

SSC0511-2020 ORGANIZAÇÃO DE COMPUTADORES DIGITAIS

INSTRUÇÃO PARA SIMULADOR E JOGO EM ASSEMBLY

Grupo:

Ellian Carlos, 11846324; Giovanna Fardini, 10260671;
Thales Damasceno, 11816150; Vinicius Baca, 10788589.

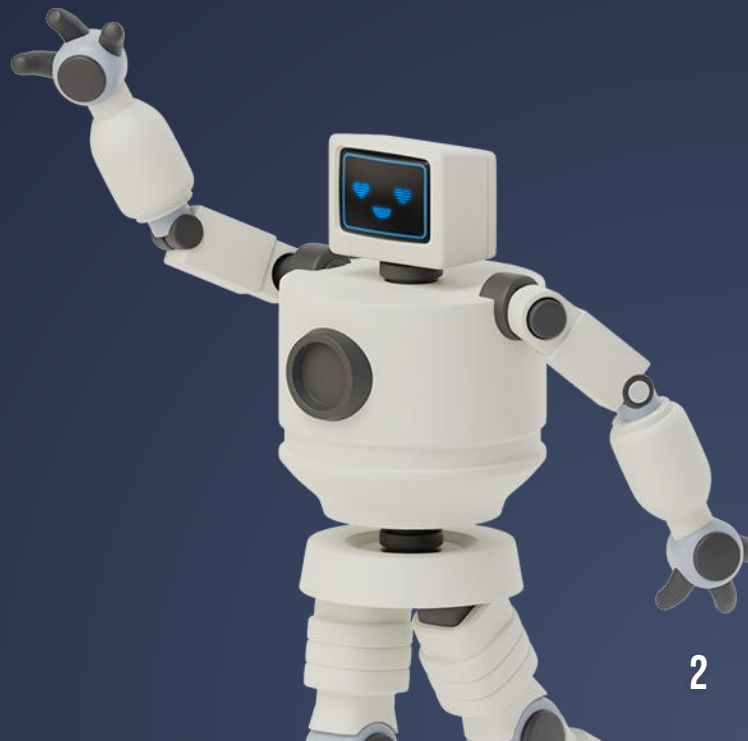


PARTE 1: INSTRUÇÃO PARA O SIMPLE SIMULATOR


Projeto: Criação da instrução XCHG – *Exchange*

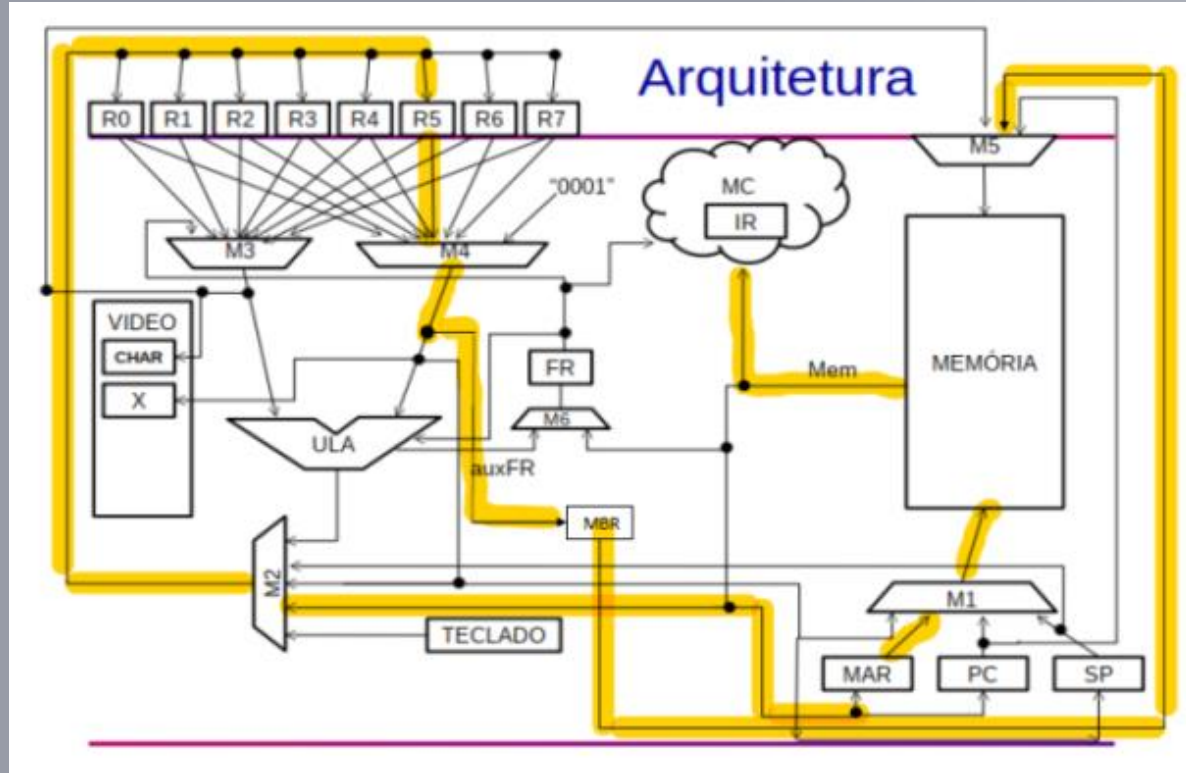
Argumentos a serem passados: registrador e endereço de memória.

Objetivo: trocar os conteúdos entre o registrador e o endereço de memória.



ARQUITETURA

Necessidade de criar um registrador de memória auxiliar – MBR (*memory buffer register*) 



CÓDIGO DO SIMULADOR

Definições no defs.h:

Op code em binário:

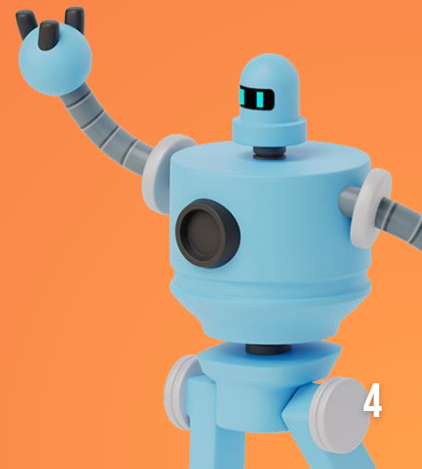
```
/* Data Manipulation Instructions: */  
#define LOAD          "110000"  
#define STORE         "110001"  
#define LOADIMED      "111000"  
#define STOREIMED     "111001"  
#define LOADINDEX     "111100"  
#define STOREINDEX    "111101"  
#define MOV           "110011"  
#define XCHG          "111011"
```

Op code interno

```
#define OUTPUT_CODE    97  
#define XCHG_CODE     98
```

String da função

```
#define XCHG_STR       "XCHG"
```



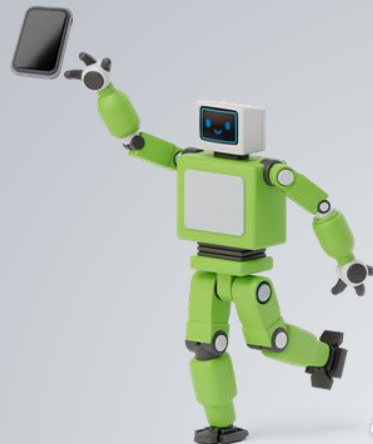
CÓDIGO DO SIMULADOR

Definição do op code:

```
// Opcodes das Instrucoes:
// Data Manipulation:
#define LOAD 48      // "110000"; -- LOAD Rx END -- Rx <- M[END] Format: < inst(6) | Rx(3) | xxxxxxx > + 16bit END
#define STORE 49     // "110001"; -- STORE END Rx -- M[END] <- Rx Format: < inst(6) | Rx(3) | xxxxxxx > + 16bit END
#define LOADIMED 56  // "111000"; -- LOAD Rx Nr (b0=0) -- Rx <- Nr ou Load SP Nr (b0=1) -- SP <- Nr Format: < inst(6) | Rx(3) | xxxxxxx >
#define LOADINDEX 60 // "111100"; -- LOAD Rx Ry -- Rx <- M[Ry] Format: < inst(6) | Rx(3) | Ry(3) | xxxx >
#define STOREINDEX 61 // "111101"; -- STORE Rx Ry -- M[Rx] <- Ry Format: < inst(6) | Rx(3) | Ry(3) | xxxx >
#define MOV 51       // "110011"; -- MOV Rx Ry -- Rx <- Ry Format: < inst(6) | Rx(3) | Ry(3) | xxxx >
#define XCHG 59      // "111011"; -- XCHG Rx END -- M[END] <-> Rx Format: < inst(6) | Rx(3) | xxxxxxx > + 16bit END
```

Ciclo de Decodificação

```
case XCHG:
    // MAR = MEMORY[PC];
    // PC++;
    selM1 = sPC;
    RW = 0;
    LoadMAR = 1;
    selM4 = rx; // move o conteudo do rx
    LoadMBR = 1; // para o MBR
    IncPC = 1;
    // -----
    state=STATE_EXECUTE;
    break;
```



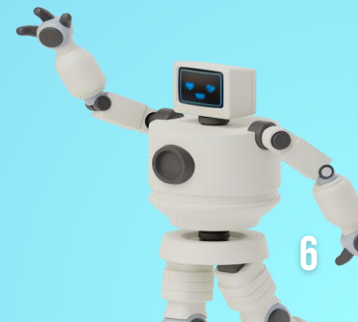
CÓDIGO DO SIMULADOR

- Ciclo de execução 1:

```
case XCHG:
    //reg[rx] = MEMORY[MAR];
    selM1 = sMAR;
    RW = 0;
    selM2 = sDATA_OUT;
    LoadReg[rx] = 1; // manda o conteudo da mem p rx
    // -----
    state=STATE_EXECUTE2;
    break;
```

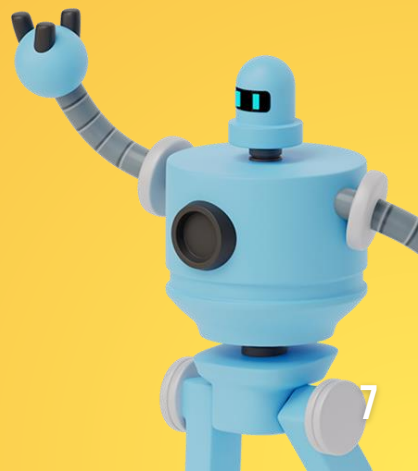
- Ciclo de execução 2:

```
case XCHG:
    //MEMORY[MAR] = reg[rx];
    selM1 = sMAR;
    RW = 1;
    selM5 = sMBR;
    // -----
    state=STATE_FETCH;
    break;
```



CÓDIGO DO MONTADOR

```
case XCHG_CODE :  
    str_tmp1 = parser_GetItem_s();  
    val1 = BuscaRegistrador(str_tmp1);  
    free(str_tmp1);  
    parser_Match(',',');  
    val2 = RecebeEndereco();  
    str_tmp1 = ConverteRegistrador(val1);  
    str_tmp2 = NumPBinString(val2);  
    sprintf(str_msg, "%s%s0000000", XCHG, str_tmp1);  
    parser_Write_Inst(str_msg, end_cnt);  
    end_cnt += 1;  
    sprintf(str_msg, "%s", str_tmp2);  
    parser_Write_Inst(str_msg, end_cnt);  
    end_cnt +=1;  
    free(str_tmp1);  
    free(str_tmp2);  
    break;
```

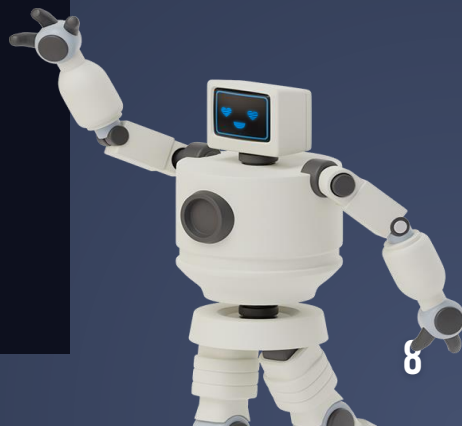


TESTE DA FUNÇÃO

testaCPU.asm

```
3
4 ; 4 Perguntas ao implementar as instrucoes:
5 ; 1) O Que preciso fazer para esta instrucao?
6 ; 2) Onde Comeca: Pegargcc simple_simulator.c -O3
   -march=native -o simulador -Wall -lm -lcurses o que tem
   que fazer e ir voltando ate' chegar em um registrador
   (ie. PC)
7 ; 3) Qual e' a Sequencia de Operacoes: Descrever todos os
   comandos que tem que dar nos ciclos de Dec e Exec
8 ; 4) Ja' terminou??? Cumpru o que tinha que fazer??? O
   PC esta' pronto para a proxima instrucao (cuidado com
   Load, Loadn, Store, Jmp, Call)
9
10 ; Teste do xchg
11 loadn r1, #'B'
12 loadn r0, #1
13 store Dado, r1
14 loadn r2, #'A'
15 outchar r2, r0
16 xchg r2, Dado
17 outchar r2, r0
18
19
20 Fim:
21 halt
22
23
```

```
clang version 7.0.0-3-ubuntu0.18.04.1 (tags/RELEASE_700/
)
cl./mont/montador testaCPU.asm CPURAM.mif
Montador v.0.0
Mensagem (0): Encontrando labels...
Mensagem (20): Label "Fim" em 0xc.
Mensagem (23): Label "Dado" em 0xd.
Mensagem (0): Montando codigo...
Mensagem (0): Inicializando buffer de saida...
Mensagem (24): Descarregando buffer de saida...
Mensagem (24): Concluido.
❏ ./simulador
Rodando...
AB
█
```



JOGO EM LINGUAGEM DE MAQUINA

**Projeto: criação de um jogo baseado no T-Rex
Game do Google Chrome**



JOGO EM LINGUAGEM DE MAQUINA

Estruturação do jogo:

Personagem, obstáculos, cenário, função pulo e pontuação.

Personagem (dino):

Função para mover para pular, apaga e printa personagem e altera posição.

Obstáculo (cacto):

Função que sorteia uma nova posição para o cacto aparecer (no original os cactos aparecem em diferentes tamanhos), função que move o cacto ao longo da tela.



JOGO EM LINGUAGEM DE MAQUINA

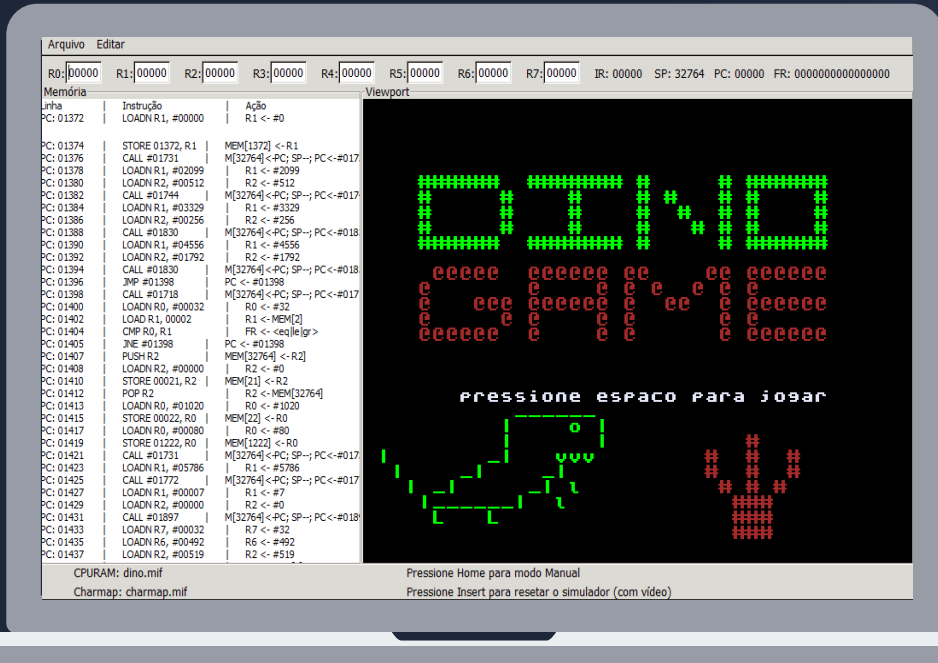
Estruturação do jogo:

Funções auxiliares: gerar números aleatórios para sortear a posição, apaga e printa telas, move cenário, lê o espaço (pulo) e função de delay.

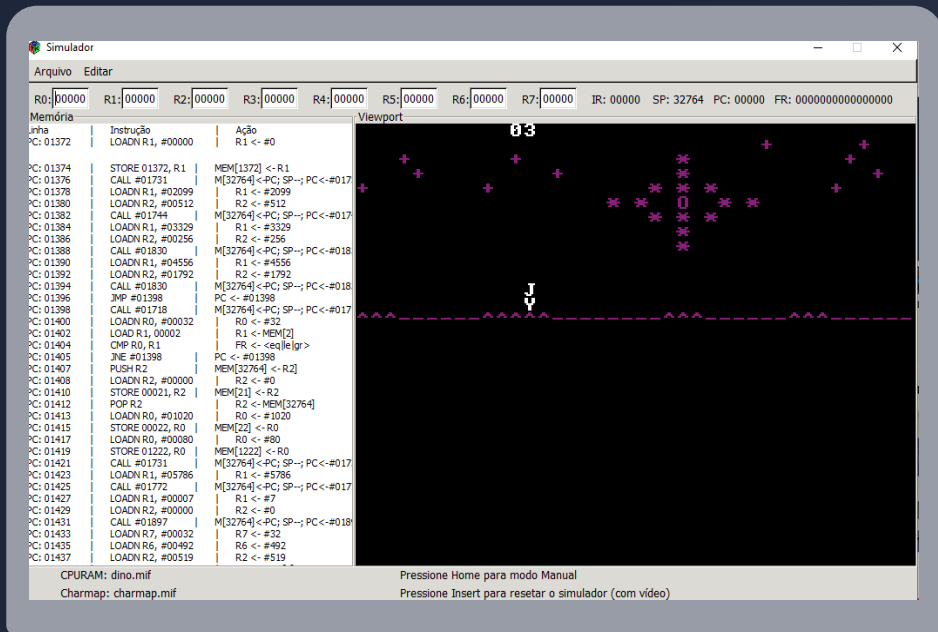
Função de pontuação: atualiza e incrementa os pontos do jogador e ao final do jogo printa a pontuação total.



TELAS DO JOGO



TELAS DO JOGO



TELAS DO JOGO

