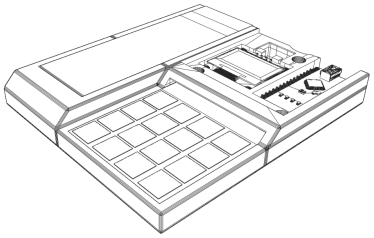
Pocket Ramses Simulator



Manual de Usuário
Marcelo Johann
inf.ufrgs.br
2023

Ramses é um computador didático de 8 bits, criado pelo prof. Raul Weber (1953-2018), descrito no livro (Weber, 2012) e usado como um dos modelos intermediários no ensino de arquitetura e organização de computadores no Instituto de Informática da UFRGS desde a década de 1980 (<https://www.inf.ufrgs.br/arq/wiki/doku.php>).

O simulador de bolso Ramses é uma implementação "física" da arquitetura Ramses com recursos básicos de monitoramento e edição semelhantes aos simuladores de s*oftware* existentes. A implementação usa uma placa do tipo Arduino Nano com microcontrolador ATmega328P, um *display* OLED 0.96" e um módulo sensor capacitivo de 16 teclas TTP229.

Os recursos do microcontrolador permitem implementar todas funções necessárias, mas algumas restrições e compromissos foram feitos para adequar o projeto às limitações de interface, principalmente o *display* com apenas 128*64 pixels (8 linhas de 21 caracteres 5x7) e o teclado com 16 teclas, de forma a ter uma visão compacta e poucas telas. O simulador exibe e recebe todos os dados apenas em hexadecimal, em dois dígitos de 0-F, com uma única exceção: o resultado da última operação da ULA também é exibido como um número decimal com sinal.

A memória é exibida em duas janelas para que se possa ver simultaneamente um trecho do programa e uma área de dados. Cada janela exibe 12 bytes a cada vez, alinhados sempre a um endereço par, e se movem de 8 em 8 bytes com os comandos de avanço ou retrocesso de página (*PageUp* e *PageDown*). As janelas são atualizadas automaticamente no avanço do PC (ponteiro de instrução) ou acesso a operando na memória, exceto em modo imediato, quando o dado já está na área de instruções. Ambas as janelas podem mostrar qualquer parte da memória (de 00H a FFH), mas a segunda pode ser inicializada com 80H para facilitar a visão do início da área de dados, e também exibe os caracteres ascii correspondentes aos valores armazenados.

No modo principal, após a tela de ajuda, o simulador exibe na primeira linha o conteúdo dos três registradores, o estado dos três *flags* N Z C (normal em 0 e invertido em 1), e na segunda linha o conteúdo do PC, uma decodificação da *próxima* instrução (atualmente apontada pelo PC), e o resultado da *última* operação feita pela ULA em decimal com sinal. Nas 6 linhas restantes da tela são exibidas as duas janelas de memória, cada uma com seu cursor, em cor invertida. Para entrada de novos valores para registradores ou posições de memória, são lidos dois dígitos hexadecimais, e usados os números já impressos nos *pads* do módulo TTP229, de 1 a 16, sendo as teclas 1 a 15 mapeadas para os dígitos 1 a F a última tecla 16 usada como dígito de valor 0.

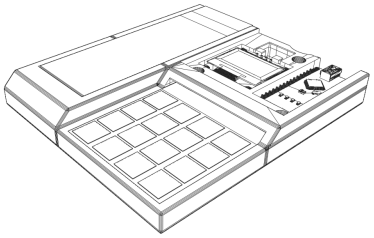
O Simulador contém 8 exemplos pré programados em ROM (Flash), e pode armazenar mais 4 imagens da memória em *slots* de usuário na EEPROM de 1KiB do microcontrolador, os quais podem ser acessados pelas operações de *load* e *save* dentro do modo de edição de memória.

O circuito elétrico contém apenas esses três componentes: Arduino Nano, display e teclado, além de uma gaveta para 4 pilhas AAA de 1.5V e uma chave *on*/*off*. O modelo da caixa para impressão em 3D foi feito em Blender. Os arquivos original e exportados para formato ".STL", junto com a versão atual do simulador em linguagem C da IDE Arduino, e programas de exemplo em assembler, estão disponíveis em: <http://www.inf.ufrgs.br/~johann/ramses2023>

Em anexo estão a lista dos comandos das 16 teclas nos modos principal (simulador) e de edição de memória, mapas de teclas com os comandos em inglês, resumo das instruções da arquitetura Ramses e descrição dos programas armazenados nas imagens de memória.

Referência

[Weber 2012] WEBER, R. F. Fundamentos de arquitetura de computadores. 4. ed. Porto Alegre: Bookman, 2012. 424 p. (Série Livros Didáticos Informática UFRGS, v. 8).



Comandos (modo simulador):

tecla 1 - reseta registradores
tecla 2 - mostra página de visão de memória de dados em 80H
tecla 3 - exibe novamente a tela de ajuda
tecla 4 - entra em modo de edição de memória

tecla 5 - edita valor de RA
tecla 6 - edita valor de RB
tecla 7 - edita valor de RX
tecla 8 - edita valor de PC

tecla 9 - executa uma instrução
tecla 10 - executa instruções até encontrar instrução *Halt* (HLT) ou uma tecla ser pressionada
tecla 11 - executa em modo rápido até encontrar *Halt* (HLT) ou uma tecla ser pressionada
tecla 12 - executa sem atualizar a tela até encontrar *Halt* (HLT) ou uma tecla ser pressionada

tecla 13 - retrocede página 1 de memória (programa) em 2/3 de página (8 *bytes*)
tecla 14 - avança página 1 de memória (programa) em 2/3 de página (8 *bytes*)
tecla 15 - retrocede página 2 de memória (dados) em 2/3 de página (8 *bytes*)
tecla 16 - avança página 2 de memória (dados) em 2/3 de página (8 *bytes*)

Comandos (modo edição de memória):

tecla 1 - carrega uma das 12 imagens de memória em RAM para execução, reseta *regs* e retorna
tecla 2 - armazena o conteúdo atual da RAM em um dos 4 *slots* de usuário
tecla 3 - mostra o endereço da página do projeto com este manual e códigos-fonte
tecla 4 - retorna ao modo principal de simulação

tecla 5 - edita posição de memória apontada pelo cursor 1
tecla 6 - edita posição de memória apontada pelo cursor 2
tecla 7 - reservada
tecla 8 - apaga toda a memória, após pedir confirmação

tecla 9 - retrocede cursor 1 (programa) em uma unidade
tecla 10 - avança cursor 1 (programa) em uma unidade
tecla 11 - retrocede cursor 2 (dados) em uma unidade
tecla 12 - avança cursor 2 (dados) em uma unidade

tecla 13 - retrocede página 1 de memória (programa) em 2/3 de página (8 *bytes)*
tecla 14 - avança página 1 de memória (programa) em 2/3 de página (8 *bytes*)
tecla 15 - retrocede página 2 de memória (dados) em 2/3 de página (8 *bytes*
tecla 16 - avança página 2 de memória (dados) em 2/3 de página (8 *bytes)*

Mapas de teclas:

comandos no Modo Simulador

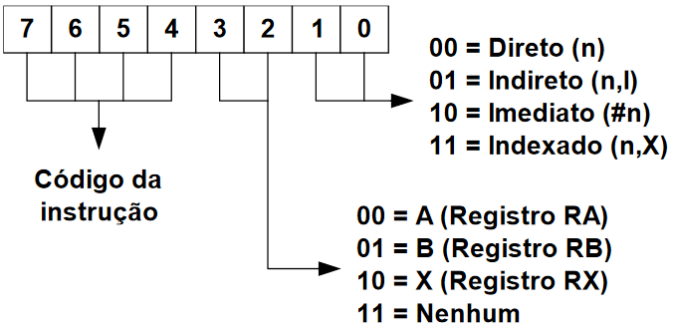
Reset Regs 1	*Pag2=80H* 2	*Help* 3	*Edit Mem* 4
Set RA 5	*Set RB* 6	*Set RX* 7	*Set PC* 8
STEP 9	*RUN* 10	*FAST* 11	*DARK* 12
Pag1Down 13	*Pag1Up* 14	*Pag2Down* 15	*Pag2Up* 16

comandos no Modo Edição de Memória

Load 1	*Save* 2	*WebPage* 3	*Exit* 4
SetMEM 1 5	*SetMEM 2* 6	*void* 7	*Erase* 8
--cursor1 9	*++cursor1* 10	*--cursor2* 11	*++cursor2* 12
Pag1Down 13	*Pag1Up* 14	*Pag2Down* 15	*Pag2Up* 16

Instruções da Arquitetura Ramses

[adaptado de material de apoio desenvolvido por Paulo Krüger Costa]



H	*Código e Operando*	*Endereço (2o byte)*	*Instrução (modo direto)*	*Operação (modo direto)*	*N*	*Z*	*C*
0x	0000 xxxx	–	NOP	Nenhuma operação			
1?	0001 rrmm	dd8	STR rr dd8	MEM(dd8) ← rr			
2?	0010 rrmm	dd8	LDR rr dd8	rr ← MEM(dd8)	t	t	
3?	0011 rrmm	dd8	ADD rr dd8	rr ← rr + MEM(dd8)	t	t	t
4?	0100 rrmm	dd8	OR rr dd8	rr ← rr ∨ MEM(dd8)	t	t	
5?	0101 rrmm	dd8	AND rr dd8	rr ← rr ∧ MEM(dd8)	t	t	
6?	0110 rrxx	–	NOT rr	rr ← ¬ rr	t	t	
7?	0111 rrmm	dd8	SUB rr dd8	rr ← rr - MEM(dd8)	t	t	t [1]
8?	1000 xxmm	dd8	JMP dd8	PC ← dd8 [2]			
9?	1001 xxmm	dd8	JN dd8	if N=1 then PC ← dd8 [2]			
A?	1010 xxmm	dd8	JZ dd8	if Z=1 then PC ← dd8 [2]			
B?	1011 xxmm	dd8	JC dd8	if C=1 then PC ← dd8 [2]			
C?	1100 xxmm	dd8	JSR dd8	MEM(dd8) ← PC PC ← dd8 + 1 [2]			
D?	1101 rrxx	–	NEG rr	rr ← 0 – rr, ou seja, rr ← ¬ rr + 1	t	t	t
E?	1110 rrxx	–	SHR rr	0 → [registrador] → C	t	t	t
Fx	1111 xxxx	–	HLT	Para a execução			

rr - indica um registrador (A, B ou X);
mm - indica um modo de endereçamento (direto, indireto, imediato ou indexado);
x - indica que o *bit* não importa para a execução;
dd8 - representa um endereço de 8 bits;
t - indica que o código de condição é testado pela unidade de controle e ajustado de acordo;
A ausência de t indica que o código de condição não é alterado e mantém seu estado anterior;
(1) o *carry* gerado na instrução SUB é o inverso do *borrow*, ou seja, C=1 indica que não houve *borrow* e C=0 indica que ocorreu *borrow*;
(2) as instruções de desvio no modo imediato são tratadas como NOP e o segundo *byte* é ignorado;

Programas nas imagens de memória

***Slot* 1: load_store** - inicie com este programa que apenas testa instruções de *load* a *store*;

***Slot* 2: add_sub** - testa instruções de soma e subtração com valores simples, para verificar o comportamento de operações aritméticas e principalmente dos *flags* de condição, N, Z e C;

***Slot* 3: Fibonacci** - escreve os primeiros 14 números da série de *Fibonacci*;

***Slot* 4: multiply** - faz a multiplicação simples e não otimizada de pares de números armazenados em um vetor; o tamanho do vetor deve ser informado no endereço 80h, e os números devem estar a partir da posição 82h em diante. O programa multiplica os dois primeiros números, depois os dois seguintes e assim por diante, sempre considerando o segundo como multiplicador. Ou seja, no caso do par [1,200] ele faz 200 somas do valor 1; o resultado de cada multiplicação será armazenado nas posições seguintes ao final do vetor de acordo com o tamanho especificado;

***Slot* 5: looper** - escreve o caractere 'o' (decimal 111 ou hex 6Fh) nas posições de 80h a 8Ah e de 8Bh a 81h, com incrementos de +2 e -2, de forma que se veja esse caractere em um laço cíclico no sentido anti-horário em ASCII na área de dados da tela no simulador;

Slot* 6: Bubbles** - ordena uma lista de números armazenados em um endereço indicado na variável "*databegin*" em 7Ch, com o algoritmo de *BubbleSort*; demora cerca de 13 minutos para ordenar os 32 primeiros números em modo ***fast; A quantidade de números está armazenada na variável "*size*" no endereço 7Dh, e você pode reduzir, ou ordenar até 128 números que já estão na memória a partir do endereço 80h, em poucos segundos, rodando em modo ***dark***;

Slot* 7: Histogram** - faz um histograma de caracteres em um texto; este programa varre uma mensagem de texto de até 110 caracteres armazenada entre os endereços 92h e FFh, conta o número de ocorrências das letras maiúsculas ou minúsculas de A a Z, armazenando em uma tabela entre os endereços 5Ch e 8Fh, onde cada entrada tem o código ascii maiúsculo e o número de ocorrências, e depois ordena a tabela por ordem decrescente de ocorrências, com uma versão simplificada e dedicada (endereços *hardcoded*) do mesmo algoritmo *BubbleSort* anterior; o programa demora cerca de 10 minutos para rodar em modo ***fast, com a entrada dada como exemplo, sendo a maior parte do tempo dedicada ao algoritmo de ordenamento;

Slot* 8: TAplacer** - posicionamento de células lógicas para fabricação de um circuito é um problema *NP-hard*, o que significa que não existe algoritmo que possa encontrar a melhor solução em tempo polinomial. Este programa faz o posicionamento de células (nodos de um grafo) minimizando conexões (as arestas), usando o método estocástico de otimização chamado *Threshold Accepting* (G. Dueck and T. Scheuer, 1990), que é uma simplificação do mais conhecido *Simulated Annealing* (Kirkpatrick, Gelatt and Vecchi, 1983). O algoritmo seleciona duas células ao acaso, troca suas posições, e aceita a troca se o custo for reduzido ou se for pior mas dentro de uma pequena margem (*threshold*), que vai sendo reduzida ao longo do processo. Isso permite que ele busque soluções melhores mas mantendo a capacidade de escapar de mínimos locais. O código faz a minimização do somatório de comprimento de conexões em um grafo contendo exatamente 16 nodos e 32 conexões entre eles, dispostos em um quadrado de tamanho 4x4. O exemplo contido nos dados foi gerado com um método conhecido como PEKO - *Placement Example with Known Optimal* (Chang, Cong, Romesis and Xie, 2003), onde conexões são criadas apenas entre células vizinhas já posicionadas, e depois sua posição embaralhada, e neste exemplo é garantido então que existe solução com custo=32 (número de conexões de tamanho unitário), de um total de quase 21 trilhões de possibilidades (fatorial de 16). Este código consegue encontrar uma solução com custo=39 (hex 27) em cerca de 1 minuto e 15 segundos rodando em modo ***dark, e deve completar em cerca de 10 dias se rodar atualizando a tela em modo ***fast***. As coordenadas das células estão armazenadas nos vetores X e Y nos endereços 90h e A0h, respectivamente, e o custo final pode ser observado nas posições 8Ah e 8Bh;

***Slots* 9 a 12**: **usuário** - espaço para armazenar imagens de usuário na EEPROM;