

## Elementos de Sistemas - Projeto F - Ferramental - Programando

Rafael Corsi - rafael.corsi@insper.edu.br

Abril, 2018

### Configurando o ambiente

#### Windows

Modificar a variável de ambiente PATH para incluir os programas do Quartus

Para isso, execute na linha de comando :

```
rundll32 sysdm.cpl>EditEnvironmentVariables
```

Edite a variável **Path**, clique em **PROCURAR** e inclua o diretório :

```
C:\intelFPGA_lite\17.1\quartus\bin32
```

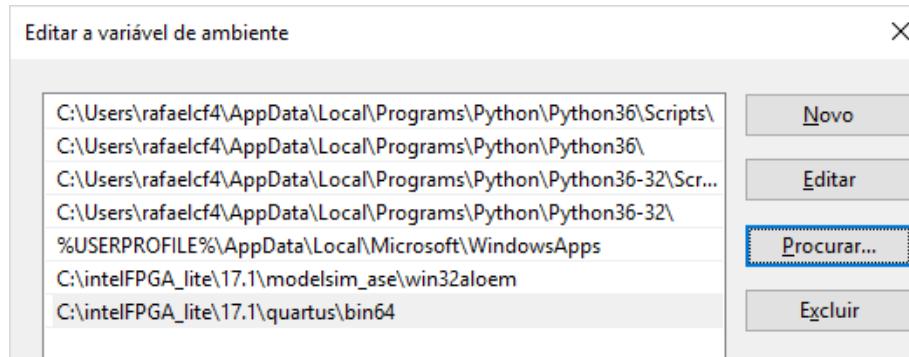


Figure 1: Windows Path

Feche todos os terminais

## Linux

Não é necessário fazer nenhuma modificação se utilizado o Ubuntu, caso contrário adicione o quartus/bin/ ao path do sistema.

## Validando

No terminal execute : quartus\_pgm -v e verifique se o programa é reconhecido pelo bash

## Programando o Z01

Existem dois scripts na pasta do projeto F-Assembly que interfaceia com a FPGA: programSOF.py e programNASM.py. O primeiro deve ser usado uma única vez sempre que a FPGA for energizada e serve para programar o nosso Hardware (Computador) na FPGA já o segundo serve para atualizar a memória ROM atualizando-a com um novo programa, e deve ser executada sempre que desejar testar um novo código.



Executando da pasta Z01/Projetos/F-Assembly/

- programSOF.py : programa o Hardware da FPGA com o Z01

```
python scripts/programSOF.py
```

- programNASM.py : programa a ROM do Z01 com o nasm passado.

```
python scripts/programNASM.py -i nasmFile.nasm
```

## Testando

Para testar vamos gravar o R-LCD.nasm na FPGA, esse código gera a letra R e a grava na FPGA.

```
python scripts/programNASM.py -i src/nasm/R-LCD.nasm
```

## I/O mapeado em memória

Input/Output mapeado <sup>1</sup> em memória é uma técnica muito utilizada para possibilitar que a CPU tenha acesso a periféricos externos ao computador, como por exemplo: teclado, tela, mouse, .... Nessa técnica, mapea-se esses periféricos para

<sup>1</sup>[https://pt.wikipedia.org/wiki/E/S\\_mapeada\\_em\\_mem%C3%B3ria](https://pt.wikipedia.org/wiki/E/S_mapeada_em_mem%C3%B3ria)

um endereço de memória que na verdade não é necessariamente uma memória, mas sim um periférico.

Por exemplo, no caso nosso computador Z01, o endereço 21184 da memória RAM não é a memória, mas sim os LEDs da FPGA, escrevendo nesse endereço faz com que a informação seja transmitida para os LEDs e não é salva no endereço 21184 da memória RAM (RAM[21184]).

Essa técnica é muito útil utilizada pois simplifica a CPU, não necessitando de instruções especiais para acessar o mundo externo.

## Periféricos do Z01

O nosso computador possui quatro periféricos externos que podem ser acessados via a CPU, são eles :

1. RAM : Memória RAM de 16384 endereços ( $2^{14}$  bits)
2. LCD : LCD de 320x240 px (conectado no GPIO0)
3. LED : 10 leds do kit DE0-CV (LEDR9 ... LEDR0)
4. SW : 10 chaves do kit DE0-CV (SW9 ... SW0)

Os periféricos estão mapeados em memória conforme a tabela a baixo :

Endereço	SPAN (words)	Periférico	Read/ Write
0	16384	R/W	RAM
16384	4801	W	LCD
21184	1	W	LED
21185	1	R	SW

O diagrama a seguir ilustra os periféricos e seus respectivos endereços :

## Leitura/Escrita

Nem todos os endereços de memória suportam a operação de leitura e escrita (write/read), em alguns casos a operação é limitada apenas a um desses modos. Por exemplo: o endereço reservado para os LEDs da FPGA é de apenas escrita (Write-only), já o endereço reservado para as chaves é apenas de leitura (Read-Only).

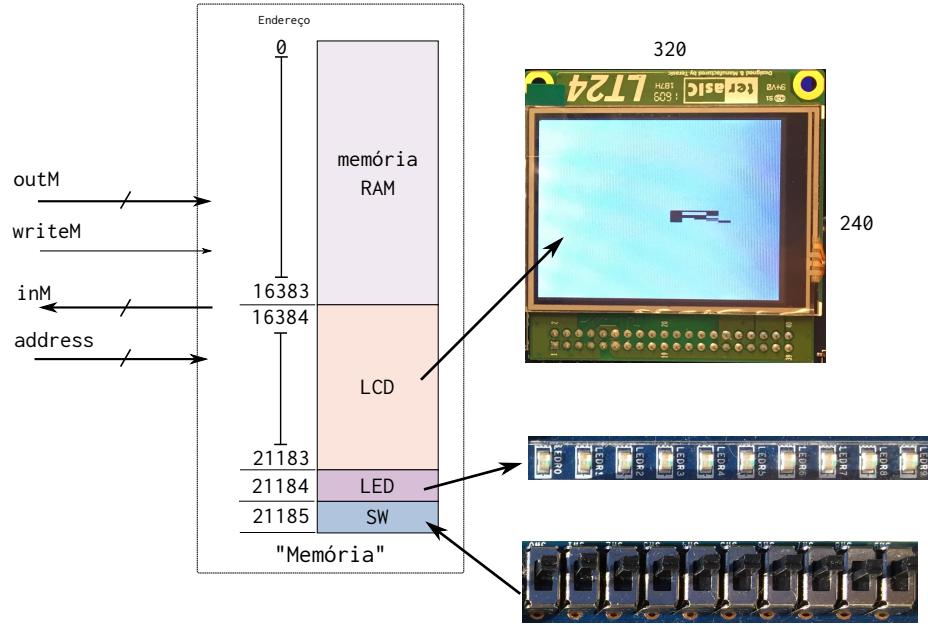


Figure 2: Mapa de Memória

## Memória RAM

**Endereço : [0 - 16383]** Read-Write

A memória RAM é um periférico da CPU já que está localizado externo a ela, a nossa memória possui 16384 endereços e largura de 16bits. Podemos acessar a memória RAM para armazenar ou restaurar (ler) informações simplesmente apontando para o endereço que desejamos acessar (registrador %A) e movendo o dado para ou de esse endereço.

Alguns endereços da memória RAM possuem nomes que facilitam o desenvolvimento de códigos e que futuramente darão um uso especial para esse locais, é possível utilizar esses nomes na operação leaw, eles serão automaticamente traduzidos para o endereço correspondente. Esses nomes são chamados de *label*, pois somente dão um nome ao endereço.

Label	Endereço
SP	0
LCL	1
ARG	2
THIS	3
THAT	4

Label	Endereço
R0-R15	0-15
SCREEN	16384

No código podemos fazer a seguinte operação para carregarmos o endereço 2 no registrador A :

```
leaw $R2, %A
```

ou, que é equivalente já que o R2 e ARG são labels para o mesmo endereço :

```
leaw $ARG, %A
```



Você pode criar arquivos na pasta F-Assembly/src/nasm/ e adicionar esses códigos lá. Note que a extensão do arquivo é **.nasm**

```
; regS = RAM[256]

leaw $256, %A      ; carrega o valor 256 no reg. A
; A serve como um apontador de memoria
; ele é o "address".
movw (%A), %S      ; Armazena em S o valor que está no
; endereço 256 da memoria RAM
```

---

```
; RAM[128] = 12
```

```
leaw $12, %A      ; Carrega o valor 12 no reg. A, essa
; é a unica forma que temos para carregar
; uma constante na CPU
movw %A, %D        ; Move o valor carregado do reg. S
; para o reg. D
leaw $127, %A      ; Carrega 128 no reg. A
movw %D, (%A)       ; move o valor que esta no reg D
; para onde A ponta
```

---

Execute esses códigos no simulador e verifique se os resultados são concordantes.

## LCD - Display

Endereço : [16384 - 21183] Write-Only

O display utilizado no Z01 é o LT24<sup>2</sup> que possui 320x240 pixels e está conectado no GPIO0 do kit de desenvolvimento (DE0-CV). Cada endereço de memória do LCD escreve simultaneamente em 16 pixels do LCD, se o bit em questão for ‘1’ o LCD irá interpretar como cor preto e irá pintar esse px de preto, se o valor for ‘0’ o LCD irá pintar o px de branco (mesma cor do fundo), conforme diagrama a seguir :

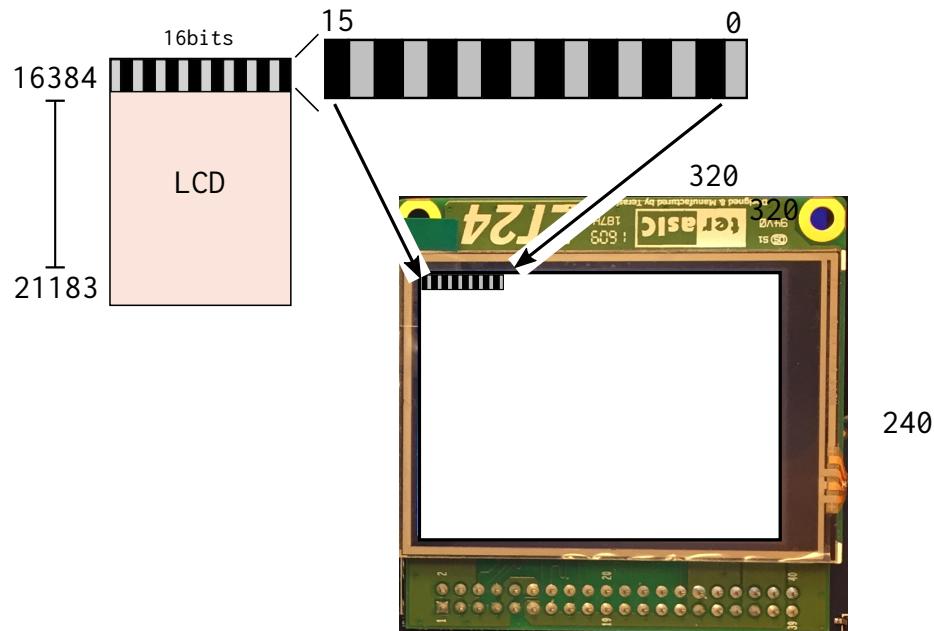


Figure 3: LCD

O exemplo a seguir pinta os 16 primeiros bits do LCD :

---

```
; Liga os 16 primeiros pxs do LCD

movw $0, %S          ; Carrega 0 no reg. S
decw $S               ; faz 0 - 1 = 0x111111111111
                      ; faz com que todos os bits sejam 1
leaw $16384, %A       ; Carrega o endereço dos primeiros bits
movw %S, (%A)         ; Move o valor de %S para o LCD
```

Qual endereço de memória é equivalente aos pixels centrais do LCD ?

<sup>2</sup><https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=65&No=892>

Faça um código que acenda esses pxs.



Execute esses códigos no simulador e verifique se os resultados são coincidentes.

## LEDs

**Endereço : [21184] Write-Only**

Escrevendo no endereço 21184 faz com que os 10 primeiros bits da palavra alterem o estado dos leds (bit a bit). O valor ‘1’ no bit faz com que o LED respectivo acenda e o valor ‘0’ faz com que o LED apague.

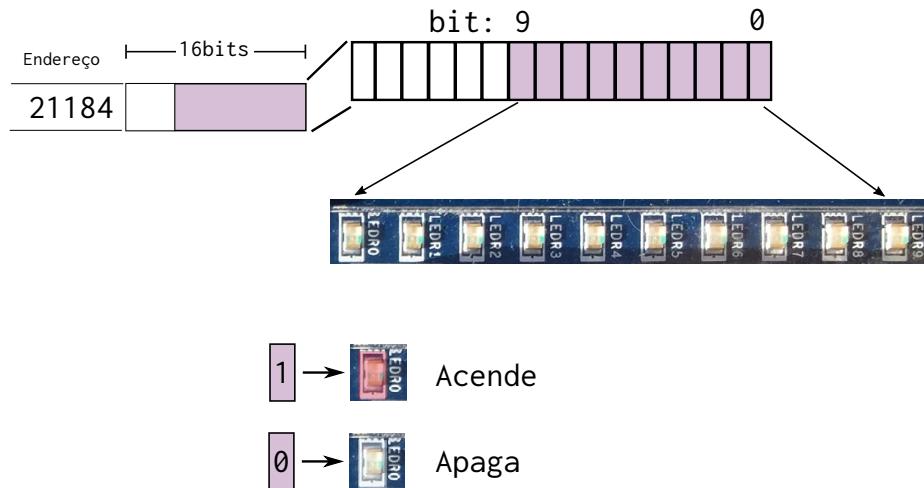


Figure 4: LEDs

O código exemplo a seguir acende a metade inferior dos leds.

```
; Acende metade inferior dos LEDs  
; LED = 0000011111  
  
leaw $31, %A          ; Carrega 31 em A  
                      ; 31 = 0000000000011111  
movw $A, %S            ; S = A  
leaw $21184, %A        ; Carrega o endereço dos LEDs em A  
movw %S, (%A)          ; Salva nos LEDs o valor de A
```

Acenda os LEDs mais significativos - Programe a FPGA para testar.

Acenda os LEDs intercalados - Programe a FPGA para testar.

## Chaves

**Endereço : [21185] Read-Only**

Ler do endereço 21185 implica em acessar o estado das chaves, os bits afetados são do 9 ao 0, como no diagrama a seguir :

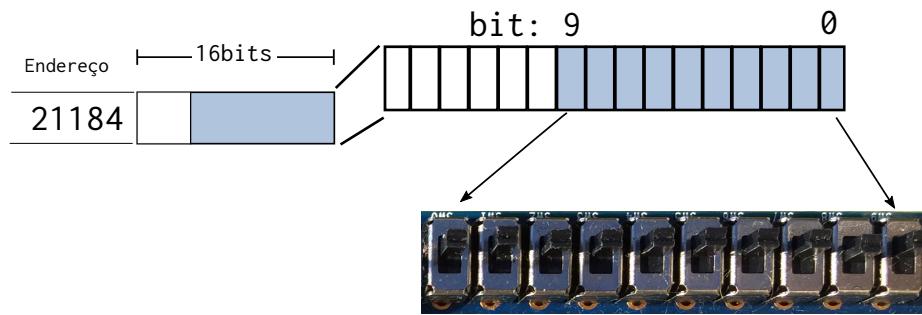


Figure 5: Chaves

O exemplo a seguir copia o valor das chaves para o endereço 0 da memória RAM

*; Le o valor das chaves e salva na RAM[0]*

```
leaw $21185, %A      ; Carrega o endereço das SW em A
movw ($A), %S         ; S = SWs
leaw $0, %A            ; Carrega 0 em A
movw %S, (%A)          ; Move o valor das chaves (S) para RAM[0]
```

Acenda os LEDs conforme o valor das chaves (LED = SW) - Programe a FPGA para testar.

Acenda os LEDs de forma invertida que as chaves (LED = not SW) - Programe a FPGA para testar.