Computação Digital
Trabalho Final

Felipe Calliari
<calliari@puc-rio.br>

Sumário

1 Objetivo

O objetivo deste projeto é consolidar os conhecimentos adquiridos na disciplina de Computação Digital por meio da criação de um processador (CPU) implementado em FPGA. O sistema deverá ser capaz de executar um conjunto predeterminado de instruções utilizando memória RAM estática, ALU, e interface de entrada/saída (LEDs e LCD).

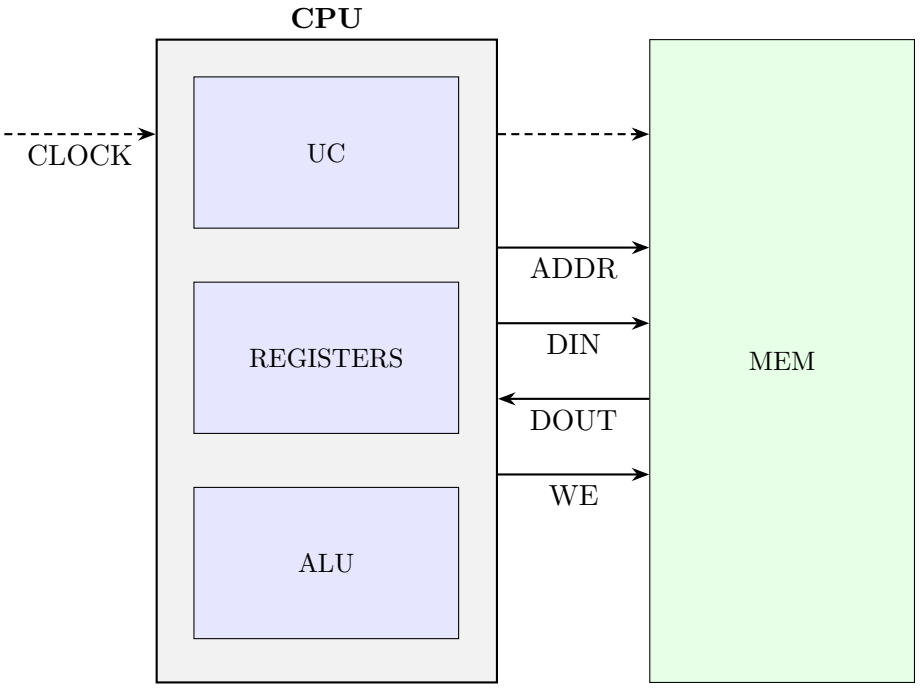


Figura 1.1: Exemplo de diagrama de blocos.

2 Arquitetura (ISA)

2.1 Componentes

O processador deverá conter:

- 4 registradores de propósito geral: `regA`, `regB`, `regC`, `regD`
- Registradores especiais: `IR`, `PC`, `SP`, `MBR`, `MAR`
- Memória RAM com 256 posições de 8 bits
- ALU assíncrona com suporte a operações aritméticas e lógicas
- Unidade de Controle (UC) com máquina de estados (implementa os ciclos *fetch*, *decode*, *execute* e *write* da CPU)

2.1.1 Formato das Instruções

Cada instrução ocupa 1 byte (8 bits). Algumas instruções requerem um byte adicional para endereço imediato. A decodificação deve identificar o tipo da instrução e atuar conforme os ciclos: *fetch*, *decode*, *execute* e *write*.

2.2 Tabela de Instruções

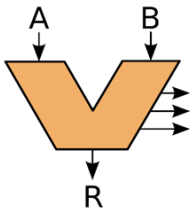
	Assembly	Opcode	Descrição
aritmético	`add Rx, Ry`	0000 Rx Ry	Rx ← Rx + Ry; PC ← PC + 1
	`sub Rx, Ry`	0001 Rx Ry	Rx ← Rx - Ry; PC ← PC + 1
	`inc Rx`	0010 Rx 00	Rx ← Rx + 1; PC ← PC + 1
	`dec Rx`	0010 Rx 01	Rx ← Rx - 1; PC ← PC + 1
		0010 Rx 10	–
		0010 Rx 11	–
lógico	`and Rx, Ry`	0011 Rx Ry	Rx ← Rx & 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] Rx[7]; PC ← PC + 1
	`ror Rx`	0111 Rx 01	Rx ← Rx[0] 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
	`st Rx, ADDR`	1000 Rx 10	MEM[ADDR] ← Rx; PC ← PC + 2
	`ld Rx, ADDR`	1000 Rx 11	Rx ← MEM[ADDR]; 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
	`mov Rx, Ry`	1011 Rx Ry	Rx ← Ry; PC ← PC + 1
saltos	`jmp ADDR`	1100 00 00	PC ← MEM[ADDR]
	`jmpr Rx`	1100 Rx 01	PC ← Rx
	`bz Rx`	1100 Rx 10	if zero: PC ← Rx else: PC ← PC + 1
	`bnz Rx`	1100 Rx 11	if not-zero: PC ← Rx else: PC ← PC + 1
	`bcs Rx`	1101 Rx 00	if carry: PC ← Rx else: PC ← PC + 1
	`bcc Rx`	1101 Rx 01	if not-carry: PC ← Rx else: PC ← PC + 1
	`beq Rx`	1101 Rx 10	if equal: PC ← Rx else: PC ← PC + 1
	`bneq Rx`	1101 Rx 11	if not-equal: PC ← Rx else: PC ← PC + 1
	`bgt Rx`	1110 Rx 00	if greater: PC ← Rx else: PC ← PC + 1
	`blt Rx`	1110 Rx 01	if smaller: PC ← Rx else: PC ← PC + 1
	`bneg Rx`	1110 Rx 10	if negative: PC ← Rx else: PC ← PC + 1
	`nop`	1111 00 00	Nenhuma operação; PC ← PC + 1
	`halt`	1111 11 11	Fim do programa; PC ← PC

Onde, `Rx` e `Ry` podem ser quaisquer registradores **A**, **B**, **C** ou **D**. As operações `st`, `ld` e `jmp` usam `ADDR` que é o endereço armazenado no byte seguinte (em `pc+1`), ou seja, existe uma leitura adicional de memória. Por isso, ao final dessas instruções `pc` é incrementado duas vezes (`pc+2`).

2.3 ALU

A ALU é uma "unidade lógica e aritmética" é um circuito digital assíncrono que realiza operações como soma, subtração e operações aritméticas. A ALU deve receber as entradas `ALU_A`, `ALU_B` e `CMD` para realizar determinada operação apenas quando o Opcode for uma instrução de operações lógicas e aritméticas.

Como a ALU é assíncrona, ela não recebe o sinal de "clock". Portanto, a ALU só é responsável por realizar o que foi pedido e a UC é responsável por salvar: o resultado da operação no devido registrador e as flags em Flip-Flops dentro da própria UC (Unidade de Controle) após as instruções da ALU (add, sub, and, etc), isto é, ao final da operação no ciclo de "*execute*".



Nota: Nossa ALU só sabe realizar soma e subtração com valores *unsigned*! Não existem Opcodes para operações com *signed*, cabe ao programador saber que tipo de dado foi utilizado e interpretá-lo de forma adequada.

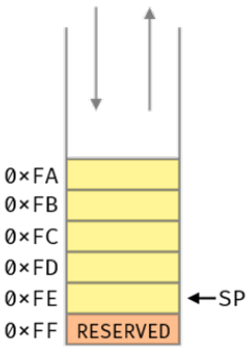
2.3.1 Flags da ALU

A ALU deverá ter as seguintes *flags*:

- **ZERO**: Resultado == 0
- **NEGATIVE**: Bit mais significativo do resultado é 1
- **OVERFLOW**: Overflow ou underflow (**carry**)
- **EQUAL**: A == B
- **GREATER**: A > B
- **SMALLER**: A < B

2.4 Stack Pointer

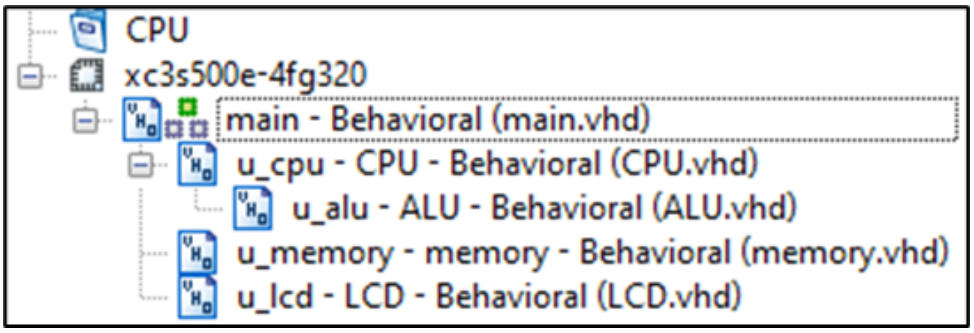
O registrador SP é o registrador de "stack pointer", ou seja, é o ponteiro para uma pilha. A cada vez que colocamos algo na pilha (`PUSH`), o "stack pointer" é decrementado! E, cada vez que tiramos algo da pilha (`PULL`), o "stack pointer" é incrementado. Antes de utilizarmos o SP, devemos inicializá-lo! 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 dentro da UC.



3 Instruções

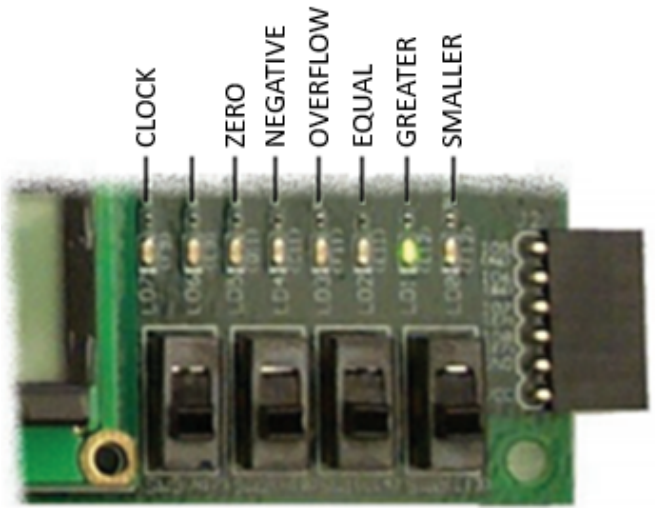
1. O processador deverá ser capaz de executar as instruções do programa, armazenadas na RAM, assim que o sinal de "reset"for desativado.
2. Idealmente, a implementação deve considerar as etapas *fetch*, *decode*, *execute* e *write*. Caso ache necessário implementar mais uma etapa de decodificação implemente: *fetch*, *decode1*, *decode2*, *execute* e *write*.

3. Todos os sinais da ALU são assíncronos.
4. Inicialização do SP obrigatória (254).
5. Reduza o clock da CPU para que cada instrução executada possa ser visualizada. As instruções devem ser visivelmente executadas em ∼2 segundos para que seja possível acompanhar o passo-a-passo das transições.
6. O clock do LCD é o clock da FPGA (50 MHz).
7. Faça um código bem estruturado, com componentes instanciados, comentários, etc. Por exemplo:



3.1 Interface com Usuário

- Os LEDs devem refletir as flags, armazenadas nos FFs da UC, da última operação lógica/aritmética (ALU).
- O display LCD deve mostrar:
  - Linha 1: nome da instrução atual (*assembly*)
  - Linha 2: valor da posição 255 da memória



4 Entrega e Avaliação

4.1 Conteúdo do Relatório

- Relatório em PDF com ao menos 15 páginas
  - Diagrama do caminho de dados
  - Descrição detalhada do funcionamento de cada unidade
  - Máquina de estados da UC
  - Código Assembly comentado para teste (ex.: Fibonacci)
  - Explicação dos testes, funcionamento do display e LEDs
- Em arquivo "zip":
  - Código VHDL completo
  - Arquivo de constraints
  - Testbench pronto para simulação

4.2 Apresentação

Durante a apresentação, será solicitado:

- Execução de um programa demonstrativo na CPU
- Tabela com passo a passo das instruções

Nota: O professor poderá fornecer um programa surpresa a ser carregado na RAM.

4.3 Entrega

Este trabalho deve ser realizado **individualmente** ou em **duplas**. O relatório deve ser entregue no formato PDF, junto com a pasta do projeto e as bibliotecas (se houver) em um **arquivo ZIP** até a data de entrega pelo **Moodle**. Como parte da entrega do relatório, tire fotos e/ou faça um vídeo da implementação do projeto e poste no YouTube como "*não listado*", se preferirem. Antes de zipar o projeto apague os arquivos de simulação, relatórios, etc. Para isto, acesse:

- *Project* → *Cleanup Project Files...*
- *Project* → *Archive...*