Computação Digital

ENG 1448

Trabalho final

Objetivos:

- Testar os conhecimentos adquiridos na disciplina ao criar um “computador” (CPU) a partir de uma FPGA.

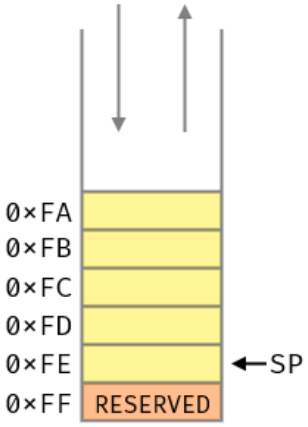
Descrição:

Você deverá fazer uma estrutura mínima (CPU + memória RAM estática), capaz de executar as instruções da tabela abaixo, cujo *OpCode* já está definido na 2ª coluna. O código do programa deverá ser carregado na memória na largada (isto é, a memória já será inicializada com o código do programa de vocês). O CPU deverá possuir 4 registradores (A, B, C e D), além dos registradores de IR, PC, SP, MBR e MAR.

A memória é estática e deverá ter 256 posições de 8 bits cada, o que requer também 8 linhas de endereçamento.

	ASSEMBLY	OpCode	Descrição	
aritmético	add Rx, Ry	0000 Rx Ry	Rx <- Rx + Ry,	pc <- pc + 1
	sub Rx, Ry	0001 Rx Ry	Rx <- Rx – Ry,	pc <- pc + 1
	inc Rx	0010 Rx 00	Rx <- Rx + 1,	pc <- pc + 1
	incc Rx	0010 Rx 01	Rx <- Rx + carry,	pc <- pc + 1
	dec Rx	0010 Rx 10	Rx <- Rx – 1,	pc <- pc + 1
	decc Rx	0010 Rx 11	Rx <- Rx + carry – 1,	pc <- pc + 1
lógico	and Rx, Ry	0011 Rx Ry	Rx <- Rx & Ry,	pc <- pc + 1
	or Rx, Ry	0100 Rx Ry	Rx <- Rx | Ry,	pc <- pc + 1
	not Rx	0101 Rx 00	Rx <- ~Rx,	pc <- pc + 1
	xor Rx, Ry	0110 Rx Ry	Rx <- Rx ^ Ry,	pc <- pc + 1
	rol Rx	0111 Rx 00	Rx <- Rx[6..0] carry,	pc <- pc + 1
	ror Rx	0111 Rx 01	Rx <- carry Rx[7..1],	pc <- pc + 1
	lsl Rx	0111 Rx 10	Rx <- Rx[6..0] 0,	pc <- pc + 1
	lsr Rx	0111 Rx 11	Rx <- 0 Rx[7..1],	pc <- pc + 1
memória	push Rx	1000 Rx 00	MEM[SP] <- Rx,	pc <- pc + 1, sp <- sp – 1
	pop Rx	1000 Rx 01	Rx <- MEM[SP+1],	pc <- pc + 1, sp <- sp + 1
	ld Rx, 0x--	1000 Rx 11	Rx <- MEM[PC+1],	pc <- pc + 2
	ldr Rx, [Ry]	1001 Rx Ry	Rx <- MEM[Ry],	pc <- pc + 1
	str Rx, [Ry]	1010 Rx Ry	MEM[Ry] <- Rx,	pc <- pc + 1
saltos	jmp 0x--	1011 00 00		pc <- MEM[PC+1]
	jmpr Rx	1011 Rx 01		pc <- Rx
	bz Rx	1011 Rx 10	if (zero)	pc <- Rx else pc <- pc + 1
	bnz Rx	1011 Rx 11	if (~zero)	pc <- Rx else pc <- pc + 1
	bcs Rx	1100 Rx 00	if (carry)	pc <- Rx else pc <- pc + 1
	bcc Rx	1100 Rx 01	if (~carry)	pc <- Rx else pc <- pc + 1
	beq Rx	1100 Rx 10	if (equal)	pc <- Rx else pc <- pc + 1
	bneq Rx	1100 Rx 11	if (~equal)	pc <- Rx else pc <- pc + 1
	bgt	1101 Rx 00	if (greater)	pc <- Rx else pc <- pc + 1
	bgez	1101 Rx 01	if (greater and zero)	pc <- Rx else pc <- pc + 1
	blt	1101 Rx 10	if (smaller)	pc <- Rx else pc <- pc + 1
	blez	1101 Rx 11	if (smaller and zero)	pc <- Rx else pc <- pc + 1
	halt	1111 00 00		pc <- pc

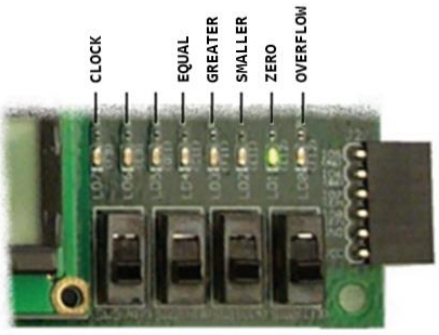
O registrador SP é o registrador de “stack pointer”, ou seja, é o ponteiro para uma pilha. A cada vez que colocamos algo na pilha (PUSH), o “stack pointer” é decrementado! E, cada vez que tiramos algo da pilha (PULL), o “stack pointer” é incrementado. Antes de utilizarmos o SP, devemos inicializá-lo! Geralmente, o “stack pointer” é inicializado com o maior valor de memória. No nosso caso seria 255, mas esta posição já está reservada para uso de I/O. Portanto, vocês devem inicializar o “SP” como 254. Para inicializar o “stack pointer” é utilizada uma diretiva assembly “equ”, por exemplo:




```
.equ   STACK, 0xFE   ; STACK POINTER = 254
loop:  ld  ra, 15    ; carrega 15 no registrador A
       ld  rb, 1     ; carrega 1  no registrador B
       add ra, rb    ; guarda no registrador A a soma 15+1
       jmp @loop     ; pulo incondicional
```

Instruções:

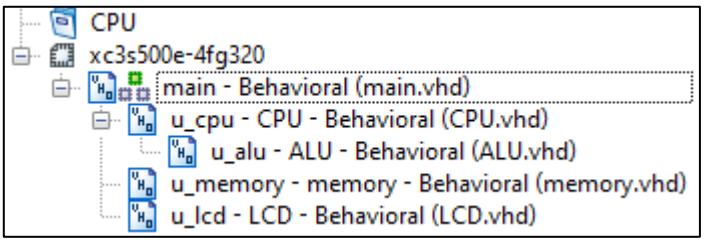
1. Seu computador deverá ser capaz de executar as instruções do programa a partir do momento em que se desligue o reset (via chave);
2. Tanto o OpCode como o complemento, quando aplicável, são de 8 bits cada, então o tamanho da palavra é sempre de 8 bits – o que também vale para os registradores;
3. A depender do OpCode, a decodificação da instrução deve ensejar em fazer ao todo, uma ou duas buscas a cada fetch. Por exemplo: Fetch, Decode1, Decode2, Execute, Write;
4. Sua CPU deverá ter minimamente quatro registradores, A, B, C e D;
5. Os sinais da ALU devem ser mostrados nos LEDs discretos e assim devem permanecer até que uma nova operação lógica/aritmética seja realizada (ver imagem);



6. O conteúdo do Registrador de Instruções (IR) deve ser permanentemente espelhado no LCD (é para ter o nome da instrução escrita!);
7. Na segunda linha do display LCD, é para apresentar o valor posição 255 da memória RAM; (0, 1, 2, 3, ..., 254, 255)
8. A ALU deverá fazer operações de soma, subtração e operações lógicas;
9. **Lembre-se:** que só a ALU sabe fazer conta! Então instruções como “INC”, “DEC”, etc., devem usar a ALU para tal;
10. Seu código deve ser bem comentado!

Importante:

1. Reduza o clock para que cada instrução seja visivelmente carregada. Recomenda-se utilizar ~2 segundos, para que seja possível acompanhar o passo a passo das transições.
    a. O clock da CPU é que deve ser reduzido!
    b. O clock do LCD é o mesmo clock da FPGA (50 MHz)
2. Faça um código bem estruturado, com componentes instanciados, comentários, etc. Por exemplo:



3. O professor pode entregar um código surpresa que deverá ser carregado na memória RAM e executado. Portanto, tudo deve estar funcionando perfeitamente.

Entrega:

A entrega deverá ser feita por e-mail, contendo o relatório e todo o projeto em um arquivo zip (código em VHDL, constraint file e um testbench pronto para simulação).

O relatório deverá conter, no mínimo:

- O desenho do caminho de dados
- Descrição do funcionamento das unidades (código), incluindo a máquina de estados da unidade de controle (UC).
- A explicação do funcionamento da arquitetura implementada
- Programas escritos por vocês para testar o processador (em assembly), explicando o conteúdo do código (seu funcionamento) assim como comentários sobre os resultados obtidos. Por exemplo: implementação da sequência de Fibonacci.

No dia da apresentação, vocês deverão mostrar um programa rodando na CPU que vocês implementaram. Vocês devem ter uma tabela apresentando o passo-a-passo das instruções que devem ser mostradas no LCD, dos LEDs (ALU) e do conteúdo da posição 255 de memória! O relatório deve conter uma explicação detalhada dos trechos de códigos e da forma que a implementação funciona.