Assembler

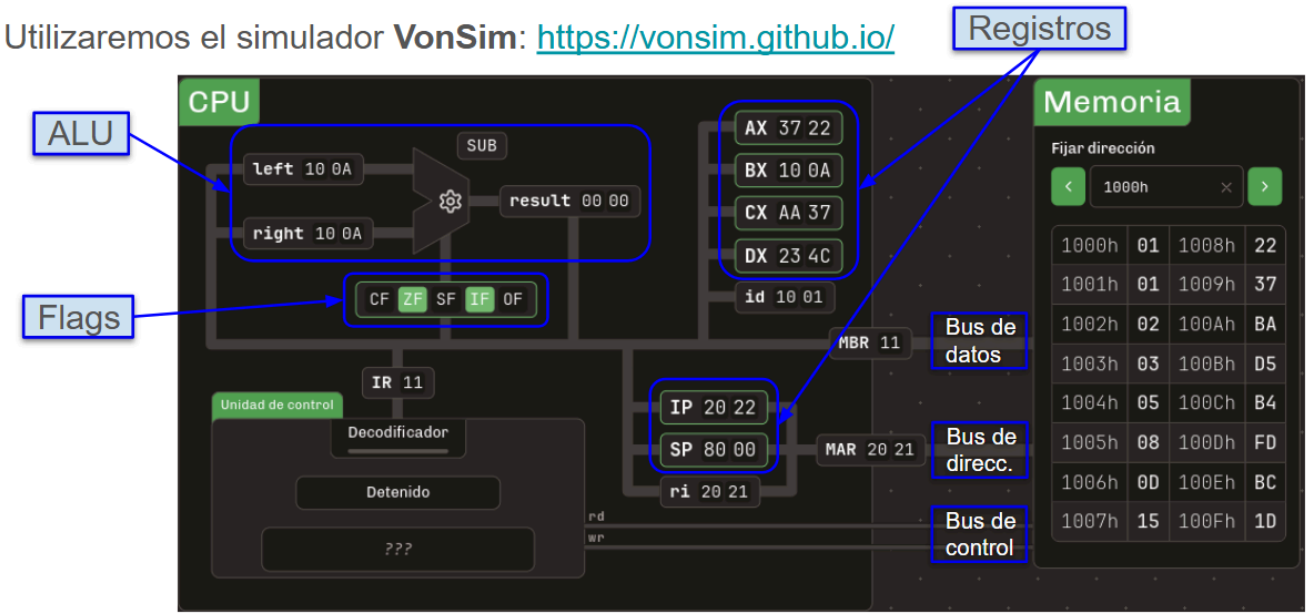
#tercerParte

Implementa una representación simbólica de los códigos de máquina binarios y otras constantes necesarias para programar una arquitectura de procesador y constituye la representación más directa del código máquina específico para cada arquitectura legible por un programador. Cada arquitectura deberá brindar su propio lenguaje ensamblador en función de las correspondientes especificaciones de dicha arquitectura (conjunto de instrucciones, formato de instrucción, modos de direccionamiento, etc.).

Tipos de arquitecturas:
CISC (Complex Instruction Set Computing)
x86, IMB/360, Motorola 68k
RISC (Reduced Instruction Set Computing)
ARM, MIPS, PowerPC

En la materia veremos características básicas del assembler correspondiente al *Intel 8088*
Características:

- Bus de datos de 8 bits
- Bus de direcciones de 16 bits
- Registros ppales: AX, BX, CX, DX, IP, SP; todos de 16 bits



Memoria principal que cubre el espacio de direcciones 0000h hasta FFFFh.

- La mitad más baja (0000h hasta 7FFFh) reservada para el usuario:
  - Aquí se almacenan los programas y datos.
- La mitad más alta (8000h hasta FFFFh) reservada para el "sistema operativo":

    - Programa monitor simple que permite al usuario interactuar con varios dispositivos.
- Instrucciones (mnemónicos):
    - Movimiento de datos: MOV, IN, OUT, etc.
    - Logico-matemáticas: AND, OR, XOR, NOT, ADD, SUB, etc.
    - Saltos condicionales/incondicional: JZ, JNZ, JC, JNC, JMP, etc.
    - Pila: PUSH, POP
- Etiquetas:
    - Referenciar una dirección de memoria en donde se aloja una instrucción o un dato
    - Entre otras cosas se utilizan para
        - "alojar/definir variables"
        - "referenciar/saltar" a cierta parte del código mediante saltos o subrutinas
- Directivas:
    - Son exclusivas para el compilador del programa, no son instrucciones
    - Ejemplos: ORG, EQU, END, etc.

Información sobre VonSim:

```
https://github.com/vonsim/vonsim
```

Etiquetas:
Una etiqueta permite identificar una dirección de memoria, la cual puede almacenar un dato (una variable) o una instrucción de programa.
Se utiliza la palabra db para alocar 1 byte de memoria y dw para alocar 2 bytes.
Ejemplos de etiquetas para definir datos:
posX db 255 ; valor de la variable expresado en decimal
posY db 0FFh ; valor de la variable expresado en hexadecimal
posZ db 11111111b ; valor de la variable expresado en binario
temperatura db ? ; espacio de mem. reservado pero no inicializado
distancia dw 23ABh ; ocupará 2 bytes en memoria, inicializado con un valor
periodo dw 11110000b ; ocupará 2 bytes en memoria, inicializado con un valor

Dirección	Valor	Etiqueta
1000H	255	$posX$
1001H	$0FFh$	$posY$
1002H	$11111111b$	$posZ$
1003H	?	$temperatura$
1004H	ABh	$distancia$
1005H	$23h$	
1006H	$00h$	$periodo$
1007H	FFh	

Notar:

- Si bien en la declaración de variables y en la tabla de la memoria los valores se representan en decimal, hexadecimal o binario, en la realidad los valores son siempre almacenados como secuencias de bits.
- Para los datos de 16 bits, la parte alta del valor se almacena en la parte alta de la memoria (little-endian).
- DW = Define word.
- DB = Define byte.

También es posible definir una etiqueta que referencia el incio de una cadena de caracteres:
texto db "Hola mundo"
excl db 21h, 33

Dirección	Valor	Etiqueta
1000H	'H'	*texto*
1001H	'o'	
1002H	'l'	
1003H	'a'	
1004H	' '	
1005H	'm'	
1006H	'u'	
1007H	'n'	
1008H	'd'	
1009H	'o'	
100AH	'!'	*excl*
100BH	'!'	

- Cada caracter requerirá un byte de memoria, correspondiente a la codificación ASCII
- El espacio también es un caracter ASCII y consume 1 byte
- El código ASCII 21h (33 decimal) corresponde al signo de exclamación !
- En la tabla de memoria los valores se muestran como caracteres ASCII, pero en realidad los valores son siempre almacenados como secuencias de bits.
- Es posible definir valores contiguos a partir de una etiqueta, como los valores 21h y 33 en la etiqueta excl

---

Ejemplos de etiquetas para definir posiciones específicas del código:
main: MOV BX, DX
DEC BX
JZ other
MOV AX, 3
loop: DEC AX
JNZ loop
other: INC BL
JMP main

Dirección	Valor	Etiqueta
2000H	MOV BX,DX	main
2001H	MOV BX,DX	
2002H	DEC BX	
2003H	JZ other	
2004H	JZ other	
2005H	MOV AX,3	
2006H	MOV AX,3	
2007H	MOV AX,3	
2008H	DEC AX	loop
2009H	JNZ loop	
200AH	JNZ loop	
200BH	INC BL	other
200CH	JMP main	
200DH	JMP main	
200EH	JMP main	

Notar:

- Dependiendo la instrucción se requerirá una cantidad diferente de bytes para cada una de éstas.
- Las etiquetas permiten identificar puntos en el código a fin de poder realizar "saltos", tal como veremos más adelante.

Instrucciones

ADD

Realiza la suma de dos operandos; Siempre es DESTINO = DESTINO + ORIGEN.

$\text{ADD DX}, CX \leftarrow DX = DX + CX$

SUB

Realiza la resta de dos operandos; Siempre es DESTINO = DESTINO - ORIGEN.

CMP: Similar a SUB pero no escribe en DESTINO el resultado, solo modifica los flags.

$\text{SUB DX}, CX \leftarrow DX = DX - CX$

AND

Realiza la operación lógica AND bit a bit; Siempre es DESTINO = DESTINO AND ORIGEN.

$\text{AND DX}, CX \leftarrow DX = DXANDCX$

JZ, JNZ, JC, JNC, JS, JNS, JO, JNO

Saltos condicionales según el estado activo o no de los flags cero, carry, signo, overflow.

JMP: Salto incondicional a otro punto del código.

Si el salto es incondicional o bien la condición se cumple (en los casos condicionales), se copiará la dirección de la etiqueta referenciada al registro IP.

Estructuras de control:

If:

CMP AX, BX ; CMP resta AX-BX y setea los flags sin modificar AX

```
JZ then ; Si AX y BX poseen mismo valor, su resta encenderá flag Z
JNZ else ; Si AX y BX poseen distinto valor, flag Z no se encenderá

then: INC AX ; Accedo por el then
JMP cont ; Debo evitar ejecutar la lógica del else en este caso

else: DEC BX ; Accedo por el else
cont: ; resto del codigo
```

While:

```
MOV AX, 10 ; valor inicial de AX para el loop
loop: ADD BX, 5 ; lógica especifica del bucle
DEC AX ; una iteración menos
JNZ loop ; si el flag de Z no se enciende, volver a loop
```

Directivas

ORG

Especifica la dirección de memoria en donde se alojará el código escrito a continuación (ya sea alocación de memoria para datos o código).
ORG 1000H; … ← si declaro variables, se almacenarán a partir de la celda 1000H.

EQU

Directiva para equivalencias, similar a definir constantes.
EVEREST_HEIGHT_METERS EQU 8849

```
; Constantes
MAX_VIDAS    equ 3              ; numero maximo de vidas
MAX_ENERGIA  equ 0AH            ; energia maxima por vida
PISO_POS_Y   equ 0              ; posicion Y del piso

; Variables
org 1000h
vidas     db MAX_VIDAS          ; numero de vidas del jugador
energia   db MAX_ENERGIA        ; nivel de energia del jugador
posicionX db ?                  ; posicion X del jugador
posicionY db ?                  ; posicion Y del jugador

; Programa
org 2000h

loop:   ; logica del juego. Aqui deberia ir codigo "util"
MOV posicionY, 0          ; simulamos que el jugador cae
CMP posicionY, PISO_POS_Y  ; evaluamos si hemos
JNZ loop                  ; golpeado el piso

golpe:  DEC energia               ; golpe por caida, decrementar
JZ muerte                 ; la energia y restar una
JMP loop                  ; vida si corresponde
```

```
muerte: DEC vidas               ; una vida menos, si quedan
JZ FIN                  ; vidas reiniciar la energia
MOV energia, MAX_ENERGIA   ; y continuar, en caso
JMP loop                ; contrario finalizar

fin:    hlt                 ; fin del juego
end
```

OFFSET

Es un operador que utiliza el compilador para obtener la dirección de memoria de una etiqueta o variable en un programa en ensamblador.

$\mathrm{MOV\ AX, OFFSET\ unaVar}$; en AX se almacena la dirección de unaVar.

PUSH y *POP*: Pila (Stack)

Pila: Espacio reservado de la memoria para alojar información de manera temporal. Política LIFO (Last In First Out).

El registro SP (Stack Pointer) apunta al tope de la pila. Inicialmente SP = 8000H

PUSH y POP trabajan con AX, BX, CX, DX (siempre 16 bits).

PUSH envía un dato a la pila y POP lo recupera de ésta.

PUSH AX: SP se decrementa en 2 unidades y vuelca el dato de AX a la pila en la dirección apuntada por SP y la siguiente.

POP AX: Se copia la información de pila en la dirección apuntada por SP (y la siguiente) hacia AX y se incremente SP en 2 unidades.

CALL y *RET*

Son similares a los llamados a funciones o procedimientos en lenguajes de alto nivel, aunque la definición de parámetros y valor de retorno no es parte de la definición a nivel código.
