



INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE COMPUTO

ROM 32

Practica 11B

Materia:

Arquitectura de computadoras

Profesor:

Castillo Cabrera Gelacio

Alumno:

Cortés Piña Oziel

Grupo:

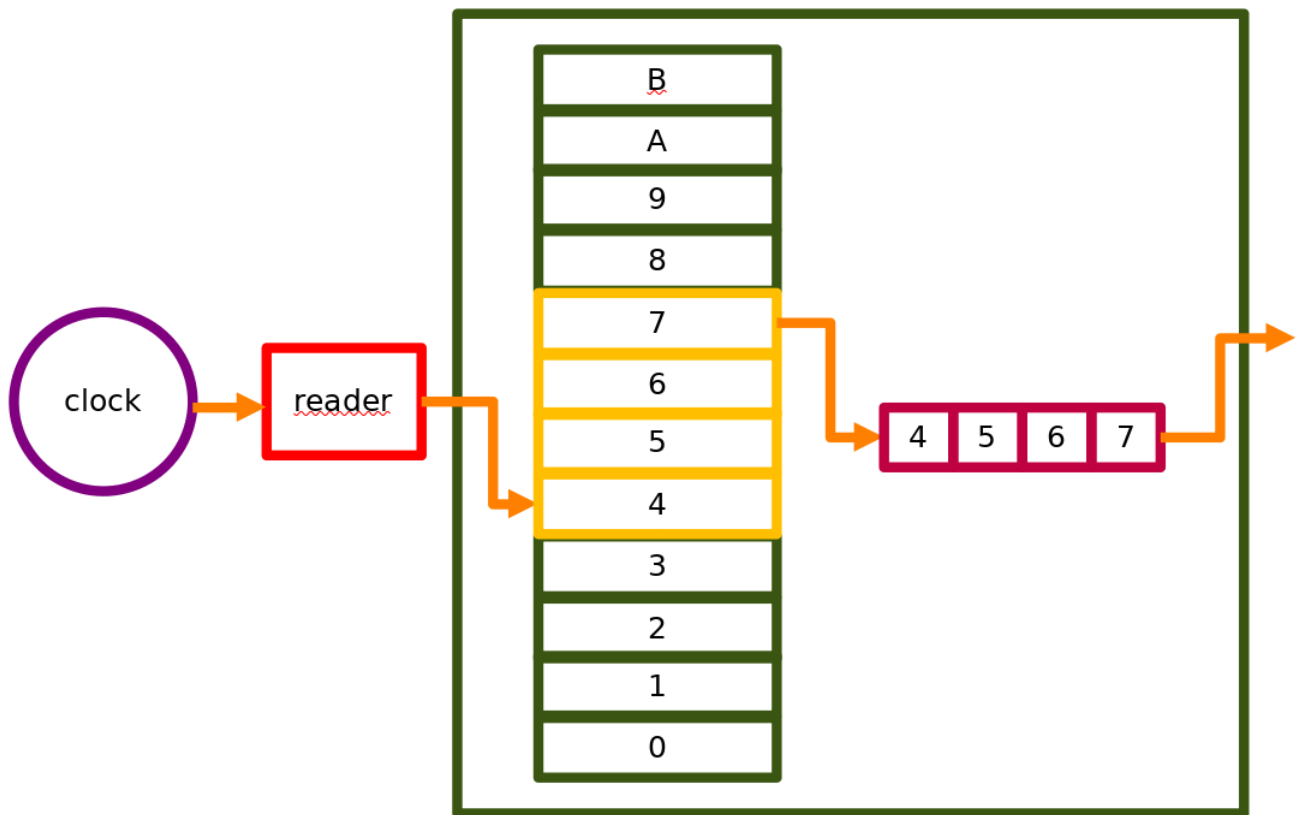
3CM12



Introducción

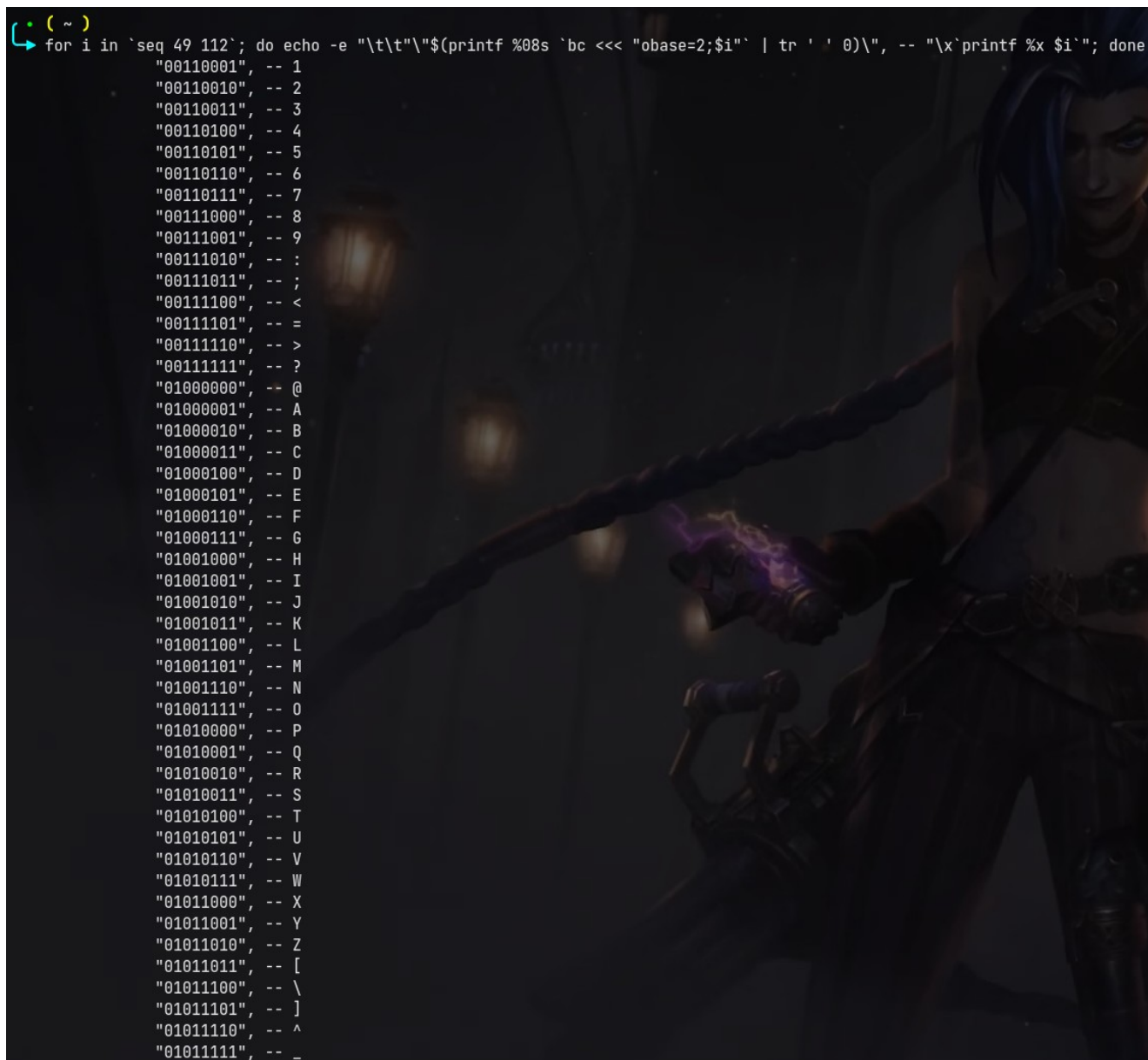
La memoria rom es muy fácil al solo ser un elemento de lectura. El elemento rom en un arreglo de bytes (siete bits) al que se puede acceder mediante un número que recibe por un puerto o bus de cinco bits. Para hacer uso de la rom se genera un componente llamado reader (en mi caso) que genera un número que aumenta con cada ciclo de reloj. Así pues puedo barrer la memoria y hacer lectura de los registros de la memoria. En este caso la rom no contiene mucho registros por lo que si solicita un elemento que no encuentra en la memoria arrojará un valor por defecto.

Esta memoria tiene la modificación de ser de 32 bites, es decir 4bytes por lo que el bus de datos es más grande y el direccionamiento ahora es por un bus de seis bits que da la posibilidad de direccionar 64 unidades de memoria. Para poder entregar esta información la rom toma las siguientes 3 unidades después de la dirección que el reader le indica para concatenar una word.



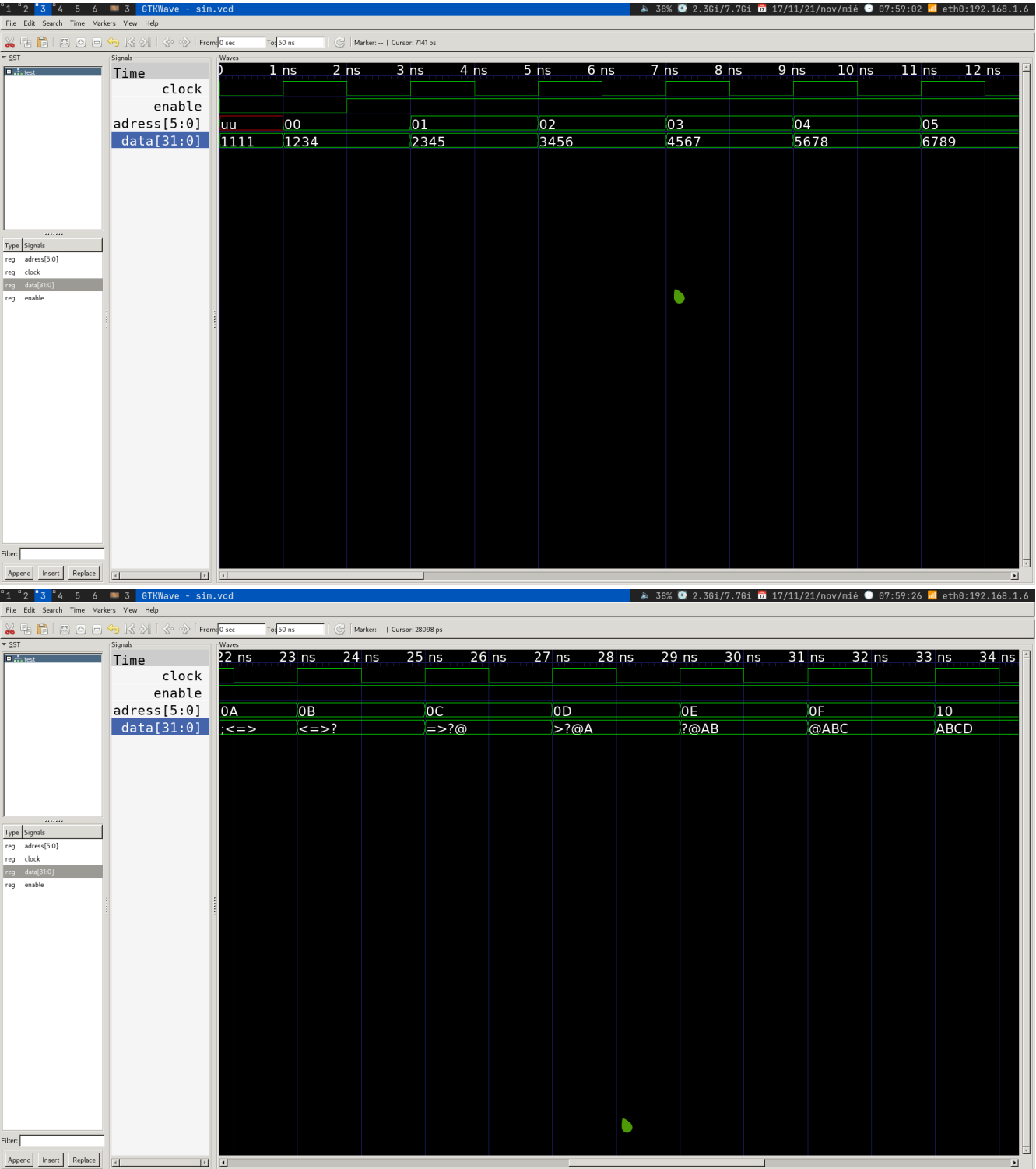
Como extra dado que estoy en simulación uso ascii para visualizar información. Use el siguiente comando para generar esa parte del código escrito en bash.

```
for i in `seq 49 112`;
do
    echo -e "\t\t\t\"$(printf %08s `bc <<< obase=2`;$i` | tr ' ' 0)`\", -- \"\x`printf %x $i`;
done >> mem.vhdl
```



```
( ~ )
for i in `seq 49 112`; do echo -e "\t\t\t\"$(printf %08s `bc <<< obase=2`;$i` | tr ' ' 0)`\", -- \"\x`printf %x $i`; done
"00110001", -- 1
"00110010", -- 2
"00110011", -- 3
"00110100", -- 4
"00110101", -- 5
"00110110", -- 6
"00110111", -- 7
"00111000", -- 8
"00111001", -- 9
"00111010", -- :
"00111011", -- ;
"00111100", -- <
"00111101", -- =
"00111110", -- >
"00111111", -- ?
"01000000", -- @
"01000001", -- A
"01000010", -- B
"01000011", -- C
"01000100", -- D
"01000101", -- E
"01000110", -- F
"01000111", -- G
"01001000", -- H
"01001001", -- I
"01001010", -- J
"01001011", -- K
"01001100", -- L
"01001101", -- M
"01001110", -- N
"01001111", -- O
"01010000", -- P
"01010001", -- Q
"01010010", -- R
"01010011", -- S
"01010100", -- T
"01010101", -- U
"01010110", -- V
"01010111", -- W
"01011000", -- X
"01011001", -- Y
"01011010", -- Z
"01011011", -- [
"01011100", -- \
"01011101", -- ]
"01011110", -- ^
"01011111", -- _
```

Simulación en GHDL y GTKWAVE



Análisis de vectores

Dirección	Información de salida
01	"1234"
02	"2345"
03	"4567"
04	"5678"
05	"6789"
06	"789."
07	"89:."
09	"9:;<"
0A	".:;<="
0B	".:<=>"
0C	"<=>?"
0D	"=>?@"
0E	">?@A"
0F	"?@AB"
10	"@ABC"
11	"ABCD"
12	"BCDE"
13	"CDEF"
14	"DEFG"
15	"EFGH"
16	"FGHI"
17	"GHIJ"
18	"HIJK"
19	"IJKL"

Conclusión

Podemos concluir el funcionamiento básico del direccionamiento básico que es tomar cada posible casilla de la memoria como un número. Esa visión no acerca por mucho a la idea que tenemos de un archivo o de una zona de memoria referencia-ble en un lenguaje de programación como C.