SAMUEL FELIPE DOS SANTOS

PROJETO DE UM SISTEMA OPERACIONAL

Relatório apresentado à Universidade Federal de São Paulo como parte dos requisitos para aprovação na disciplina de Laboratório de Sistemas Computacionais: Sistemas Operacionais.

Professor: Tiago de Oliveira

São José dos Campos - SP

2016

Lista de Figuras

Lista de Tabelas

Sumário

1 Introdução

1.1 Contextualização

A Unidade Central de Processamento, ou CPU, é um dos componentes mais vitais do computador, e também um dos que evoluiu de maneira mais rápida.

Tem a função de executar um conjunto de instruções lógicas definidas pelo software.

Para aumentar a produtividade de um programador, e permitir que programas mais complexos sejam produzidos, faz-se o uso de Compiladores.

Um compilador tem a função de traduzir uma linguagem para outra, e pode ser utilizado para traduzir uma linguagem de mais alto nivel para o conjunto de instruções de um determinado processador.

Para facilitar a interação entre o usuário e o sistema computação, existe um programa denominado Sistema Operacional.

O Sistema Operacional tem três principais objetivos: Executar programas do usuário e solucionar seus problemas; Tornar o uso do Sistema Computacional Conveniente; E utilizar o hardware do computador de maneira eficiente.

1.2 Objetivos

Objetivos Gerais

Este projeto tem como objetivo a implementação de um sistema operacional capaz de gerenciar processos, memoria e unidades de entrada e saída.

O sistema operacional será desenvolvido na linhagem de programação C-, fazendo o uso de um compilador para traduzi-lo para o conjunto de instruções do sistema computacional onde este sistema será executado.

O sistema computacional foi mapeado em uma FPGA (Field Programmable Gate Array) Cyclone IV EP4CE115F29C7.

Objetivos Especificos

O projeto foi dividido em diversas etapas menores listadas abaixo.

1. Ajustes no Compilador e projeto de algoritmos;
2. Definição do Sistema Operacional a ser projetado, técnicas e algoritmos a serem virtualizados;
3. Adaptações necessárias na plataforma de hardware e implementação da Bios, HD e Sistema de Comunicação entre Componentes;
4. Finalização da implementação do Sistema Operacional.

Atualmente o projeto se encontra em sua segunda etapa.

2 Fundamentação Teórica

2.1 CPU

A CPU (Central Processing Unit), ou UCP (Unidade Central de Processamento) em português, também chamado de processador, funciona como o cérebro do computador, sendo capaz de realizar cálculos e operações de acordo com um programa.

Pode ser divido em Unidade de Processamento e Unidade de Controle.

2.2 Unidade de Processamento

É o local onde o processamento ocorre, sendo formada por diversas unidade, como a ULA (Unidade Lógica Aritimética), memórias, banco de registradores e etc.

2.3 Unidade de Controle (UC)

Tem a função de controlar as diversas unidades que compoem a Unidade de processamento, definindo como elas devem funcionar de acordo com as entradas e instruções.

2.4 Unidade de Controle Hardwired e Microprogramada

Uma Unidade de Controle Hardwired é formada por elementos que possuem lógica sequencial, onde suas saídas, os sinais de controle, são definidos pelas entradas atuais e anteriores. Caso seja necessário alterar ou incrementar o conjunto de instruções, é necessário mudar a UC também, geralmente tem desempenho melhor que a UC Microprogramada.

A Unidade de Controle Microprogramada é mais flexível, funcionando como uma memória de bits de controle e parte do fluxo de controle para sequências desses padrões.

Dessa forma, uma UC microprogramada pode ser programada através de um "microprograma", para reconhecer padrões de controle diferentes.

2.5 Circuito Combinacionais e Sequenciais

Circuitos combinacionais são aqueles em que as saídas dependem exclusivamente das entradas atuais.

Já em um circuito sequencial, as saídas dependem dos estados anteriores, possuindo assim, elementos de memória.

2.6 Máquina de Estados

"Um sistema lógico que exibe uma sequência de estados condicionados pela lógica interna e as entradas externas; qualquer circuito sequencial que exibe uma sequência especifica de estados". (Floyd, 2009, p.496)

Máquina de Estados Finitos de Moore

Trata-se de uma máquina de estados com uma quantidade finita de estados, onde as saídas dependem apenas dos estados armazenados.

Máquina de Estados Finitos de Mealy

Trata-se de uma máquina de estados com uma quantidade finita de estados, onde as saídas dependem dos estados armazenados atualmente, e das entradas.

2.7 Conjunto de Instruções

Conjunto de operações que um processador, microprocessador, CPU ou outro periférico programável é capaz de realizar.

2.8 RISC e CISC

Reduced Instruction Set Computer, ou seja Computador de Conjunto de Instruções Reduzidas trata-se de uma linha de arquitetura de computadores que favorece um conjunto simples e pequeno de intruções, que levam tempo aproximadamente igual para serem executadas.

Alguns Exemplos de processadores que seguem essa arquitetura são, o MIPS, POWER PC,.

A maioria dos processadores atuais utilizam um modelo hibrido entre RISC e CISC (Complex Instruction Set Computer).

2.9 Arquitetura LOAD/STORE

Trata-se de uma arquitetura em que apenas as instruções Load e Store tem acesso a memória.

2.10 FPGA

"A Field-Programmable Gate Array (FPGA) é um dispositivo semicondutor que pode ser programado após a fabricação. Em vez de ser restrito a qualquer função de hardware pré-determinada, um FPGA permite programar recursos e funções do produto, se adaptar às novas normas, e reconfigurar hardware para aplicações específicas, mesmo depois que o produto foi instalado em campo, daí o nome de "field-programmable ". Você pode usar um FPGA para implementar qualquer função lógica que um application-specific integrated circuit (ASIC) poderia realizar, mas a capacidade de atualizar a funcionalidade após o envio oferece vantagens para muitas aplicações". (Altera, 2015)

2.11 Linguagem de Descrição de Hardware

"Uma linguagem de descrição de hardware descreve o que um sistema faz e como; Um sistema descrito em linguagem de hardware pode ser implementado em um dispositivo programável FPGA (Field Programmable Gate Array) ou um dispositivo ASIC (Aplication Specific Integrated Circuit), permitindo o uso em campo do Sistema". (CASILLO, 2015)

2.12 Expressão Regular

Expressão regular se trata de uma notação que permite identificar uma sequência de caracteres.

"Uma expressão regular é constituida de expressões regulares mais simples usando-se um conjunto de regras de definição. Cada expressão regular r denota uma linguagem $L(r)$. As regras de definição especificam como $L(r)$ é formado através da combinação, em várias formas. das linguagens denotadas pelas subexpressões de r". (AHO et al., 2008, p.43)

2.13 Gramática Livre de Contexto

Gramáticas livres de contexto podem ser usadas para definir linguagens que fazem uso de recursão, como as linguagens de programação.

"Uma gramática livre de contexto possui quatro componentes: 1 - Um conjunto de tokens, conhecidos como símbolos terminais. 2 - Um conjunto de não-terminais. 3 - Um conjunto de produções, onde uma produção consiste em um não-terminal, chamado de lado esquerdo da produção, uma seta e uma sequência de tokens e/ou não-terminais, chamados de lado direito da produção. 4 - Uma designação a um dos não-terminais como símbolo de partida."(AHO et al., 2008, p.12)

2.14 Compilador

"Posto de forma simples, um compilador é um programa que lê um programa escrito numa linguagem - a linguagem fonte - e o traduz num programa equivalente numa outra linguagem - a linguagem alvo ... Como importante parte desse processo de tradução, o compilador relata a seu usuário a presença de erros no programa fonte".(AHO et al., 2008, p.1)

2.15 Sistema Operacional

"Um sistema operacional é um programa que atua como intermediário entre o usúario e o hardware de um computador. O propósito de um sistema operacional é fornecer um ambiente no qual o usuário possa executar programas. O principal objetivo de um sistema operacional é portanto tornar o uso do sistema de computação conveniente. Uma meta secundaria é usar o hardware do computador de forma eficiente". (Silberschatz, 2008, p.3)

2.16 Processos

"Informalmente, um processo é um programa em execução. Um processo é mais do que o código do programa, que às vezes è chamado seção de texto. Também inclui a atividade corrente, conforme representado pelo valor do contador do programa e 0 conteúdo dos registradores do processador. Um processo geralmente inclui a pilha de processo, que contém dados temporários (como parâmetros de métodos, endereços de retorno e variáveis locais) c uma seção de dados, que contém variáveis globais". (Silberschatz, 2008, p.63)

2.17 Escalonamento da CPU

"O escalonamento de CPU é a base dos sistemas operacionais multiprogramados. Ao alternar a CPU entre os processos, o sistema operacional pode tornar o computador produtivo". (Silberschatz, 2008, p.95)

Principais Critérios

Os principais critérios para escalonamento da CPU são,:

- Utilização da CPU, ou seja, manter a CPU ocupado o maior tempo possível;
- Throughput, o número de processos completos em uma unidade de tempo;
- Tempo de retorno, quanto tempo leva para executar um processo, de sua submissão até sua conclusão;
- Tempo de Espera, o tempo que um processo espera na fila de prontos;
- Tempo de resposta, tempo entre a submissão do processo até a primeira resposta ser produzida.

Preempção

"A estratégia de permitir processos que são logicamente executáveis serem temporariamente suspensos é chamada agendamento preemptivo, e está em oposição ao método de executar até concluir dos antigos sistemas de lote. Esse método também é chamado de agendamento não-preemptivo". (Tanenbaum and Woodhull, 2009, p.70)

Principais Algoritmos

Serão listados abaixo alguns dos principais algoritmos de escalonamento da CPU vistos na literatura.

- **Round Robin:** *"A cada processo é atribuída um intervalo de tempo, chamado de seu quantum, durante o qual lhe é permitido executar. Se o processo ainda está executando no fim do quantum, é feita a preempção da CPU e ela é dado a outro processo. Se o processo bloqueou ou terminou antes de o quantum ter passado, é feita a comutação da CPU quando o processo bloqueia naturalmente".* (Tanenbaum and Woodhull, 2009, p.70)

  Trata-se de um algoritmo simples, é grande parte de seu desempenho esta relacionado ao tamanho do quantum.

- **Prioridade:** *"A necessidade de levar em conta fatores externos conduz ao agendamento por prioridade. A ideia básica é simples e direta: a cada processo é atribuida uma prioridade, e o processo executável com a maior prioridade recebe permissão para executar".* (Tanenbaum and Woodhull, 2009, p.71)

  Ações podem ser tomadas para evitar que um processo de alta prioridade execute indefinidamente, como por exemplo, a cada determinada unidade de tempo, diminui-se a prioridade do processo.

- **Multiplas Filas:** Variação do agendamento por prioridade, são definidas diferentes filas para diferentes prioridades, executando-se entre os processos da fila de prioridade mais alta, se esta estiver vazia, executa-se os algoritmos da fila de prioridade menor.

  Dentro de cada fila, é executado o algoritmo Round Robin. Pode-se também mover os processos entre as filas, por exemplo, mudar para uma fila de prioridade mais baixa um processo que já foi executado pela CPU por várias vezes e não foi concluído.

- **Job Mais Primeiro:** *"Quando vários jobs igualmente importantes estão na fila de entrada esperando para ser iniciados, o agendador deve utilizar job mais curto primeiro"*. (Tanenbaum and Woodhull, 2009, p.72)

  Neste algoritmo, é escolhido o processo que terá menor tempo de execução para serem executados primeiro.

  Problemas deste algoritmo são que é necessário saber o tempo de execução de um algoritmo antes de sua execução, e seu tempo de resposta médio pode ser grande.

2.18 Gerenciamento de Memória

Sistemas de gerenciamento de memória podem ser divididos em dois tipos, os que movem processos entre a memória principal e o disco, e os que não fazem isto.

Os que não fazem são mais simples, e devido a este fato são os que serão estudados aqui.

A seguir serão apresentados alguns métodos de gerenciamento de memória.

Monoprogramação sem Troca ou Paginação

"O esquema mais simples possível de gerenciamento de memória é executar somente um programa por vez, compartilhando a memória entre esses programa e o sistema operacional". (Tanenbaum and Woodhull, 2009, p.212)

Três possíveis maneiras de se fazer a divisão da memória nesse caso são: *"O sistema operacional pode estar na parte inferior da mamória RAM (Random Access Memory, Memória de Acesso Aleatório) , ... ou pode estar na ROM (Reead Only Memory, Memória Apenas de Leitura) na parte superior ... , ou os drivers de dispositivo podem estar em uma ROM e o restante do sistema em RAM na parte inferior"*. (Tanenbaum and Woodhull, 2009, p.212)

Multiprogramação com Partições Fixas

Mantém múltiplos programas na memória ao mesmo tempo, *"A maneira mais fácil de alcançar a multiprogramação é simplesmente dividir a memória em n partições (possivelmente desiguais). Esse particionamento pode ser feito por exemplo, manualmente, quando o sistema é iniciado"*. (Tanenbaum and Woodhull, 2009, p.212)

Problemas com esse método é a fragmentação interna, isto é espaço não utilizado por um programa dentro de sua partição e o fato de partições grandes ou pequenas podem ficar sempre vazias devido ao fato de não haverem programas que se encaixem nelas.

Para solucionar esses problemas, deve-se fazer uso de paginação e segmentação.

Paginação

"A paginação é um esquema que permite que o espaço de endereçamento físico de um processo seja não-contíguo. A paginação evita o problema de ajustar os pedaços de memória dos mais diversos tamanhos no armazenamento auxiliar, um problema sério que afetou a maioria dos esquemas de gerência de memória anteriores. Quando alguns fragmentos de código ou dados que residem na memória principal precisam ser descarregados (operação de sivap otit), deve haver espaço disponível no armazenamento auxiliar. Os problemas de fragmentação discutidos em relação à memória principal também prevalecem com o armazenamento auxiliar, exceto pelo fato de que o acesso é muito mais lento,

por isso é impossível fazer a compactação. Devido às vantagens em relação aos métodos anteriores, a paginação em suas muitas formas é utilizada com frequência em muitos sistemas operacionais". (Silberschatz, 2008, p.189)

Segmentação

Um aspecto importante da gerência de memória que se tornou inevitável com a paginação é a separação da visão de usuário da memória e a memória física real. A visão de usuário da memória não é igual à memória física real. A visão de usuário é mapeada na memória física. O mapeamento permite a diferenciação entre memória lógica e física. (Silberschatz, 2008, p.200)

2.19 Gerênciamento de Entrada e Saída

Será apresentado a seguir uma visão geral de como um sistema operacional gerência dispositivos de entrada e saída.

Dispositivos de Entrada e Saída

Podem ser divididos, de maneira geral, em duas categorias (apesar de existirem casos específicos que não se encaixam perfeitamente em nenhuma das classificações).

- **Dispositivos de Bloco:** Dispositivo que armazena informação em blocos de tamanho fixo, sendo possível acessar cada um deles de maneira independente dos demais.

  Exemplo: Disco Rígido

- **Dispositivo de Caractere:** Dispositivo que aceita ou entrega uma sequência de caracteres, não respeitando blocos e não sendo possível realizar pesquisa sobre ele ou endereçá-lo.

  Exemplo: Mouse, Teclado e Impressora.

Acesso Direto a Memória (DMA)

Componente de hardware que permite que dispositivos de I/O, principalmente dispositivos de bloco, escrevam seus dados direto na memória.

Software

As principais metas do gerenciamento de entrada e saída pelo software são:

- Criar uma independência de dispositivo, isto é manipular dispositivos diferentes com chamadas similares;
- Atribuição uniforme de nomes;
- O tratamento de erros.

Outra questão importante é transferência síncrona ou assíncrona de dados.

A transferência síncrona é baseada em bloqueios, enquanto a assíncrona se baseia em interrupções.

A transferência assíncrona deixa o projeto do software mais simples.

3 Termos Utilizados

Na tabela 1 estão definidos os termos que ser ao utilizados nas seções seguintes.

Tabela 1: Termos Utilizados

Termo	**Descrição**
Opcode	Código que define as operações, possui 5 bits.
R1, R2 e R3	Endereços de 5 bits que definem registradores ou as saídas
ENT	Registrador de 32 bits que armazena as entradas (botões e chaves)
SAIDAS	Conjunto de 32 registradores de 32 bits que armazenam os valores das saídas
$[x]$	Valor de 32 bits armazenado pelo registrador de endereço x
I	Imediato de 17 bits
E	Endereço ou valor de 22 bits
-	Conjunto aleatório de bits
PC	Contador que marca qual instrução esta sendo executada (22 bits)
RAM_DADOS	Refere-se a saída da memória de dados (32 bits)

4 O Processador

Esta seção irá descrever de forma geral a arquitetura do processador utilizado, o código fonte esta disponível em:
https://github.com/felipe-samuel/Processador-Verilog

4.1 Visão Geral da Arquitetura

A arquitetura do processador utilizado possui um conjunto de instruções reduzidas, similar ao MIPs possuindo apenas 28 instruções, cada uma com o tamanho de 32 bits, possui uma arquitetura load/store.

Possui um funcionamento monociclo, ou seja executará uma instrução a cada ciclo de clock.

O processador pode ser dividido em duas Unidade, uma de controle, e outra de processamento, nas Figuras 1 e 2, os elementos pertencentes a estas unidades aparecem com as cores vermelha e azul, respetivamente.

4.2 Elementos da Unidade de Processamento

A unidade de processamento possui os seguintes elementos.

- **Program Counter (PC):** Registrador de 22 bits que armazena, o endereço da instrução que está sendo executada na memória de instruções.
- **Unidade Lógica Aritmética (ULA):** Tem a função de realizar operações lógicas e aritméticas e comparações (maior, menor, igual e diferente).

  Tem duas entradas de dados de 32 bits, que definem os dois operandos, e uma entrada de controle de 4 bits, que define que operação será realizada, e 2 saídas, uma de 32 bits, que é o resultado de operações aritméticas e lógicas, e a saída de 1 bit L, que é o resultado de comparações (0 para falso e 1 para verdadeiro).
- **Memória de Instruções:** Trata-se de uma memória vetorial com capacidade de armazenar 4194304 instruções de 32 bits. Possui uma entrada (EMI), que é o endereço no vetor no qual o dado qual ocorrerá uma escrita de dados, ou será devolvido em sua saída de 32 bits.

  A entrada de controle MIW define se haverá escrita ou não.
- **Memória de Dados:** Armazena dados de 32 bits, funcionamento igual a memória de instruções.
- **Banco de Registradores:** Vetor de 32 registradores, possui 3 entradas, RE (Endereço do registrador em que ocorrerá a escrita de dados), RL1 (endereço do registrador cujo conteúdo será devolvido na saída Rout1 de 32 bits), RL2 (endereço do registrador cujo conteúdo será devolvido na saída Rout2 de 32 bits).
- **Unidade de Entradas:** Vetor de 32 registradores, cada um de 32 bits, que armazenam o valor de dispositivos de entradas conectados a ele de forma síncrona com o sinal de clock, a entrada e tem a função de fazer a selecionar qual dos registradores será devolvido como saída.

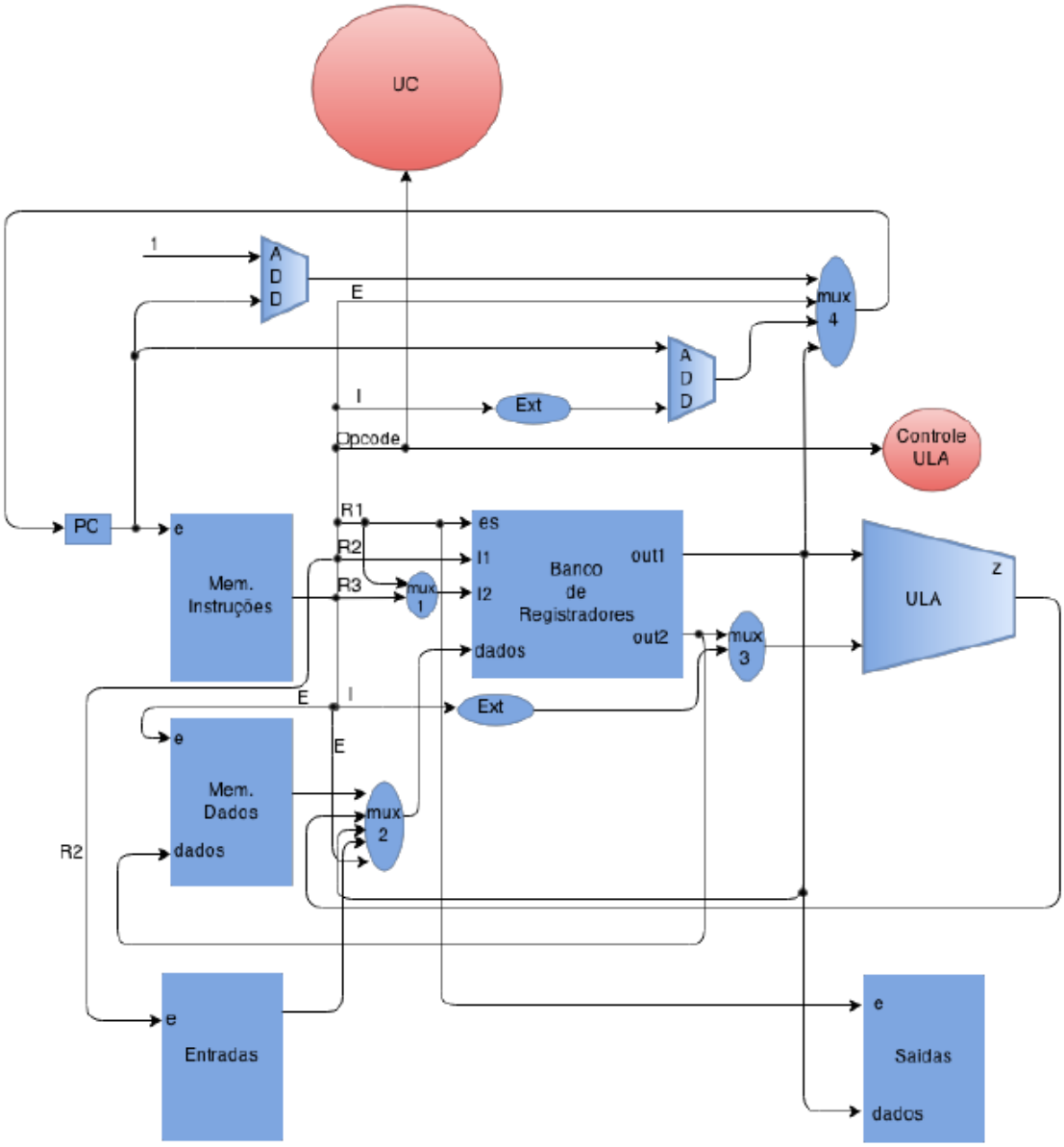

Figura 1: Diagrama da CPU

- **Unidade de Entradas:** Vetor de 32 registradores, cada um de 32 bits, que armazenam o valor de dispositivos de entradas conectados a ele de forma síncrona com o sinal de clock, a entrada e tem a função de fazer a selecionar qual dos registradores será devolvido como saída.
- **Unidade de Saídas:** Vetor de 32 registradores, armazena valores de saídas que podem ser lidos por dispositivos externos, a entrada e tem a função de realizar a seleção sobre em qual registrador será escrito o valor da entrada dados.

  O sinal de controle SW define se haverá escrita durante este ciclo de clock.
- **Somadores (ADD):** Somam os dois valores de suas entradas;

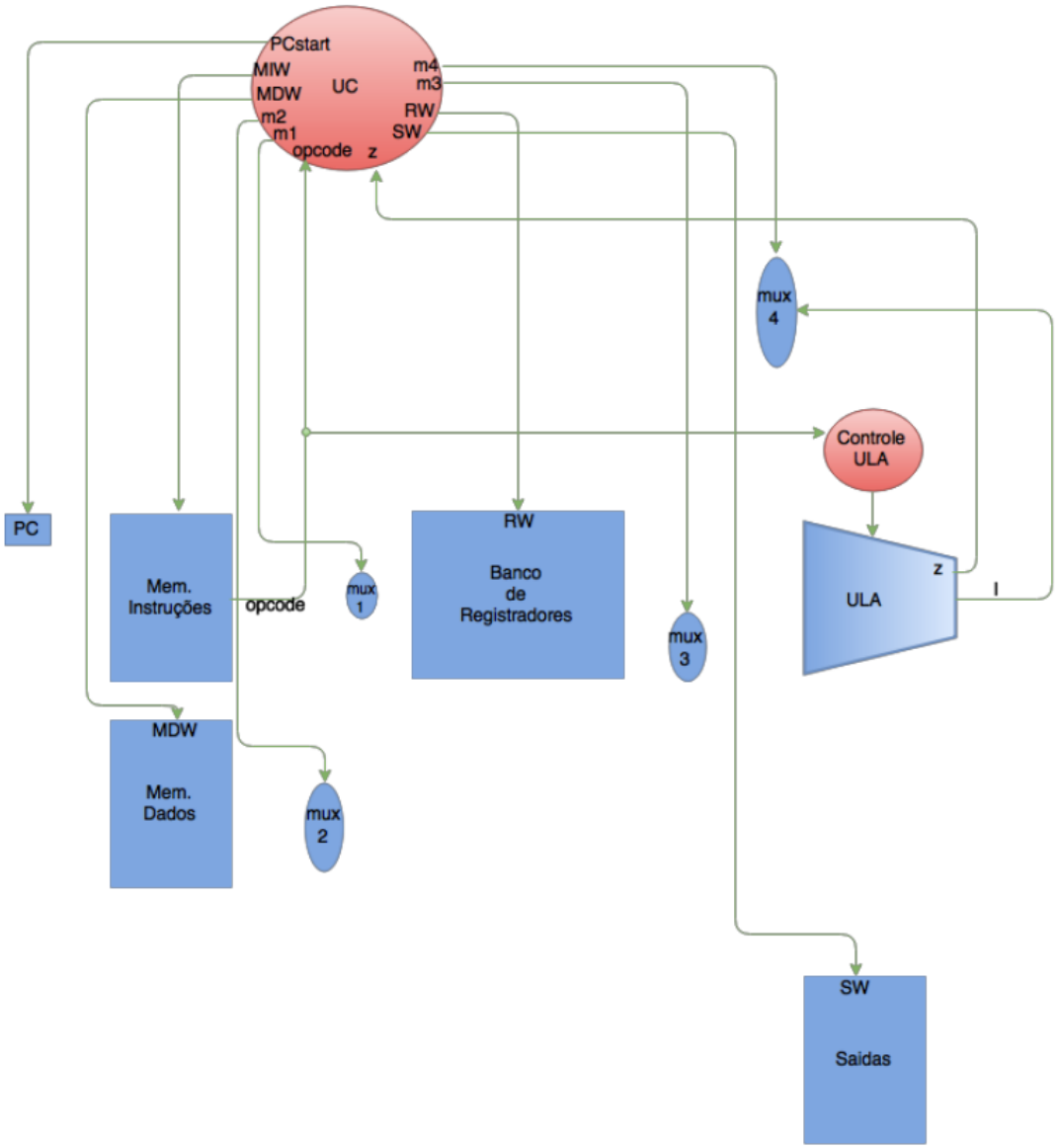

Figura 2: Diagrama dos Sinais de Controle da CPU

- **Extensores de Bits (Ext):** Aumentam a quantidade de bits do sinal de entrada acrescentando zeros.
- **Multiplexadores (m):** De acordo com o sinal de controle recebido, a saída será igual a uma de suas entradas.

  Controle da ULA: Recebe o Opcode como entrada, e define qual operação a ULA terá de realizar.

4.3 Elementos da Unidade de Controle

- **Unidade de Controle Geral (UC):** Trata-se de uma máquina de estado, que recebe como entrada o Opcode, e tem como saída diversos sinais de controle, que controlam a escrita de dados nas memória, banco de registadores, entradas e saídas,

e o funcionamento dos multiplexadores.

- **Unidade de Controle da ULA:** Recebe o Opcode, e determina qual operação a ULA deverá realizar durante aquele ciclo de clock.

4.4 Formato de Instruções

Serão aceito 3 formatos diferentes de instruções.

- Formato 1:

Opcode[31:27]	R1[26:22]	R2[21:17]	R3[16:12]	-[11:0]

- Formato 2:

Opcode[31:27]	R1[26:22]	R2[21:17]	I/-[16:0]

- Formato 3:

Opcode[31:27]	R1/-[26:22]	E/-[21:0]

4.5 Conjunto de Instruções

A tabela 2 a seguir lista o conjunto de instruções que o processador é capaz de realizar.

Tabela 2: Conjunto de Instruções

Opcode	**Instrução**	**Funcionamento**	**formato**
Operações aritiméticas			
00000	Add[R1,R2,R3]	$[R1] <= [R2] + [R3]$	1
00001	AddI[R1,R2,I]	$[R1] <= [R2] + I$	2
00010	Sub[R1,R2,R3]	$[R1] <= [R2]–[R3]$	1
00011	SubI[R1,R2,I]	$[R1] <= [R2] - I$	2
00100	Mult[R1,R2,R3]	$[R1] <= [R2] * [R3]$	1
00101	MultI[R1,R2,I]	$[R1] <= [R2] * I$	2
Operações lógicas e de comparação			
00110	And[R1,R2,R3]	$[R1] <= [R2] and [R3]$	1
00111	AndI[R1,R2,I]	$[R1] <= [R2] and I$	2
01000	Or[R1,R2,R3]	$[R1] <= [R2] or [R3]$	1
01001	OrI[R1,R2,I]	$[R1] <= [R2] or I$	2
01010	Not[R1,R2]	$[R1] <= NOT([R2])$	2
01011	Slt[R1,R2,R3]	$[R1] <= 1, Se[R2] < [R3]$ $[R1] <= 0, Se[R2] >= R3$	1
01100	SltI[R1,R2,I]	$[R1] <= 1, Se[R2] < I$ $[R1] <= 0, Se[R2] >= I$	2
Operações de deslocamento de bits			
01101	Sll[R1,R2]	$[R1] <= SLL([R2])$	2
01110	Slr[R1,R2]	$[R1] <= SLR([R2])$	2
Operações de tranferências de dados			
01111	Move[R1,R2]	$[R1] <= [R2]$	2
Operações de saltos			
10000	J[E]	$PC <= E$	3
10001	Jr[R1]	$PC <= [R1]$	3
10011	Beq[R1,R2,I]	$PC <= PC + I, Se[R1] = [R2]$	2
10100	Bnq[R1,R2,I]	$PC <= PC + I, Se[R1]! = [R2]$	2
10101	Bob[R1,R2,I]	$PC <= PC + I, Se[R1] > [R2]$	2
10110	Bos[R1,R2,I]	$PC <= PC + I, Se[R1] < [R2]$	2
Operações de acesso a memória interna			
10111	Load[R1,E]	$[R1] <= RAM\_DADOS(E)$	3
11000	Store[R1,E]	$RAM\_DADOS(E) <= [R1]$	3
Operações de acesso as entradas e saídas			
11001	Input[R1,R2]	$[R1] <= ENT(R2)$	2
11010	Output[R1,R2]	$SAIDAS(R1) <= [R2]$	2
Outras			
11011	SetR[R1,E]	$[R1] <= E$	3
11111	End[ ]	$PC <= PC$	3

5 O Compilador

O compilador foi implementado fazendo uso da linguagem de programação C++, em conjunto com as ferramentas Flex (para realizar a analise léxica) e Bison (para realizar a análise sintática).

Seu objetivo consiste em traduzir um código escrito na linguagem C- para o conjunto de instruções do Sistema Computacional, listando os erros léxicos, sintáticos e semânticos, caso encontrados.

Nesta seção há uma breve descrição de seu funcionamento, o código fonte deste projeto pode ser encontrado em:

https://github.com/felipe-samuel/compilador-c-

5.1 A linguagem C-

A linguagem de programação C- trata-se de uma simplificação da linguagem C, possuindo as seguintes limitações:

- Não podem haver declarações de protótipos de funções;
- Apenas um tipo do dados, os inteiros (int);
- Não há utilização de ponteiros;
- As variáveis devem ser declaradas no inicio do código ou da função;
- A entrada de dados é feita pela função input, e a saída pela função output.

5.2 Tokens

A Tabela 3 mostra a lista de tokens aceitos pela linguagem.

Tabela 3: Lista de Tokens Reconhecidos

Texto Reconhecido	**Token Retornado**
if	IF
else	ELSE
while	WHILE
return	RETURN
int	INT
void	VOID
+	SOMA
−	SUB
*	MULT
/	DIV
! =	DIFERENTE
==	IGUAL
>=	MAIOR_IGUAL
<=	MENOR_IGUAL
=	ATRIBUICAO
>	MAIOR
<	MENOR
(	PARENTESES_ABRE
)	PARENTESES_FECHA
[	COLCHETES_ABRE
]	COLCHETES_FECHA
{	CHAVES_ABRE
}	CHAVES_FECHA
;	PONTO_VIRGULA
,	VIRGULA
texto iniciado por uma letra, podendo conter letras e números	ID
número inteiro	INT_NUM
sequencia de caracteres diferente dos citados anteriormente	ERROR

5.3 Expressões Regulares

O Algoritmo 1 mostra as expressões regulares usadas nessa linguagem.

$$digito = [0-9]$$
$$letra = [a-zA-Z]$$
$$INT\_NUM = (+|-)?digito+$$
$$ID = letra + (letra|digito)*$$

Algoritmo 1: Expressões Regulares da Linguagem C-

5.4 Gramática Livre de Contexto

A gramática livre de contexto aceita por esta linguagem é mostrada no Algoritmo 2.

```
1  programa -> programa declaracao
2  declaracao -> var-declaracao | fun-declaracao | error
3  var-declaracao -> tipo-especificador var-decl mult-var-decl
       PONTO_VIRGULA
4  mult-var-decl -> VIRGULA var-decl mult-var-decl | /*entrada vazia*
5  var-decl -> ID | ID COLCHETES_ABRE INT_NUM COLCHETES_FECHA
6  tipo-especificador -> INT | VOID
7  fun-declaracao -> tipo-especificador ID PARENTESES_ABRE params
       PARENTESES_FECHA composto-decl | tipo-especificador error composto-
       decl
8  params -> param param-lista | VOID  | /*entrada vazia*/
9  param-lista -> VIRGULA param param-lista  | /* entrada vazia */
10 param -> tipo-especificador ID | tipo-especificador ID COLCHETES_ABRE
       COLCHETES_FECHA
11 composto-decl -> CHAVES_ABRE local-declaracao statement-lista
       CHAVES_FECHA
12 local-declaracao -> local-declaracao var-declaracao | local-declaracao
       error var-declaracao | /* entrada vazia */
13 statement-lista -> statement-lista statement | statement-lista error
       statement | /* entrada vazia */
14 statement -> expressao-decl | composto-decl | selecao-decl | iteracao-
       decl | retorno-decl
15 expressao-decl -> expressao PONTO_VIRGULA | PONTO_VIRGULA
16 selecao-decl -> IF PARENTESES_ABRE expressao PARENTESES_FECHA statement
       | IF PARENTESES_ABRE expressao PARENTESES_FECHA statement ELSE
       statement
17 iteracao-decl -> WHILE PARENTESES_ABRE expressao PARENTESES_FECHA
       statement
18 retorno-decl -> RETURN PONTO_VIRGULA | RETURN expressao PONTO_VIRGULA
19 expressao -> var ATRIBUICAO expressao | simples-expressao
20 var -> ID | ID COLCHETES_ABRE expressao COLCHETES_FECHA
21 simples-expressao -> soma-expressao relacional soma-expressao | soma-
       expressao
22 relacional: MAIOR_IGUAL | MAIOR | MENOR | MENOR_IGUAL | IGUAL |
       DIFERENTE
23 soma-expressao -> soma-expressao soma termo | SOMA termo | SUB  termo |
       termo
24 soma -> SOMA | SUB
25 termo -> termo mult fator | fator
26 mult -> MULT | DIV
27 fator -> PARENTESES_ABRE expressao PARENTESES_FECHA | var | ativacao |
       INT_NUM
28 ativacao -> ID PARENTESES_ABRE args PARENTESES_FECHA
29 args -> arg-lista | /* entrada vazia */
30 arg-lista -> arg-lista VIRGULA expressao | expressao
```

Algoritmo 2: Gramática Livre de Contexto Aceita pela Lingagem de Programação C-

5.5 Retorno de Erros

O compilador retorna erros léxicos que não se encaixam em nenhum token e erros sintáticos que não são satisfazem a gramática livre de contexto.

Os seguintes erros semânticos também são retornados:

- Atribuição de variável a vetor ou vetor a variável;
- Retorno de função como vetor;
- Criação de variáveis do tipo void;
- Utilização de variáveis não declaradas;
- Declaração de variáveis já declaradas;
- Programa sem função main.

6 O Sistema Operacional

Nesta seção serão definidas as especificações do Sistema Operacional a ser implementado, sendo eles os algoritmos de gerenciamento de processos, memória, e entrada e saída.

O sistemas operacional que será proposto terá partes de sua implementação realizada no compilador e outra parte em processos que irão ser executados pelo sitema computacional.

6.1 Alterações no Compilador

Para implementar o sistema operacional, serão necessários realizar as seguintes alterações no compilador de C- existente.

Buffers

Serão criadas três buffers de propósito geral.
Esses buffers poderão funcionar no modo FIFO (*First in First Out*) ou no modo pilha.
Cada uma desses buffers poderá ser acessada através das seguintes instruções:

- **push[R1]:** Salvará o conteúdo do registrador R1 no final do buffer, se não houverem posições vazias, nada acontece;
- **FIFOpop[*R1*]:** Salvará no registrador R1 o valor contido na primeira posição do buffer, apagará este valor do buffer, se estiver vazio, nada acontece;
- **STACKpop[*R1*]:** Salvará em no registrador R1 o valor contido na última posição do buffer, apagará este valor do buffer, se estiver vazio, nada acontece;
- **write[R1,R2]:** Copiara o conteúdo do registrador R1 para a posição do buffer contida no registrador R2;
- **read[R1,R2]:** Copiara o conteúdo armazenado no buffer, na posição contida no registrador R2, para o registrador R1;
- **top[R1]:** Armazena no registrador R1 o endereço do topo do buffer;
- **down[R1]:** Armazena no registrador R1 o endereço do fundo do buffer.

Essas esses buffers serão utilizadas pelo software para realizar o gerenciamento de processos e passagens de parâmetro.

Interrupções

Será necessário também a criação de um sistema de interrupções para os retirar um processo da CPU após um determinado quantum.

Esse quantum será determinado por um determinado número de instruções executadas pela CPU.

Após esse quantum, o gerenciador de processos deverá ser chamado.

6.2 Processos do Sistema Operacional

Em software, serão criados processos do próprio sistema operacional que cuidarão das seguintes funcionalidades:

- Gerenciamento de processos;
- Gerenciamento de memória;
- Gerenciamento de entradas e saídas.

Gerênciamento de Processos

Os processos serão constituídos dos seguintes componentes:

- Instruções contidas na memória de instruções;
- Uma zona na memória de dados definida por dois ponteiros, um de inicio (PI) e outro de fim (PF);
- Um contador de programa (PC), que defini a instrução a ser executada.

No sistema operacional a ser implementado, os processos poderão estar em três estados, sendo eles pronto, executando e bloqueado, como mostrado na Figura 3.

Os processos no estado *"Pronto"* estão prontos para serem executados, e ficarão em armazenados em uma pilha (seus PCs,PIs e PFs).

O processo no estado *"Executando"* esta sendo executado no momento.

Os processos no estado *"Bloqueado"* estão esperando por uma operação de entrada e saída, estão armazenados em outra pilha (seus PCs,PIs e PFs).

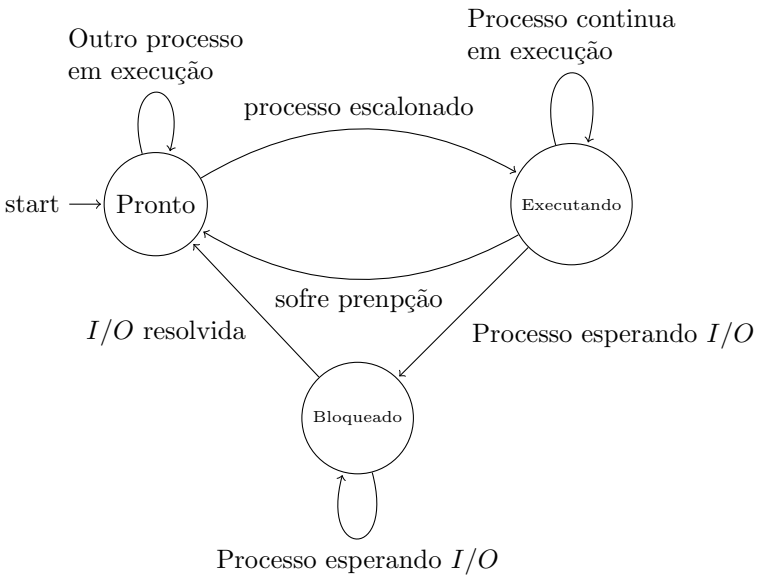

Figura 3: Diagrama de Estados dos Processos

Será implementado um processo que terá a funcionalidade de implementar o escalonamento dos demais processos.

Inicialmente, será implementado o algoritmo Round Robin para realizar o escalonamento, de forma que a cada*"quantum"*, o processo de gerenciamento seja chamado, escolhendo o próximo processo da fila de prontos.

A fila de prontos funcionará no modo FIFO.

Gerênciamneto de Memória

A memória sera gerenciada em um modelo de multiprogramação com partições fixas, isto é, quando o sistema operacional é iniciado, um processo de gerenciamento de memória terá inicio, e dividirá a memória interna em partições, atribuindo uma partição a cada processo que deverá ser executado pelo SO.

Trata-se de partições de tamanho variável, adequando-se ao que é exigido pelo processo, e fixa, um processo sempre utilizará a mesma partição de memória.

Os processos a serem executados são fixos, e devem ser definidos antes do início do SO.

As informações do início e do fim da zona de memória interna de cada processo são armazenadas pelos próprios processo.

Gerênciamento de Entrada e Saída

Haverá um processo cuja função será gerenciar os dispositivos de entrada e saída de maneira mais eficiente.

Sua principal funcionalidade será enviar e receber dados dos dispositivos conectados aos componentes de hardware *Entradas* e *Saídas*, fornecendo aos processos um funcionamento assíncrono.

Os dispositivos ainda poderão ser acessados de maneira síncrona através das instruções de *Output* e *Input*.

Este processo deverá mover um processo que fez uma requisição I/O para fila de processos bloqueados, e quando esse requisição for resolvida, mover novamente o processo para fila de prontos.

7 Etapas Futuras e Possíveis Melhorias

Esse projeto tem como próxima etapa a implementação do que foi definido neste projeto numa placa didática da empresa Altera, que possui o FPGA (Field Programmable Gate Array) Cyclone IV EP4CE115F29C7.

Possíveis melhorias que poderiam ser implementadas são:

- Implementação de métodos mais complexos de gerenciamento de memória, como paginação e segmentação;
- Implementação de outros métodos de escalonar processos, como FIFO e fila de prioridades;
- Implementação de interrupções;
- Implementação de comunicação entre processos;
- Implementação de threads.

Referências

AHO, A., Sethi, R., and Lam, S. (2008). *Compiladores: princípios, técnicas e ferramentas*. LONGMAN DO BRASIL.

Altera (2015). What is an fpga? Disponivel em: <http://www.altera.com/products/fpga.html>. Acesso em: 13 abril 2015 .

CASILLO, L. A. (2015). Tópicos em vhdl. Disponivel em: <http://www2.ufersa.edu.br/portal/view/uploads/setores/145/arquivos/arq/trabalhos/vhdl_epoca.pdf> Acesso em: 23 outubro 2016.

Floyd, T. (2009). *Sistemas Digitais: Fundamentos e Aplicações*. Bookman.

Silberschatz, A. (2008). *Sistemas Operacionais com Java*. Elsevier/Campus.

Tanenbaum, A. and Woodhull, A. (2009). *Sistemas Operacionais: Projetjos e Implementação*. Bookman.