O Simulador SimuS na Plataforma Raspberry Pi

Gabriel P. Silva
Departamento de Ciência da Computação
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brasil
gabriel@dcc.ufrj.br

José Antonio dos S. Borges
Núcleo de Computação Eletrônica
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brasil
antonio2@nce.ufrj.br

Resumo – *O Raspberry Pi é um nanocomputador de baixo custo, do tamanho de um cartão de crédito, capaz de executar um sistema operacional do tipo Linux, que foi desenvolvido primariamente para o ensino de computação. Entre suas características particulares destacamos a existência de um conjunto de pinos de E/S (GPIO), onde é possível a conexão de sensores e atuadores, de forma a controlar dispositivos externos com diversas funções. O simulador SimuS, desenvolvido para emular a arquitetura do processador hipotético Sapiens, foi portado para a plataforma Raspberry Pi e novas funcionalidades foram adicionadas, de modo a tornar viável ao programador acessar e controlar esses pinos de E/S diretamente a partir de um programa, escrito em linguagem de montagem, executado no simulador. Acreditamos que essas novas facilidades possam enriquecer de forma significativa a experiência de ensino-aprendizagem na disciplina de Arquitetura de Computadores.*

Palavras-Chave: *ensino de arquitetura de computadores; Sapiens; Simulador Simus; Raspberry Pi; Linux; Raspbian.*

I. INTRODUÇÃO

O Raspberry Pi é um nanocomputador de baixo custo e consumo de energia, no tamanho aproximado de um cartão de crédito, capz de executar um sistema operacional do tipo Linux, desenvolvido primariamente para o ensino de computação [13]. Nos últimos anos o Raspberry Pi teve o seu poder de processamento aumentado consideravelmente, permitindo o uso em diversas aplicações no domínio da Internet das Coisas [14, 15]. Neste sentido, o Raspberry Pi apresenta um conjunto de pinos de E/S (GPIO – General Purpose Input Output), onde é possível a conexão de sensores e atuadores, de forma a controlar dispositivos externos com diversas funções. O seu uso possibilita ao educador enriquecer a experiência de ensino-aprendizagem, com a realização de diversos experimentos, desde os mais simples, como acender um LED, até aqueles mais sofisticados, como a montagem de uma estação meteorológica acessível através da internet.

Neste artigo apresentamos as modificações que foram realizadas no simulador SimuS, para o processador hipotético Sapiens, de modo a permitir o acesso aos pinos de GPIO do nanocomputador Raspberry Pi. Além dos tradicionais dispositivos que são emulados no próprio simulador SimuS, como o conjunto de chaves e o visor hexadecimal, pretendemos ampliar o espectro de possibilidades ao alcance do professor, com o acesso aos pinos de GPIO diretamente a partir do código executado no simulador.

Entendemos que essa possibilidade de manipulação de dispositivos concretos, através do contato do aluno com componentes eletrônicos como relés, diodos, leds, displays, sensores de temperatura, pressão, entre outros, é extremamente benéfica e facilitadora da compreensão dos conceitos básicos de construção e funcionamento dos computadores.

Acrescente-se a possibilidade de uso do simulador SimuS [1], onde todo o ciclo de edição, compilação, execução e depuração de um programa em linguagem de montagem é possível, potencializando assim a formação de um sólido conhecimento a respeito do funcionamento dos computadores e sua arquitetura por parte do estudante.

A filosofia principal do simulador SimuS é apresentar uma arquitetura que possa ser explorada em diversos níveis de complexidade, com o emprego de exemplos mais simples, para os cursos iniciais, até exemplos mais complexos, para os cursos mais avançados, sempre com o uso da mesma ferramenta. Pretendemos ainda que essa ferramenta tenha um uso ampliado, tanto para os cursos de Licenciatura e Ciência da Computação, assim como Engenharia de Computação, Engenharia de Software ou Sistemas de Informação.

A arquitetura e o conjunto de instruções usados no simulador Simus foram inicialmente propostos para o processador hipotético Neander-X [3], sendo depois estendidos no processador Sapiens, permitindo uma arquitetura de processador com maiores possibilidades, tais como capacidade de endereçamento de até 64 Kbytes; inclusão de um apontador de pilha; a adição de instruções para chamada e retorno de procedimento e um conjunto ampliado e mais poderoso de instruções.

Apesar dessas mudanças, não houve perda de compatibilidade, em nível de linguagem de montagem, com códigos produzidos para as arquiteturas legadas do Neander [4] e Neander-X que ainda podem ser compilados e executados no simulador SimuS.

O simulador SimuS teve a interface de usuário renovada e apresenta um ambiente integrado de desenvolvimento, onde o aluno pode editar, compilar, depurar e executar código de programas escritos na linguagem de montagem do processador Sapiens, configurando um ambiente integrado de desenvolvimento em que constam:

- Um editor de código com uma ferramenta de apoio ao desenvolvimento;
- Um montador com informações sobre os erros de sintaxe;
- Um simulador completo, onde o funcionamento final do programa pode ser verificado, com acesso a todos os elementos que compõem o estado do processador;

- Emuladores de dispositivos de entrada e saída como "banner" de 16 caracteres e teclado de 12 teclas, entre outros.

O simulador SimuS é distribuído em código aberto[1], o que viabiliza a sua expansão (por outros professores ou por alunos em projeto), possibilitando a exploração de variantes da arquitetura ou adição de novas ferramentas de ensino ou projeto. O código fonte deste simulador, elaborado em Object Pascal (Free Pascal, Delphi), pode ser compilado para diversos processadores e tipos de sistema operacional. As arquiteturas de processadores suportadas por esta linguagem de programação são o x86, x86-64, PowerPC, SPARC e ARM. Os sistemas operacionais suportados são o MacOS, Microsoft Windows, FreeBSD, Linux e suas variantes, incluindo o Raspbian, sistema operacional mais difundido para uso com o Raspberry Pi.

Este artigo está organizado da seguinte maneira: na Seção II apresentamos os trabalhos correlatos, na Seção III a arquitetura do processador didático Sapiens é apresentada. Na seção seguinte discutimos a arquitetura do nanocomputador Raspberry Pi. Na Seção V são apresentados exemplos de programas com a utilização do GPIO do Raspberry Pi e, finalmente, apresentamos as conclusões e trabalhos futuros.

II. Trabalhos Correlatos

Há diversas ferramentas de simulação para processadores e microcontroladores comerciais, dentre os quais destacamos: 8051, ATmega, PIC e variantes de processadores do tipo RISC, como o ARM. Mas são quase sempre sistemas completos, contendo muitas ferramentas integradas, destinados ao uso profissional, sendo necessário em muitos casos o uso de licenças pagas e são relativamente complexos, inviabilizando o uso amplo no ensino de graduação em muitos cursos no Brasil.

Na área dos simuladores didáticos existem vários sistemas desenvolvidos no exterior, muitos deles descritos em [5]. Como exemplos de simuladores didáticos de processadores comerciais podemos citar o GNUSim8085 [6], MARS [11] e o WinMIPS64 [7].

O GNUSim8085 (desenvolvido inicialmente para plataformas Unix, mas com versões agora também para o sistema operacional Windows) simula o microprocessador 8085 da Intel, que tem sido utilizado amplamente no ensino de arquitetura de computadores, devido à sua simplicidade e por não possuir pipeline. O mesmo apresenta os valores dos registradores no decorrer da execução das instruções, além de exibir o conteúdo presente na memória do processador. O GNUsim08085 conta com um editor e montador integrados, mas não possui nenhum tipo de dispositivo de E/S.

O WinMIPS64 é um simulador para o processador MIPS64, com uma única versão disponível no sistema operacional Windows, com diversas funcionalidades, tais como a visualização das instruções passando pelos diversos estágios do pipeline, execução ciclo a ciclo, além da possibilidade do aluno examinar e alterar o conteúdo de todos os registradores do processador, sejam inteiros ou de ponto flutuante. Este simulador faz ainda a emulação de um terminal muito simples, cujo acesso é feito através de registradores de controle, com endereços mapeados em memória.

O simulador MARS é um simulador sofisticado para o processador MIPS de 32 bits escrito na linguagem Java, o que permite o seu uso tanto em ambientes Windows como Linux. Conta com um editor e montador integrados, acesso e modificação do conteúdo de todos os registradores. Não possui visualização da execução das instruções no pipeline, mas apresenta o conceito de chamadas de sistema, com o uso da instrução *syscall,* para acesso a dispositivos virtuais de E/S pelo programa simulado.

Há alguns exemplos de ferramentas didáticas desenvolvidas para processadores hipotéticos no Brasil, entre as quais podemos citar o simulador R2DSim [8], uma ferramenta didática para simulação de uma arquitetura RISC de 16 bits com pipeline de quatro estágios, cujo banco de registradores possui tamanho e largura configurável. No conjunto de ferramentas apresentadas estão um montador e um simulador integrados.

O SIMAEAC [9] - Simulador Acadêmico para Ensino de Arquitetura de Computadores—implementa um subconjunto da arquitetura do processador 8085, mas com uma memória de apenas 256 bytes, com um montador e simulador integrados.

Temos ainda o SEAC [10], um simulador online para uma arquitetura hipotética com pipeline de quatro estágios, com foco nos passos de microprogramação necessários para executar cada instrução.

Outro simulador interessante é o W-Neander, produzido como software companheiro do livro Fundamentos de Arquitetura de Computadores [4], mas que carece de montador e de editor integrados. Dada a sua simplicidade, este processador atraiu nossa atenção, com um potencial de uso em cursos iniciais de graduação, apresentando grande facilidade de aprendizado. No mesmo livro encontramos outras arquiteturas mais sofisticadas, porém quando aplicadas em sequência num curso haveria necessidade de migração para outros ambientes de desenvolvimento, consumindo um tempo precioso em sala de aula.

A simplicidade da arquitetura do Neander é um dos seus atrativos, porém também a sua maior debilidade, pois o conjunto original de instruções apresenta sérias limitações que dificultam a criação de programas pouco mais do que triviais.

Para minimizar esses problemas, apresentamos em 2006 o simulador Neanderwin para uma versão estendida dessa arquitetura, que chamamos de Neander-X, ambos descritos em detalhes em [2]. Nossa ideia foi unir as estratégias de integração de ferramentas, encontradas nas ferramentas profissionais com uma arquitetura simples como a do Neander, que seria expandida para diminuir algumas limitações e ampliar o seu potencial didático.

Consolidamos e avançamos nesses conceitos, apresentamos em 2016 tanto o simulador SimuS quanto o processador Sapiens, que são uma evolução do simulador Neanderwin e da arquitetura Neander-X, mas sempre guardando compatibilidade em nível de linguagem de montagem com os programas já

1Disponível em https://sourceforge.net/projects/simus/

desenvolvidos para os processadores originais. No SimuS podemos encontrar o conceito de dispositivos virtuais de E/S e o uso de funções especiais como temporização e geração de números aleatórios, que são acessíveis ao programa simulado com o uso de instruções especiais de TRAP, emulando o que seria um tipo de "chamada ao sistema", cujos detalhes são apresentados na seção seguinte.

Quanto à proposta apresentada neste artigo, de extensão do simulador SimuS para uso no Raspberry Pi, não encontramos nenhuma ferramenta didática, com uso de linguagem de montagem, que oferecesse esta facilidade para acesso aos pinos de GPIO e a dispositivos reais de E/S a eles conectados.

III. O Processador Sapiens

A. *Características Principais*

A arquitetura do processador Sapiens foi apresentada em detalhes em [1], sendo que aqui faremos um resumo das suas características principais. A sua arquitetura é uma evolução do processador Neander-X, ideal para introduzir o funcionamento de um processador e de sua programação em linguagem de montagem. O diagrama em blocos do processador Sapiens é mostrado na Figura 1, onde destacamos as seguintes características:

- Acumulador de 8 bits;
- Apontador de instruções (PC) de 16 bits;
- Apontador de pilha (SP) de 16 bits;
- Códigos de condição (*flag*) N, Z e C;
- Modos direto, imediato e indireto de endereçamento aos operandos em memória;
- Operações de entrada e saída de dados para dispositivos E/S, em espaço de endereçamento separado da memória;
- Instrução de TRAP para simular a chamada de instruções no nível de sistema operacional, com o intuito de simular a realização de operações mais elaboradas de E/S;
- Novas instruções de movimentação de pilha, deslocamento do registrador, soma e subtração com vai-um/vem, entre outras.

O conjunto original de instruções foi expandido para permitir uma maior capacidade de processamento. Um código escrito em linguagem de montagem para os processadores Neander ou Neander-X é totalmente compatível com o Sapiens e executado sem problemas no SimuS.

Como destaque para as novas instruções introduzidas na arquitetura:

- Adição e subtração com carry, (ADC e SBC);
- "Ou exclusivo", deslocamento para a direita e esquerda do acumulador (XOR, SHL, SHR e SRA);
- Desvio condicional (JP, JC, JNC);
- Chamada e retorno de procedimento (JSR e RET);
- Manipulação da pilha (PUSH e POP);
- Carga e armazenamento do apontador de pilha em memória (LDS e STS).

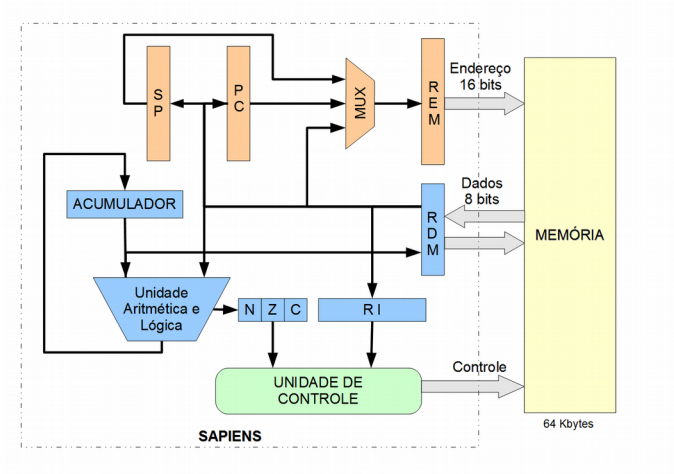

Figura 1: Processador Sapiens

B. *Simulando operações de E/S complexas usando a instrução TRAP*

A necessidade de ensinar técnicas de entrada e saída num momento preliminar do ensino, sempre foi uma das mais problemáticas observadas em nossa experiência didática. Mesmo em qualquer problema menos trivial, havia a necessidade de realizar leituras e escritas de dados, porém isso exigia do aluno conhecimentos específicos ainda não aprendidos nas primeiras aulas (laços, testes, *polling, flags*, etc.). Desta forma se tornava quase mandatório que os primeiros programas fossem centrados em resolver problemas abstratos usando valores pré-carregados na memória.

Isso poderia ser resolvido, por exemplo, com a inclusão de um pequeno *kernel* pré-carregado em linguagem de máquina na memória do simulador SimuS, que fornecesse diversas funcionalidades (leitura e escrita simplificada, formatação de dados, funções específicas dos dispositivos, etc.). Porém, isso acabaria por tornar ainda mais complexo o processo de ensino, na medida em que professores e alunos ficariam tentados a explorar este código, desfocando o objetivo principal que é o de ensinar arquitetura de computadores. Além disso, a presença deste código pré-definido tornaria ainda mais limitado o uso do recurso já tão escasso como a memória. Optamos então por criar uma abstração, similar àquela apresentada em processadores reais para chamar funções de sistema operacional: a chamada ao sistema operacional, realizada em nosso caso com o uso da operação TRAP, sendo que na verdade essas funcionalidades são implementadas diretamente no simulador.

O simulador SimuS interpreta a instrução TRAP como um pedido para que uma determinada operação de E/S seja realizada. O número do *trap,* ou seja, a função que está sendo solicitada é passada no acumulador. A instrução TRAP tem um

operando adicional que é o endereço de memória onde os parâmetros adicionais necessários são passados para o módulo do simulador responsável pela realização da operação. Isso ajuda a contornar o problema da carência de registradores do processador Sapiens. O SimuS na versão suporta inicialmente as seguintes operações:

- #1 – Leitura de um caractere da console. O código ASCII correspondente é colocado no acumulador.
- #2 – Escrita de um caractere que está no endereço definido pelo operando da instrução TRAP na console. O caractere escrito é retornado no acumulador.
- #3 – Leitura de uma linha inteira da console para o endereço definido pelo operando da instrução TRAP. O tamanho da cadeia de caracteres lida é retornado no acumulador, com no máximo 255 caracteres sem o fim de linha ou retorno de carro.
- #4 – Escrita de uma linha inteira na console. O operando da instrução TRAP contém o endereço para a cadeia de caracteres que deve ser terminada pelo caractere NULL (0x00). O tamanho máximo da cadeia é de 255 caracteres. O número total de caracteres escritos é retornado no acumulador.
- #5 – Chama uma rotina de temporização. O operando da instrução TRAP contém o endereço da variável inteira com o tempo a ser esperado em milissegundos.
- #6 – Chama uma rotina para tocar um tom. A frequência e a duração do tom estão em duas variáveis inteiras de 16 bits no endereço definido pelo operando da instrução TRAP.
- #7 – Chama uma rotina para retornar um número pseudoaleatório entre 0 e 99 no acumulador.

  #8 – O operando da instrução TRAP tem o endereço com a variável de 16 bits com a semente inicial da rotina de números aleatórios.

IV. A Arquitetura do Raspberry Pi

A. Hardware

O hardware Raspberry Pi evoluiu através de várias versões que apresentam variações na capacidade de memória e no suporte a dispositivos e periféricos. Em nosso estudo utilizamos a versão 3 Modelo B do Raspberry Pi, que conta com a seguinte configuração:

- Processador ARM Cortex-A53 de 64 bits com 4 núcleos e cache compartilhado L2 de 512, do fabricante Broadcom, modelo BCM2837;
- Memória RAM de 1 Gbyte;
- Interface para Rede sem Fio e Bluetooth 4.1 (BLE) integrados na placa;
- 4 interfaces USB 2.0;
- Interface GPIO de 40 pinos;
- Saída de áudio externa e porta de vídeo composto;
- Saída HDMI padrão normal;
- Porta CSI para conexão de câmera Raspberry Pi;
- Porta DSI para conexão de tela de toque Raspberry Pi;
- Porta micro SD para carregar o sistema operacional e armazenamento persistente;
- Porta micro USB para fonte de alimentação.

O Raspberry Pi 3, com um processador quad-core Cortex-A53, é descrito como tendo 10 vezes o desempenho de um Raspberry Pi 1, assim como é aproximadamente 80% mais rápido do que o Raspberry Pi 2 em tarefas paralelizadas. [19]

B. Sistema Operacional

Diversos sistemas operacionais estão disponíveis para utilização com o Raspberry Pi. Entre eles destacamos o Raspbian, uma variante do Debian, que é a versão suportada oficialmente pela Fundação Raspberry; temos ainda o Ubuntu Mate e Windows 10 IoT Core. Há outros sistemas operacionais menos conhecidos, como aqueles com a função de Central de Mídia, como o OSMC (Open Source Media Centre) e LibreELEC (Kobi Media Centre) e ainda o distinto RISC OS, desenvolvido pela mesma equipe que desenvolveu o processador ARM, e que não é baseado em nenhuma distribuição Unix e roda com apenas um único usuário.

O sistema operacional utilizado para o durante os nossos testes foi o Raspbian. O Raspbian é uma versão de Linux feita especialmente para o Raspberry Pi. Ele vem com diversos pacotes como ambiente gráfico de trabalho PIXEL, a suíte de escritório LibreOffice, o navegador Chromium, cliente de correio eletrônico e programas para o ensino de programação para crianças e adultos, como Heck e jogos como o Minecraft. É com certeza a escolha mais difundida de sistema operacional para o Raspberry Pi.

Adicionalmente, foi instalada uma versão do compilador Free Pascal e do ambiente de desenvolvimento Lazarus [17], que foram utilizados para o transporte do simulador SimuS para o Raspberry Pi.

C. Pinos do GPIO

Uma característica importante do Raspberry Pi é o conector de pinos GPIO (entrada/saída de uso geral) que existe na sua placa. Esses pinos são uma interface física entre o Pi e o mundo exterior.

Os modelos originais A e B do Raspberry Pi possuíam um conector com 26 pinos, o que mais tarde foi expandido para 40 pinos nos modelos A+ e B+. Do total de pinos do conector, dezessete (originais) ou vinte e oito (A+ e B+) são pinos que podem ser configurados como entrada e saída digital, e os demais são pinos de alimentação ou terra. Entre os pinos digitais, existem alguns que podem ser alternativamente configurados para outras funções: dois para saídas de modulação por largura de pulso (PWM); dois para comunicação serial assíncrona; dois para a interface I2C; e finalmente cinco pinos para a interface SPI (sendo um destes pinos para seleção de um escravo). Uma limitação do Raspberry Pi é não possuir entradas que realizem conversão analógico digital.

Antes da utilização desses pinos é necessário informar se os mesmos são entradas, saídas ou utilizados para algum protocolo de comunicação como I2C ou SPI. Isso pode ser feito de diversas maneiras, de acordo com o ambiente ou a linguagem de programação utilizada. Na Seção V mostraremos como esses pinos podem ser programados no SimuS.

Os pinos de saída do conector GPIO podem gerar sinais com 3,3 volts com até 16 mA de capacidade individual de corrente, não excedendo 50 mA no conjunto [16], com configurações variadas, dependendo do modelo. Notem que esses pinos não tem proteção contra curto-circuito, necessitando de um cuidado adicional na sua manipulação para não serem danificados. Na Figura 2 apresentamos a pinagem do GPIO no Raspberry Pi 3B.

Figura 2: Pinagem GPIO Raspberry Pi

Como podemos observar há dois tipos de numeração: a numeração física sequencial, iniciando no pino 1 até o pino 40, e a numeração do controlador GPIO, que se inicia em 0 e vai até 27. Isso é causador, infelizmente, de alguma confusão na utilização desses pinos, principalmente para os usuários menos experientes.

Esses pinos podem ser utilizados para interagir de diversos modos com o mundo real. As entradas não precisam vir apenas de uma chave ou interruptor físico; elas podem ser ligadas a um sensor ou conectadas a um sinal vindo de outro computador ou dispositivo. As saídas também podem ser utilizadas de diversas maneiras, com ligar um LED, acionar um motor ou enviar dados para outro dispositivo.

A conectividade e o controle de dispositivos físicos através da internet é uma funcionalidade poderosa e de muito interessante para capturar o interesse e a atenção dos estudantes. Se o Raspberry Pi estiver conectado em uma rede, ele pode controlar dispositivos que estão conectados a ele a partir de qualquer lugar no mundo, em um conceito conhecido com Internet das Coisas.

V. O SIMULADOR SIMUS NO RASPBIAN

O processo de transporte do SimuS para sistema operacional Raspbian se iniciou com a instalação do compilador Free Pascal e do ambiente de desenvolvimento Lazarus, de modo que possam ser gerados códigos-objeto compatíveis com a arquitetura ARM, de 64 bits, que é o processador utilizado no Raspberry Pi. Embora no site oficial do Lazarus [18] encontremos apenas versões disponíveis para a arquitetura Intel, é possível baixar versões compiladas para o processador ARM no endereço <https://www.getlazarus.org/setup/>. Para tanto deve ser baixado um "script" de instalação *setup.sh* e em seguida executados os seguintes comandos:

$ chmod +x setup.sh

$ ./setup.sh

Durante o processo de instalação são verificadas se as dependências com os pacotes necessários ao correto funcionamento do Lazarus e Free Pascal estão satisfeitas. Caso seja detectada algum falha de dependência, deve ser providenciada a instalação dos pacotes.

De posse dessas ferramentas configuradas no sistema operacional, a instalação do simulador SimuS no sistema operacional Raspbian foi direta, bastando apenas baixar o código fonte e compilar o mesmo no novo ambiente. O simulador foi executado sem nenhum problema e com desempenho similar ao encontrado em ambientes do tipo Linux ou Windows com processadores x86 ou x86_64.

A. Acesso ao Pinos do GPIO no Raspbian

Um dos objetivos no desenvolvimento do Raspberry Pi foi facilitar o acesso sem esforço a dispositivos externos como sensores e atuadores. Os pinos de GPIO do Raspberry na realidade não são todos iguais, possuindo diversos tipos de conexão:

- Pinos verdadeiros de GPIO que podem ser usados para ativar e desativar LEDs, leitura de chaves, etc;
- Pinos seriais de RX e TX para comunicação com periféricos seriais.
- Pinos de interface I2C (SCL e SDA), que permitem conectar módulos de hardware com apenas dois pinos de controle;
- Pinos de Interface SPI (MOSI, MISO e SCKL), um conceito semelhante ao I2C, mas com um padrão diferente;

Além disso, alguns dos pinos podem ser usados para PWM (modulação de largura de pulso) para controle de potência (motores, intensidade de LEDs, etc.) e outro tipo de geração de pulsos para controle de servo motores denominado PPM (Modulação de Posição de Pulso).

Todos os pinos possuem níveis de lógica de 3,3V e não são seguros para 5V, portanto os níveis de saída são entre 0 e 3,3V e as entradas não devem ser superiores a 3,3V. Se você deseja conectar uma saída de 5V a uma entrada do Raspberry Pi deve usar um divisor resistivo ou circuito conversor de nível lógico.

Os pinos I2C e SPI são compartilhados com pinos de GPIO convencionais e estão desabilitados por padrão no Raspbian, já os pinos de TX/RX também são compartilhados, mas estão habilitados por padrão. Alterações nestas configurações podem ser feitas com uso do utilitário "raspi-config".

De uma foma geral, esse pinos podem ser acessados principalmente dos seguintes modos a partir de um programa escrito em Object Pascal:

1. Através de registradores de E/S mapeados em memória:

   Esta forma pode ser utilizada diretamente, com o uso de rotinas de mapeamento (mmap) da memória física

no espaço de endereçamento virtual do processo, ou indiretamente através de bibliotecas como ***wiringPi*** [12] e PXL [18].

2. Através do acesso ao sistema de arquivos dentro do diretório /sys/class/gpio:

   Esta forma pode ser utilizada com chamadas embutidas ao shell, efetuadas com uso da rotina fpsystem() da unit Unix; ou com o uso das rotinas fpopen(), fpread() e fpwrite() da unit BaseUnix.

Em qualquer dos casos há a necessidade de os programas serem executados em uma conta com privilégios para ser possível o acesso ao espaço de endereçamento de E/S do sistema operacional. Para evitar isso, alguns artifícios podem ser utilizados, com uso do programa executável com bit "setuid" ligado, tendo o "root" como proprietário do arquivo. Nas versões mais recentes do Raspbian, o usuário "pi" pertence a um grupo especial "gpio", que tem privilégios para acesso aos pinos do GPIO.

B. Acesso do SimuS ao conector GPIO do Raspberry Pi

Ao considerar as possibilidades de acesso aos pinos do conector GPIO, gostaríamos de oferecer ao programador em linguagem de montagem uma forma fácil e rápida de realizar este acesso. Levamos porém em conta o fato de que o "hardware" do Raspberry Pi tem sofrido modificações e evoluções ao longo do tempo e que, certamente, novas mudanças surgirão e, portanto, procuramos tornar este acesso o menos dependente possível das especificações do "hardware".

A solução encontrada foi utilizar o mecanismo já existente no SimuS disponível com a instrução de TRAP, realizando uma "chamada de sistema", com passagem de parâmetros relativos a operação de E/S desejada na memória do programa em execução.

A partir deste ponto a execução é transferida para simulador, que realiza as operações de E/S, reais neste caso, requisitadas. Como o desempenho não é uma questão fundamental, sendo esperado que o acesso aos pinos GPIO seja feito apenas para executar operações simples de E/S (por exemplo, piscar um LED, ler o valor de um pino), optamos pela maneira mais fácil e portátil de implementação, e o mais possível imune às eventuais mudanças que possam ocorrer no "hardware" para as próximas versões do Raspberry Pi.

Como observação adicional, deixamos claro que essas funções adicionais só estão disponíveis quando o SimuS é executado no Raspberry Pi, não tendo efeito quando utilizado em outros sistemas, seja com o sistema operacional Linux ou Windows. As rotinas de TRAP retornam silenciosamente nestes casos e nenhuma mensagem de erro é passada para o usuário.

As necessidades das operações de E/S envolvem basicamente a programação dos pinos como entrada, saída ou PWM; a escrita nos pinos de sinais digitais ou PWM; e a leitura de sinais digitais dos pinos. O suporte a interrupções não foi implementado. O acesso ao hardware encapsulado por "shell calls", resolve satisfatoriamente os primeiros casos, mas não permite a leitura de valores digitais desses pinos. Para evitar o uso de operações de E/S mapeadas em memória, escolhemos utilizar a biblioteca ***wiringPi***, com um "wrapper" , de nome ***hwiringpi,*** para uso a partir dos programas escritos em Object Pascal.

Retomando a questão da numeração dos pinos, a biblioteca ***wiringPi*** oferece ainda um terceiro método de numeração "virtual" para acessos aos pinos do GPIO, como tentativa de estabelecer um padrão independente das mudanças de "hardware" no Raspberry Pi. Apesar disso, adotamos a numeração dos pinos conforme os mesmos são vistos pelo controlador de GPIO, por acreditarmos que seja a mais simples e com menor possibilidade de indução a erros por parte dos usuários.

Mostramos um exemplo básico de programação em Internet das Coisas, que é um programa em linguagem C co uso da biblioteca ***wiringPi***, para fazer um LED piscar:

```
begin
wiringPiSetup();
pinMode(p22, OUTPUT);
While true do begin
    digitalWrite(p22, HIGH);   // Liga o  LED  em P22
    delay(500);
    digitalWrite(p22, LOW);    // Desliga o LED
    delay(500);
    end;
end.
```

Usando o esquema definido acima, foi possível implementar com relativa facilidade no SimuS as seguintes rotinas de TRAP para acesso aos pinos de GPIO do Raspberry Pi:

#101 – rotina para configurar certo pino como entrada, saída comum, ou saída com modulação por largura de pulso (PWM), ou libera a alocação desta porta. Os parâmetros, que são passados no endereço de memória definido pelo operando da instrução TRAP, são o número do PINO (1 byte) e o MODO (1 byte), que poder ter os seguintes valores: 0 – ENTRADA, 1 – SAÍDA, 3 – PWM, 4 – libera o uso desta porta.

#102 – escreve o valor lógico alto (1) ou baixo (0) no pino que deve ter sido previamente configurado como saída. Os parâmetros, que são passados no endereço de memória definido pelo operando da instrução TRAP, são o número do PINO (1 byte) e VALOR (1 byte), onde qualquer valor diferente de 0 é tratado como NÍVEL ALTO, e apenas 0 é NÍVEL BAIXO.

#103 – retorna no acumulador o valor lido em um dado pino, podendo ser nível lógico baixo (0) ou alto (1). Os parâmetros, que são passados no endereço de memória definido pelo operando da instrução TRAP, são o número do PINO (1 byte), apenas.

#104 –Rotina para configurar o modo de resistência "pull-up" ou "pull-down" em um dado pino, que deve ser configurado necessariamente como uma entrada. O BCM2835 do Raspberry Pi tem resistores internos tanto de "pull-up"

como de “pull-down”. Os parâmetros, que são passados no endereço de memória definido pelo operando da instrução TRAP, são o número do PINO (1 byte) e PUD (1 byte), que poder ter os seguintes valores: 0 – PUD_OFF, (sem nenhum resistor), 1 – PUD_DOWN (resistor de “pull-down” para terra) ou 2 – PUD_UP (resistor de “pull-up” para 3,3V). Os resistores internos de “pull up/down” têm um valor de aproximadamente 50 KΩ no Raspberry Pi.

#105 – configura o “duty cycle” do registrador PWM do pino correspondente que deve ter sido previamente configurado como saída. Os parâmetros, que são passados no endereço de memória definido pelo operando da instrução TRAP, são o número do PINO (1 byte) e VALOR (2 bytes) do “duty cycle”. No Raspberry Pi o modo PWM possui uma frequência de 600 kHz e valores entre 0 e 1023 para o "duty cycle" da forma de onda.

VI. Exemplos de Programas

A seguir apresentamos alguns exemplos de código em linguagem de montagem do Sapiens, começando com uma versão do tradicional programa “blink” para piscar um LED ligado ao pino 13 do GPIO.

```
    ORG 0
;-----------------------------------
; PROGRAMA PRINCIPAL
;-----------------------------------
; Prepara os parâmetros a serem passados
; para a rotina de TRAP no endereço de
; memória correspondente a PARAM_TRAP
;
; O primeiro parâmetro é o número
; do pino = 13
    LDA    #13
    STA    PARAM_TRAP
;
; O segundo é o modo do pino
; SAIDA = 1
    LDA    #1
    STA    PARAM_TRAP +1
;
; Coloca no acumulador o número
; que corresponde ao TRAP que
; vai ser chamado. MODO = 101
    LDA    #101
;
; Faz a chamada da rotina de TRAP.
    TRAP   PARAM_TRAP
LOOP:
;
; O primeiro parâmetro é o número
; do pino = 13, mantido inalterado.
; O segundo parâmetro é o valor a
; ser colocado na saída, ALTO = 1
    LDA    #1
    STA    PARAM_TRAP+1
;
; Coloca no acumulador o número
; que corresponde ao TRAP que
; vai ser chamado. ESCRITA = 102
    LDA    #102
;
; Faz a chamada da rotina de TRAP
    TRAP   PARAM_TRAP
;
; Coloca no acumulador o número
; que corresponde ao TRAP que
; vai ser chamado. DELAY = 5
; Espera igual a 1000 ms, defininda
; na variável T1000
    LDA    #5
    TRAP   T1000
;
; Altera o segundo parâmetro, o valor
; a ser colocado na saída, BAIXO = 0
    LDA    #0
    STA    PARAM_TRAP +1
;
; Coloca no acumulador o número
; que corresponde ao TRAP que
; vai ser chamado. ESCRITA = 102
    LDA    #102
;
; Faz a chamada da rotina de TRAP
    TRAP   PARAM_TRAP
;
; Espera mais 1000 ms
    LDA    #5
    TRAP   T1000
;
; Fica eternamente em LOOP
    JMP    LOOP
;-----------------------------------
; Variáveis auxiliares.
;-----------------------------------
; Tempo utilizado da rotina “delay”
T1000:    DW 1000
;
; Parâmetros da rotina de TRAP
; declarados sem valores iniciais
PARAM_TRAP: DS    3
;
; Fim do programa
    END    0
```

A seguir mostramos um exemplo de um programa para fazer o “fading” de um LED ligado ao pino 18 do GPIO.

```
    ORG  0
;-----------------------------------
; PROGRAMA PRINCIPAL
;-----------------------------------
; Prepara os parâmetros a serem passados
; para a rotina de TRAP no endereço de
; memória correspondente a PARAM_TRAP
;
; O primeiro parâmetro é o número
; do pino = 18
    LDA   #PWM_PIN
    STA   PARAM_TRAP
;
; O segundo é o modo do pino
; PWM = 3
    LDA   #PWM_MODE
    STA   PARAM_TRAP +1
;
; Coloca no acumulador o número
; que corresponde ao TRAP que
; vai ser chamado. MODO = 101
```

```
    LDA   #MODO_TRAP
    TRAP  PARAM_TRAP
;
; Loop eterno fade in / fade out
LOOP:
    JSR   FADEIN
    JSR   FADEOUT
    JMP   LOOP
;-----------------------------------
; Rotina para fazer FADEIN
;-----------------------------------
FADEIN:
;
; Valor inicial duty cycle PWM = 0
    LDA   #0
    STA   PWM_HIGH
    STA   PWM_LOW
LOOPFIN:
;
; Chama rotina que muda o valor do
; duty cycle do PWM (0 - 1023)
    JSR   TRAP_PWM
;
; Incrementa o valor do duty cycle
; de 16 unidades
    LDA   PWM_LOW
    ADD   #16
    STA   PWM_LOW
;
; Ajusta a parte alta se houve "vai-um"
    LDA   #0
    ADC   PWM_HIGH
    STA   PWM_HIGH
;
; Termina o loop se o valor do duty cycle
; do PWM chegou a 1024
    SUB   #04H
    JNZ   LOOPFIN
;
; Retorna da subrotina
    RET
;-----------------------------------
; Rotina para fazer FADEOUT
;-----------------------------------
FADEOUT:
;
; Valor inicial duty cycle PWM = 1008
    LDA   #03
    STA   PWM_HIGH
    LDA   #0F0H
    STA   PWM_LOW
;
LOOPFOUT:
;
; Chama rotina que muda o valor do
; duty cycle do PWM (0 - 1023)
    JSR   TRAP_PWM
;
; Diminui o valor do duty cycle
; de 16 unidades
    LDA   PWM_LOW
    SUB   #16
    STA   PWM_LOW
    LDA   PWM_HIGH
;
; Ajusta a parte alta se houve "vem-um"
    SBC   #0
    STA   PWM_HIGH
;
; Termina o loop se o valor do duty cycle
; do PWM for negativo
    JP    LOOPFOUT
    JZ    LOOPFOUT
;
; Retorna da subrotina
    RET
;-----------------------------------
; Rotina para alterar o duty cycle PWM
;-----------------------------------
TRAP_PWM:
;
; Prepara parâmetros passados nas
; variáveis gloabais
    LDA   PWM_LOW
    STA   PARAM_TRAP+1
    LDA   PWM_HIGH
    STA   PARAM_TRAP+2
;
; Coloca no acumulador o número
; que corresponde ao TRAP que
; vai ser chamado. DUTY_CYCLE = 105
    LDA   #PWM_DUTY
;
; Faz a chamada da rotina de TRAP.
    TRAP  PARAM_TRAP
;
; espera 200 Milissegundos
    LDA   #DELAY_TRAP
    TRAP  TIME_200
;
; Retorna da subrotina
    RET
;-----------------------------------
; Variáveis auxiliares
;-----------------------------------
; Tempo utilizado da rotina "delay"
TIME_200:   DW    200
;
; Parâmetros da rotina de TRAP
; Declarado sem valores iniciais
PARAM_TRAP: DS    3
;
; Valor do duty cycle do PWM
; Incialmente iguais a zero
PWM_LOW:    DB    0
PWM_HIGH:   DB    0
;
;-----------------------------------
; Constantes do programa
;-----------------------------------
; Pino PWM (18 é o único por hardware)
;
PWM_PIN     EQU   18
;
; Parâmetros da rotina TRAP
;
PWM_MODE    EQU   3
PWM_DUTY    EQU   105
MODO_TRAP   EQU   101
DELAY_TRAP  EQU   5
;
; Fim do programa
    END   0
```

VII. Conclusões

Apresentamos neste artigo extensões acrescentadas ao simulador SimuS de modo que, quando executado no nanocomputador Raspberry Pi, seja possível o acesso aos pinos do conector GPIO, realizando a leitura e escrita de valores binários, além da geração de formas de onda PWM nos pinos disponíveis no conector para essas funções. Procuramos apresentar este processo com o maior nível possível de detalhes, incluindo a análise das várias opções de projeto que foram consideradas, de modo a auxiliar outros pesquisadores interessados no tema.

O transporte do código para a plataforma Raspberry Pi realizou-se com relativa facilidade, tendo em vista a disponibilidade do compilador Free Pascal e do ambiente de desenvolvimento Lazarus para esta plataforma. O desempenho final da aplicação se mostrou mais do que adequado, justificando o uso para fins educacionais, por ser uma plataforma de baixo custo, mesmo sem a utilização das modificações do simulador apresentadas neste artigo.

Apresentamos exemplos de código em linguagem de montagem para realizar funções simples como piscar ou realizar o "fade" da intensidade de um *LED*, que demonstram a fácil utilização destas funcionalidades a partir de um programa escrito em linguagem de montagem, de modo a controlar dispositivos reais através do conector GPIO do Raspberry.

Discutimos também as alternativas possíveis de implementação do acesso aos pinos do conector GPIO, justificando as opções escolhidas. Como o desempenho não é um fator relevante no caso do simulador, a opção foi por uma implementação mais fácil do ponto de vista de programação.

Como trabalhos futuros em relação a este tema, há diversas propostas, que incluem a utilização de outras funções da biblioteca ***wiringPi***, como controle de dispositivos I2C e SPI. Avaliamos ainda a possibilidade de utilização da comunicação via biblioteca Firmata para conexão com outros tipos de microcontroladores, ainda mais acessíveis e de menor custo que o Raspberry Pi, a partir de computadores convencionais.

Ressaltamos o ineditismo desta proposta, que esperamos possa se apresentar como uma opção bastante interessante para estímulo ao aprendizado dos conceitos básicos de Arquitetura de Computadores.

Referências

[1] J. A. S. Borges, G. P. Silva "SimuS - Um Simulador Para o Ensino de Arquitetura de Computadores". WEAC, 2016.

[2] J. A. S. Borges, G. P. Silva "NeanderWin - um simulador didático para uma arquitetura do tipo acumulador". WEAC, 2006.

[3] G. P. Silva, J.A.S Borges, O Simulador Neander-X para o Ensino de Arquitetura de Computadores.. Ed. Autor, 75 pp, 2016, disponível em http://www.amazon.com

[4] R. F. Weber, Fundamentos de Arquitetura de Computadores. 2. Ed. Porto Alegre: Instituto de Informática da UFRGS: Sagra Luzzatto, 2001.

[5] B. Nikoli, V. Milutinovic, "A survey and evaluation of simulators suitable for teaching courses in computer architecture and organization", IEEE Transactions on Education, Vol. 52, No. 4, November 2009

[6] S. Ratnakumar, "GNUSim8085, versão 1.3.7", https://gnusim8085.github.io/. Acesso em 23 de Junho de 2017

[7] M. Scott, "WinMips64, version 1.57", http://indigo.ie/~mscott/. Acesso em 20 de Julho de 2016.

[8] A. A. Moreira, C. A. P. S. Martins, "R2DSim: simulador didático do RISC reconfigurável". WEAC, pp 9-14, 2009.

[9] A. B. Verona, J. A. Martini, T. L. Gonçalves, "SIMAEAC: Um simulador acadêmico para ensino de arquitetura de computadores". I ENINED - Encontro Nacional de Informática e Educação, pp 424-432, 2009.

[10] E. V. C. L. Borges et al., "SEAC: um simulador online para ensino de arquitetura de computadores" - WEAC, pp. 34-38, 2012

[11] P. Sanderson, K. Vollmar "MARS Simulator" http://courses.missouristate.edu/KenVollmar/mars/download.htm. Acesso em agosto de 2016.

[12] G. Henderson. "Wiring Pi" Disponível em <http://wiringpi.com>, acesso em 08.08.2017

[13] Ali M., Vlaskamp J.H.A, Eddiny N.N., Falconer B. and Oram C., "Technical Development and Socioeconomic Implications of the Raspberry Pi as a Learning Tool in Developing Countries", 5th Computer Science and Electronic Engineering Conference (CEEC), pp. 103-108, 2013.

[14] Sriskanthan N., Tan F. and Karande A., "Bluetooth based home automation system", Microprocessors and Microsystems, Vol. 26, no. 6, pp. 281-289, 2002.

[15] YoonD., BaeD., KoH. and Kim H., "Implementation of Home Gateway and GUI for Control the Home Appliance", The 9th International Conference on Advanced Communication Technology, pp. 1583-1586, 2007.

[16] Mike Cook, "Understanding Outputs" Disponível em <http://www.thebox.myzen.co.uk/Raspberry/Understanding_Outputs.html>, acesso em 18.07.2017

[17] Lazarus Homepage Disponível em <https://www.lazarus-ide.org/>, acesso em18.07.2017

[18] Platform Extended Library. Disponível em <http://asphyre.net/platformextendedlibrary>, acesso em 08.08.2017.

[19] Disponível em <https://www.raspberrypi.org/blog/raspberry-pi-3-on-sale/>, acesso em 09.08.2017

[20] Disponível em <http://www.raspberrypi-spy.co.uk/2012/06/simple-guide-to-the-rpi-gpio-header-and-pins/>, acesso em 09.08.2017