento a byte, tamaño de palabra de 16 bits y direcciones de memoria de 12 bits. Trabaja con aritmética complemento a 2 de 16 bits. Posee el siguiente set de instrucciones:

Instrucción	CodOp	Significado
`PUSH [M]`	0000	push [M]
`POP [M]`	0001	[M] := pop
`ADD`	0010	push(pop+pop)
`SUB`	0011	push(pop-pop)
`JUMP`	0100	PC := pop (sólo los 12 bits menos significativos)
`SKIP_N`	0101	ignora la próxima instrucción si top es < 0
`SKIP_Z`	0110	ignora la próxima instrucción si top es 0
`SKIP_GE`	0111	ignora la próxima instrucción si top es $>= 0$

El formato de instrucción de STACK1 es el que sigue:

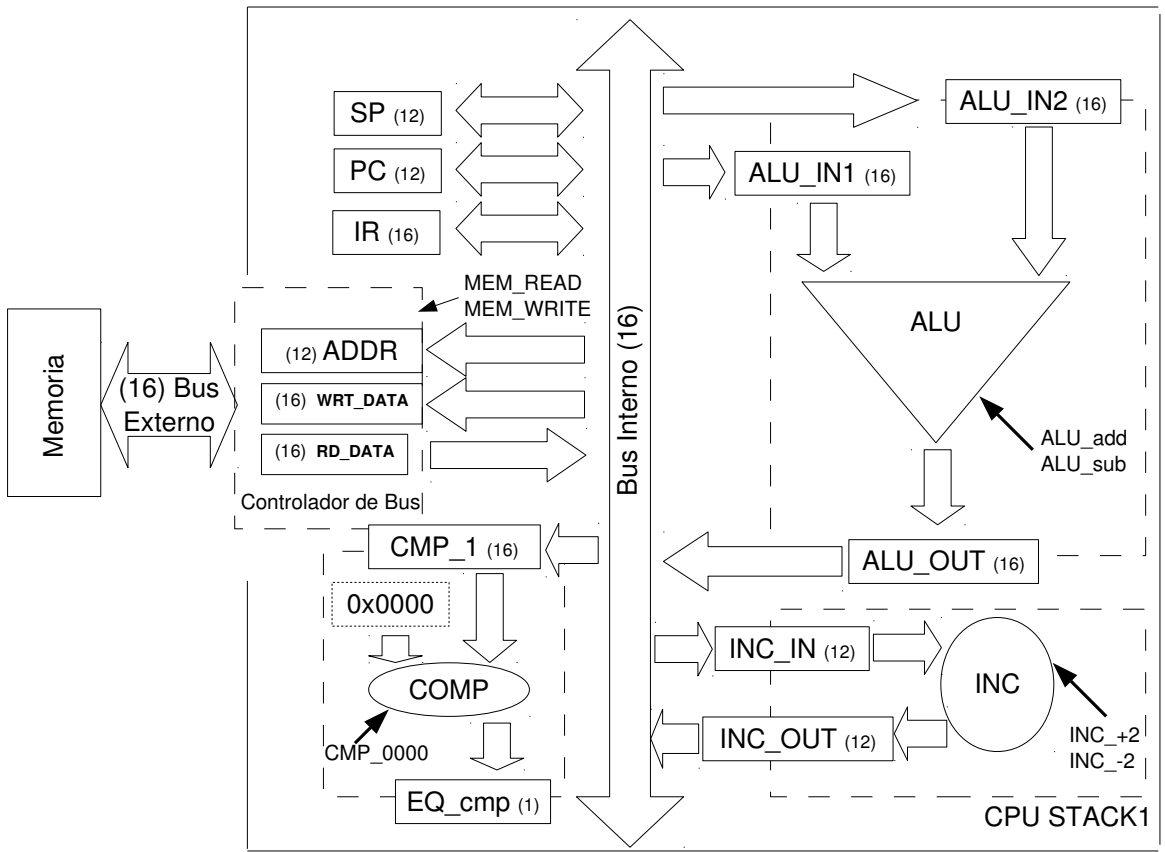
CodOp	Dirección
4 bits	12 bits

1) Definir el camino de datos y la organización del CPU de STACK1 para soportar la implementación de, al menos, estas instrucciones. Puede utilizar una única ALU con las operaciones ADD y SUB, sin *flags*.
2) Describa la secuencia de microoperaciones que realiza la unidad de control para realizar un *fetch* de una instrucción.
3) Implementar las siguientes instrucciones:

I. `JUMP`
II. `SKIP_Z`
III. `PUSH [0xFAC]`
IV. `ADD`

Ejercicio 3: Solución[4]

Se utiliza un circuito incrementador de 12 *bits* con 2 señales: `INC_+2` que suma 2 a la entrada, y `INC__2` que resta 2 a la entrada.
`ADDR`, `INC_IN`, `INC_OUT`, `SP` y `PC` son registros de 12 bits.
`IR`, `CMP_1`, `ALU_IN1`, `ALU_IN2`, `ALU_OUT`, `WRT_DATA` y `RD_DATA` y los buses INTERNO y EXTERNO son de 16 bits.
`EQ_CMP` es de 1 bit. Las 12 líneas de los registros correspondientes están conectadas a las líneas menos significativas del BUS.



[4] Usamos el resultado del este ejercicio (1)) en los siguientes ejercicios.

Ejercicio 3 - Solución: *fetch*

1. ADDR := PC
2. MEM_READ
3. IR := RD_DATA // cargo el IR
4. INC_IN := PC
5. INC_{-+2}
6. PC := INC_OUT // incremento PC

Ejercicio 3 - Solución: secuencia de microoperaciones

- JUMP
  1. INC_IN := SP
  2. INC_+2
  3. SP := INC_OUT
  4. ADDR := SP
  5. MEM_READ
  6. PC := RD_DATA[11:0]

- SKIP_Z
  1. INC_IN := SP
  2. INC_+2
  3. ADDR := INC_OUT
  4. MEM_READ
  5. CMP_1 := RD_DATA
  6. CMP_0000
  7. if EQ_cmp = 0x0
  8. INC_IN := PC
  9. INC_+2
  10. PC := INC_OUT
  11. endif

Ejercicio 3 - Solución: secuencia de microoperaciones

- PUSH [X]
  1. ADDR := IR[11:0]
  2. MEM_READ
  3. WRT_DATA := RD_DATA
  4. ADDR := SP
  5. MEM_WRITE
  6. INC_IN := SP
  7. INC_{-2}
  8. SP := INC_OUT

- ADD
  1. INC_IN := SP
  2. INC_{+2}
  3. SP := INC_OUT
  4. ADDR := SP
  5. MEM_READ
  6. ALU_IN1 := RD_DATA // primer operando
  7. INC_IN := SP
  8. INC_{+2}
  9. SP := INC_OUT
  10. ADDR := SP
  11. MEM_READ
  12. ALU_IN2 := RD_DATA // segundo operando
  13. ALU_{add}
  14. WRT_DATA := ALU_OUT
  15. ADDR := SP
  16. MEM_WRITE // push resultado
  17. INC_IN := SP
  18. INC_{-2}
  19. SP := INC_OUT

Con lo visto hoy pueden hacer toda la práctica 7.