International Journal of Computer Architecture Education (IJCAE) 2316-9915

Um Simulador Didático para o Ensino de Arquitetura de Computadores e Internet das Coisas

Eduardo Costa
Departamento de Ciência da Computação
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brasil
eduardocosta@dcc.ufrj.br

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

***Resumo*—O ensino de Arquitetura de Computadores e de Internet das Coisas apresentam convergências em diversos pontos. Contudo, existe ainda uma carência no desenvolvimento de ferramentas de ensino que possibilitem a apresentação de ambos os conteúdos de forma integrada para o aluno. Neste artigo, descrevemos as características e o desenvolvimento de uma ferramenta de simulação de processador que tem como objetivo integrar estas duas áreas de ensino. A ferramenta consiste, basicamente, de um simulador para o processador didático Sapiens. O simulador é executado no NodeMCU, uma plataforma de desenvolvimento para IoT de baixo custo baseada no *SoC* ESP8266, que possui WiFi. Os programas, previamente compilados e codificados no formato Intel HEX, podem ser carregados remotamente a partir de qualquer computador com uso de um navegador. Uma interface gráfica para o simulador foi desenvolvida em HTML e JavaScript, permitindo que o estudante acompanhe o estado dos registradores e memória do processador simulado. Foram feitas adições às instruções do Sapiens, para permitir o controle e leitura do valor dos pinos do NodeMCU diretamente pelo programa em execução no simulador. Com isso, é possível desenvolver programas em linguagem de montagem do Sapiens para interagir com dispositivos físicos, atuadores e sensores, que estejam conectados ao NodeMCU. Espera-se, assim, despertar um maior interesse dos alunos e facilitar a compreensão dos conceitos básicos de funcionamento de um processador e da Internet das Coisas.**

***Palavras-Chave*—simulador, processador, IoT, arquitetura de computadores**

I. Introdução

O ensino de Arquitetura de Computadores e de Internet das Coisas apresentam convergências em diversos pontos. Contudo, existe ainda uma carência no desenvolvimento de ferramentas de ensino que possibilitem a apresentação de ambos conteúdos de forma integrada para o aluno. Neste artigo, descrevemos as características e o desenvolvimento de uma ferramenta de simulação de processador que tem como objetivo integrar estas duas áreas de ensino.

Uma grande dificuldade no ensino de arquitetura de computadores é fazer com que os alunos compreendam corretamente o funcionamento de um processador. O uso de simuladores de processadores é uma estratégia muito utilizada para alcançar esse objetivo. Para a formação de um conhecimento sólido do funcionamento dos processadores é bastante importante a visão da arquitetura do processador, do formato das instruções e dos passos necessários para a execução de um programa por parte do estudante.

O simulador SimuS, disponível para Windows e Linux [1], conta com um ambiente integrado de desenvolvimento, para auxiliar na criação, compilação, execução e depuração do código para o processador hipotético Sapiens. Tanto o processador hipotético Sapiens como o simulador SimuS são apresentados em detalhes no livro "SimuS: Um Simulador Didático para Arquitetura de Computadores" [18].

O processador Sapiens apresenta uma arquitetura e o conjunto de instruções simplificados, mas com adições para se obter uma arquitetura com menos limitações. Dentre as melhorias, temos um maior espaço de endereçamento de memória, instruções para chamada e retorno de procedimentos, novos dispositivos de entrada/saída e a adição de instruções especiais de TRAP, que permitem ao usuário acessar dispositivos de entrada e saída mais complexos de uma maneira mais fácil, de modo similar a uma chamada de sistema, como ocorre com os modernos processadores e sistemas operacionais.

Com relação ao ensino de Internet das Coisas, a grande dificuldade encontrada é a abrangência de habilidades e conhecimentos necessários para o correto desenvolvimento de sistemas IoT: eletrônica básica; eletrônica digital; programação convencional; programação web; internet e sistemas em nuvem; para citar algumas.

Evidentemente que este conjunto de conhecimentos não pode ser adquirido em apenas uma passo e requer um conjunto de discplinas que apresente de forma gradual esses conhecimentos ao aluno. Neste sentido, acreditamos que uma aproximação adequada é inicialmente familiarizar o estudante com a programação de dispositivos sensores e atuadores de uso típico em IoT, para posteriormente avançar para disciplinas com maior aprofundamento de conteúdo.

A ferramenta mais popular utilizada com este objetivo é a plataforma de desenvolvimento Arduino, particularmente o Arduino Uno. Esta plataforma de baixo custo pode ser programada através de um computador convencional, bastando para isso que seja conectada com um cabo USB, onde programas, previamente editados e compilados em um ambiente integrado de desenvolvimento, podem ser carregados para o Arduino.

[1] https://github.com/sottam/simus

Apresenta como grande vantagem o uso de diversas bibliotecas que, de forma positiva, permitem esconder a complexidade de programação destes dispositivos dos alunos, permitindo que resultados possam ser alcançados mais rapidamente.

Este sistema, contudo, apresenta algumas deficiências: faz uso de um ambiente integrado de desenvolvimento baseado em linguagem "C", que apesar de procedural, oferece uma sintaxe complexa que inibe sua utilização pelos alunos menos experientes. Observamos que o contato linguagem de programação "C" tem ocorrido cada vez mais tardiamente, tendo em vista a preferência crescente pelo uso de linguagens como Python ou mesmo Javascript como a primeira linguagem de programação oferecida nos cursos de Ciência da Computação, Sistemas de Informação e Engenharia de Computação.

Adicionalmente, o método utilizado requer que o programa compilado seja carregado no Arduino, sem possibilidades de depuração passo-a-passo da aplicação, acesso a registradores ou memória do microncontrolador. O processo de tentativa e erro prosseque até que o aluno consiga o correto funcionamento do programa, o que pode ser longo e cansativo em muitos casos. O sistema não possui, nativamente, conexão com a internet e também há pouca memória RAM disponível no Arduino, o que pode frustante quando se pretende utilizá-lo em aplicações mais robustas.

Observamos que o ambiente de desenvolvimento integrado do Arduino, que é bastante popular, pode ser utilizado para programar outras plataformas de desenvolvimento. Entre elas destacamos o NodeMCU [5], uma sistema de desenvolvimento baseado no SoC ESP8266 [4]. O ESP8266 é um *chip* de baixíssimo custo, reduzidas dimensões, que integra um processador de 32 bits, memória *flash* de 4 MB e memória RAM de 160 KB, muito maiores que as presentes, por exemplo, no Arduino Uno. Além disso, possui pinos digitais para leitura e escrita, além de interface SPI, I2C e, principalmente, WiFi. Essas qualidadades tornam seu uso particularmente interessante para o ensino de IoT, permitindo o acesso a sensores e atuadores remotamente com o uso da Internet.

Neste artigo apresentamos a possibilidade de integração do simulador do processador Sapiens com o NodeMCU, permitindo que programas sejam baixados via WiFi ou USB e, mais importantes, depurados remotamente através de uma interface Web, tendo acesso aos pinos no NodeMCU, mas com todas as facilidades de um depurador, como execução passo-a-passo, pontos de parada, acesso aos registradores do processador Sapiens, conteúdo da memória, entre outros. A complexidade de acesso aos sensores e atuadores é escondida dos usuários através do recurso de instruções de TRAP, com chamadas feitas no código em linguagem de montagem, de fácil compreensão e aprendizado por parte dos alunos.

Novos tipos de TRAP foram adicionados ao simulador desenvolvido, de modo que o valor dos pinos do NodeMCU possam ser lidos e controlados diretamente pelo programa em execução no simulador. Com isso, é possível fazer programas que interajam com dispositivos físicos conectados ao NodeMCU, como botões, chaves, LEDs, motores DC, servo motores, *speakers*, *displays*, etc.

Adiconalmente, o NodeMCU pode ser montado numa placa *protoboard*, onde dispositivos físicos acoplados, como um *display*, um *keypad* de 12 teclas e 4 botões, para auxiliar na controle do simulador e acompanhamento da execução dos programas. Dessa forma, é possível executar programas desenvolvidos para o Sapiens em um hardware fora do computador, e visualizar os valores dos registradores em tempo real no *display*.

Com o uso deste simulador espera-se despertar desde cedo um maior interesse dos alunos e facilitar a compreensão dos conceitos básicos de funcionamento de um processador e da Internet das Coisas. Os programas desenvolvidos em linguagem de máquina do Sapiens agora poderão ir além das fronteiras da tela do computador.

II. Trabalhos Correlatos

De acordo com as recomendações da International Telecommunication Union [9], a arquitetura de rede da IoT consiste em cinco camadas: camada de detecção, camada de acesso, camada de rede, camada de *middleware* e camada de aplicação.

Particularmente a camada de detecção tem o papel de capturar as informações de interesse por vários tipos de sensores e compartilhar as informações capturadas nas unidades relacionadas na rede. A familiaridade desde cedo do aluno com esses tipos de sensores e atuadores é fundamental para a sua posterior aplicação em aplicações mais complexas de Internet das Coisas.

Existem muitas ferramentas para simulação de processadores, tanto comerciais quanto hipotéticos. Temos os simuladores de processadores comerciais mais usados em projetos de equipamentos atuais, como os de 8051, ATmega, PIC e variantes de processadores do tipo RISC. Porém, quase todos são destinados ao uso profissional, e alguns deles possuem licenças de uso pagas e ferramentas de uso complexo [17].

Há também simuladores didáticos de processadores comerciais, como o Abacus [21] e o GNUSim8085 [14], que simulam o 8085 da Intel que tem sido utilizados amplamente no ensino de arquitetura de computadores, devido à sua simplicidade.

Também existem os simuladores didáticos de processadores hipotéticos como o R2DSim [13], para simular uma arquitetura RISC de 16 bits, e o SIMAEAC [19], que implementa um subconjunto da arquitetura do 8085 [17]. O simulador SimuS, introduzido junto com o processador Sapiens, se encaixa nesta categoria. A proposta aqui apresentada tem bastante relação com o simulador SimuS, na parte relativa à execução do código de máquina do Sapiens. O SimuS, contudo, apresenta muitos outros recursos, detalhados na seção III-B.

A proposta com maior semelhança encontrada na revisão bibliográfica foi uma adaptação da versão de Linux do SimuS para rodar no Raspberry Pi, com modificações para tornar possível o uso dos pinos GPIO. Nesta versão todo simulador foi portado para o Raspbian e rotinas adicionais de TRAP permitem o acesso aos pinos do GPIO [17]. As diferenças se situam no fato de que na proposta aqui apresentada, apenas o simulador e o depurador são executados no NodeMCU, sem a

possibilidade de edição e compilação do código. Essas tarefas devem ser realizadas previamente em um computador à parte, para que depois o código compilado, em formato Intel HEX seja transferido para o NodeMCU.

Na versão para o Raspberry Pi não há nenhuma forma de acesso remoto via WiFi ou ethernet, o que só é possível na versão para o NodeMCU. Note-se ainda que o custo do Raspbery Pi, quando comparado ao NodeMCU, é bem mais elevado, dificultando a sua aquisição pelos estudadntes. Essa proposta é detalhada no livro "SimuS: Um Simulador Didático para Arquitetura de Computadores" [18].

III. Conceitos Gerais

Nesta seção serão apresentados conceitos básicos necessários para a compreensão do simulador, como a arquitetura do processador simulado e detalhes da placa de desenvolvimento utilizada na proposta.

A. O processador Sapiens

O Sapiens é um processador hipotético utilizado para o ensino de arquitetura de computadores, e foi desenvolvido como uma evolução do processador Neander-X [16].

Algumas melhorias importantes do Sapiens em relação ao Neander-X [17] são:

- Modo indireto e modo imediato de endereçamento;
- *Flag* de *carry* para "vai-um" e "vem-um";
- Expansão do conjunto de instruções para inclusão de diversos tipos de desvio condicional, novas operações lógicas e aritméticas, entre outras;
- Aumento do tamanho do apontador de instruções para 16 bits, permitindo endereçar até 64K posições de memória;
- Inclusão de um apontador de pilha, também de 16 bits;
- Instruções para chamada e retorno de procedimento;
- Instruções para a manipulação de dados na pilha.

B. O simulador SimuS

O simulador SimuS[2] foi introduzido junto com o processador Sapiens e está disponível em versões para Windows e Linux. A janela principal da interface de usuário pode ser vista na Figura 1.

Alguns dos recursos do SimuS são:

- Editor de texto;
- Montador (*assembler*) para gerar o código objeto final em linguagem de máquina;
- Execução contínua ou passo a passo (uma por vez), e possibilidade de pausar a execução;
- Modificação do valor dos registradores, de *flags*, e de valores da memória;
- *Breakpoints* para auxiliar na depuração;
- Simulador de dispositivos de entrada e saída, como um painel de chaves, um visor hexadecimal, um teclado de 12 teclas e um painel de 1 linha com 16 caracteres;

[2]https://github.com/sottam/simus

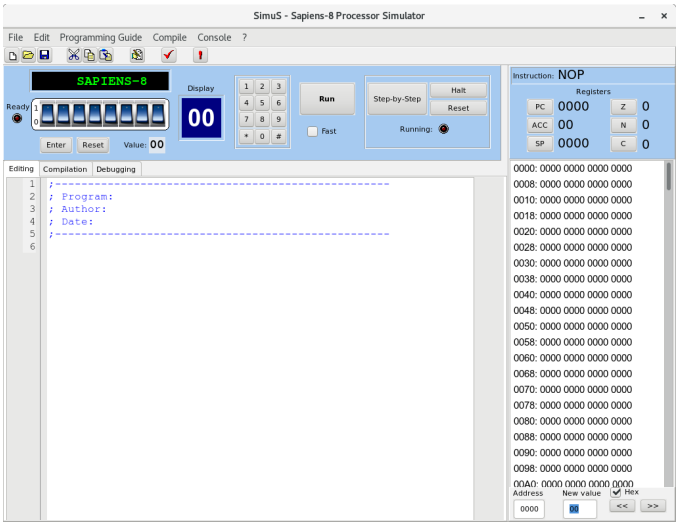

Fig. 1: Interface do simulador SimuS

- Exportação de um *dump* de memória no formato Intel HEX.

Além disso, estão disponíveis algumas diretivas do compilador, que não são instruções do Sapiens, mas sim diretivas que dão instruções para o compilador, como posições de variáveis na memória ou indicação da posição de início do código.

C. O NodeMCU

O sistema de utilizado com base foi a placa de desenvolvimento NodeMCU (Figura 2) que conta com um ESP8266, memória *flash* de 4 MB e um conversor *serial*-USB [5] [1]

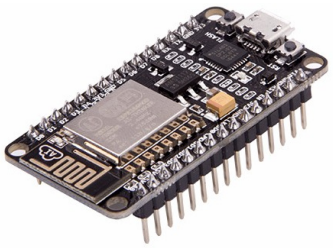

Fig. 2: A placa de desenvolvimento NodeMCU [15]

O ESP8266 é um SoC (*System on a Chip*) de baixo custo, com um processador de 32 bits e frequência de relógio de 80 MHz. Ele é largamente utilizado em aplicações de Internet das Coisas. Possui 160 KB de memória RAM, onde 80 KB estão disponíveis para o usuário. Além disso, possui diversos pinos de entrada/saída e WiFi [4].

O termo "NodeMCU" pode ser utilizado para se referir tanto à placa de desenvolvimento utilizada, quanto a um dos *firmwares* disponíveis para ela, detalhados a seguir. Dessa forma, sempre que o termo for utilizado aqui, será indicado de forma explícita quem está sendo referenciado.

O desenvolvimento para o ESP8266 pode ser feito de diversas formas, sendo possível escolher entre diversas linguagens e ambientes de desenvolvimento. Os três principais são:

- O *firmware* NodeMCU[3], para rodar *scripts* em Lua. A linguagem Lua tem a vantagem de tornar mais fácil o desenvolvimento, por ser de altíssimo nível. Por ser dinamicamente tipada e interpretada, o desenvolvimento de aplicações é mais rápido e fácil. Não é necessário se preocupar com alguns detalhes como na linguagem C. Ele é anunciado como uma plataforma de IoT simples e rápida, com exemplos de códigos de poucas linhas para rodar um servidor HTTP, por exemplo.
- Espruino[4], que conta com um interpretador de JavaScript, com diversas otimizações que melhoram o desempenho e o consumo de memória em relação ao JavaScript original. Inclui a opção de compilar o código em JavaScript para melhorar ainda mais o desempenho. O JavaScript, assim como a Lua, é uma linguagem de script de altíssimo nível, o que torna o desenvolvimento mais simples e rápido.
- IDE do Arduino[5], programando em C/C++. Através dela, é possível fazer uso do grande número de bibliotecas disponíveis para os mais diferentes módulos de *hardware*. O uso da linguagem C/C++ traz vantagens em relação ao consumo de memória e ao desempenho, em comparação com Lua e JavaScript. Como Lua e JavaScript são interpretadas e dinamicamente tipadas, é fácil chegar nos limites do ESP8266. Com o C ou C++, é possível aproveitar melhor e de forma mais eficiente o que o ESP8266 tem a oferecer.

IV. O SIMULADOR

Nesta seção será apresentada a idéia principal do simulador desenvolvido. Em seguida, será detalhado o formato do arquivo de entrada esperado. Por fim, será abordado o método de acesso aos pinos do NodeMCU através de instruções do Sapiens.

A proposta desenvolvida consiste de um simulador de código de máquina do Sapiens, a ser rodado no ESP8266. Uma versão do programa compilado em linguagem de máquina, num arquivo no formato Intel HEX [10], é enviado para o ESP8266, onde será executado.

O usuário pode interagir com o simulador de duas maneiras:

- Através de elementos físicos, numa *protoboard*, como botões, um teclado numérico e um *display*.
- Através de uma interface *web*, utilizando um navegador num computador na rede.

Na interface *web* é possível ter a experiência mais completa. É possível inserir *breakpoints*, visualizar a memória completa e interagir utilizando todos os dispositivos de entrada/saída. Na *protoboard*, por limitações físicas, nem todos os dados podem ser vistos no *display* e não estão disponíveis todos os dispositivos de entrada/saída. As duas maneiras serão detalhadas nas seções V e VI, respectivamente.

O simulador, que roda no ESP8266, foi desenvolvido em C++, através da IDE do Arduino. A interface *web* foi desenvolvida em HTML e JavaScript. O código está disponível no GitHub e pode ser acessado pelo endereço https://github.com/edunmc/tcc-ufrj.

[3]https://github.com/nodemcu/nodemcu-firmware
[4]http://www.espruino.com/EspruinoESP8266
[5]https://github.com/esp8266/Arduino

A. Limitações de memória e solução desenvolvida

O Sapiens endereça até 64 KB de memória RAM. Portanto, seria necessário alocar 64 KB de RAM para o perfeito funcionamento do simulador. O ESP8266 possui 80 KB de memória RAM disponível para o usuário, mas, após carregar o programa desenvolvido com todas as bibliotecas, restam apenas cerca de 30 KB para uso do simulador. Dessa forma, não é possível armazenar a memória inteira a ser simulada no ESP8266.

Para contornar este problema, os 64 KB de memória foram separados em páginas (blocos) de 1 KB, e cada página só é alocada quando é requisitada a escrita em algum endereço dentro dela. Assim, um programa pode utilizar, por exemplo, partes diferentes e distantes da memória, sem complicações.

Cada grupo de 1024 bytes forma uma página, e cada página é alocada por completo:

- Página 0, posições de 0 a 1023;
- Página 1, posições de 1024 a 2047;
- e assim por diante.

Este gerenciamento da memória e alocação de páginas é feito pela classe "Memoria". Neste classe, o método `ref` é utilizado para referenciar uma posição de memória, ao ser feita uma operação de escrita. O método `pega` é utilizado para uma operação de leitura. Assim, só é necessário passar a posição de memória para os métodos correspondentes, e todo o trabalho de descobrir a página à qual a posição pertence e alocar a página caso necessário é abstraído por esta classe. A seguir são mostrados exemplos.

```
// Escrever o valor 10 na posicao 1025 de memoria
memoria.ref(1025) = 10;

// Ler o valor da posicao 1025 de memoria
int valor = memoria.pega(1025);
```

Caso ainda não tenha sido por uma operação anterior, na operação de escrita à posição 1025 forçará a alocação desta página. Um vetor de 1024 posições será alocado, e uma referência à segunda posição deste vetor será retornada pela função `ref`. Em seguida, o valor 10 será colocado neste endereço. Na operação de leitura, o método `pega` identifica que a posição 1025 pertence à segunda página da memória, e acessa o vetor relativo a ela. Neste vetor, acessa o segundo elemento e retorna seu valor, que é colocado na variável `valor`.

Desta forma, o gerenciamento de páginas é feito de maneira completamente automática e abstraída, e a memória pode ser utilizada no resto do programa de forma transparente e sem preocupações. Como as páginas alocadas vão ocupando a memória disponível, e inicialmente há em torno de 30 KB disponíveis, só será possível alocar por volta de 30 páginas. Porém, na prática, isto seria muito difícil ocorrer, visto que o Sapiens é utilizado para fins didáticos. Dificilmente um programa desenvolvido nessas circunstâncias usará mais do que alguns kilobytes de memória.

B. Acesso aos pinos GPIO do ESP8266

Os programas executados pelo simulador podem acessar os pinos de GPIO do ESP8266. Com isso, códigos desenvolvidos para o Sapiens agora podem realizar operações como ler o sinal de algum pino, ou definir o nível lógico de saída de um pino como alto ou baixo. Dessa forma é possível interagir com dispositivos físicos numa *protoboard*, como LEDs e *speakers*, por exemplo.

O acesso aos pinos de GPIO é feito através de instruções de TRAP. Na Tabela I podem ser vistos os tipos de TRAP incluídos originalmente no Sapiens, e suportados pelo simulador SimuS.

Tipo	Descrição
1	Leitura de um caractere da console
2	Escrita de um caractere na console
3	Leitura de uma linha inteira da console
4	Escrita de uma linha inteira na console
5	Chama uma rotina de temporização
6	Chama uma rotina para tocar um tom
7	Chama uma rotina para retornar um número pseudo-aleatório de 0 a 99
8	Define a semente inicial da rotina de números aleatórios

Tabela I: Tipos de TRAP originais do Sapiens [18]

Além dos tipos de TRAP originais do Sapiens, no simulador desenvolvido foram adicionados novos tipos de TRAP para a comunicação com os pinos de GPIO do ESP8266.

Vale lembrar que os pinos GPIO da placa de desenvolvimento NodeMCU utilizada não são os números impressos na placa. Há uma equivalência entre os números de pino impressos na placa e os números de pinos GPIO, mostrada na Tabela II. Dessa forma, para acessar, por exemplo, o pino D5 da placa, o número do pino GPIO utilizado no código deve ser 14.

Na placa	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
GPIO	16	5	4	0	2	14	12	3	15	3	1

Tabela II: Equivalência entre pinos da placa e pinos GPIO [12]

1) Novos tipos de TRAP: Nas instruções de TRAP do Sapiens, o tipo de TRAP, indicando a função que se deseja realizar, é passado no acumulador. Já os parâmetros específicos da operação a ser feita, estão no endereço de memória passado como operando da instrução de TRAP. Na Tabela III são listados os novos tipos de TRAP introduzidos no simulador desenvolvido, assim como os parâmetros requeridos em cada um deles.

Como exemplo, apresentamos a seguir um trecho de código que define o pino 0 do NodeMCU como sendo um pino de saída e depois escreve nele o valor 1. Primeiramente, o valor do tipo da TRAP (101 ou 102) é colocado no acumulador, pois esta é a forma com que a instrução TRAP reconhece o tipo de serviço que está sendo solicitado. Em seguida os parâmetros são passados no endereço de memória que é o operando (OPERANDO1 ou OPERANDO2) da instrução TRAP.

Tipo	Descrição	Parâmetros
101	Configurar pino como entrada ou saída	Pino (1 byte) Modo (1 byte)
102	Escreve o valor lógico de saída num pino	Pino (1 byte) Valor (1 byte)
103	Lê o valor de entrada de um pino para o acumulador	Pino (1 byte)
104	Configurar resistencia *pull-up* ou *pull-down* num pino	Pino (1 byte) PUD (1 byte)
105	*Duty cycle* do PWM do pino	Pino (1 byte) Valor (2 bytes)

Tabela III: Novos tipos de TRAP adicionados

```
LDA #101        ; configurar pino como saida
TRAP OPERANDO1
LDA #102        ; escrever 1 no pino
TRAP OPERANDO2

OPERANDO1 : DB 0 , 1  ; pino , modo
OPERANDO2 : DB 0 , 1  ; pino , valor
```

C. Interagindo com o simulador

É possível interagir com o simulador de duas maneiras:

- Físicamente, através de botões, um *display* e um *keypad*, na *protoboard*; ou
- Através de uma interface *web*.

As duas maneiras serão exploradas com detalhes nas seções seguintes.

V. Protoboard

Nesta seção apresentamos um circuito com alguns dispositivos físicos que permitem uma interação básica com o simulador. Uma interação mais completa está disponível com a utilização da interface *web*, detalhada na seção seguinte.

A. Circuito proposto

Uma das formas de interagir com o simulador desenvolvido, é a partir de dispositivos físicos num circuito montado numa *protoboard*. Foi montado o circuito exibido na Figura 3, com a placa de desenvolvimento NodeMCU, que conta com o ESP8266.

No circuito, temos:

- A placa de desenvolvimento NodeMCU;
- O *display* SSD1306, de resolução 128x64, usando o protocolo I2C para se conectar ao ESP8266;
- Um *keypad* de 12 teclas;
- Um circuito integrado PCF8574, para converter os 7 pinos paralelos do *keypad* para usar o protocolo I2C, que usa apenas 2 pinos.
- Quatro botões (*push buttons*) para controlar o fluxo de execução: iniciar execução contínua, passo a passo, pausar e resetar, respectivamente.

[6]http://fritzing.org

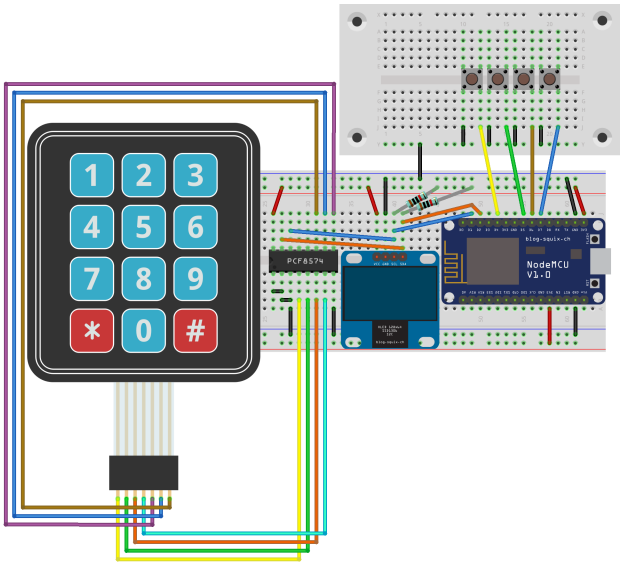

Fig. 3: Circuito montado, desenhado no Fritzing[6]

Nem todos os dispositivos de E/S estão disponíveis na *protoboard*, por limitações de espaço, assim como também não há como examinar o conteúdo da memória simulada. Para estes casos, deve ser utilizada a interface *web*, mais completa, detalhada na Seção VI.

B. Dispositivos de entrada/saída

O *display* utilizado é o SSD1306 [6]. Ele tem resolução de 128x64, e utiliza o protocolo I2C para se conectar ao ESP8266. Foi utilizada a biblioteca Adafruit_SSD1306[7]. O *display* é usado para a exibição do valor dos registradores e do *banner* de 16 caracteres. Na Figura 4, é possível ver uma foto do *display* durante a execução de um programa. Nele, são exibidos os valores dos registradores (incluindo as *flags*: *negative*, *zero* e *carry*) e o *banner* de 16 caracteres.

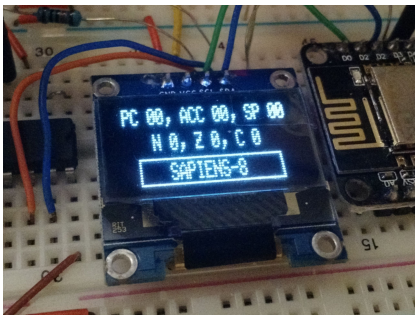

Fig. 4: O *display* durante a execução de um programa

O *keypad* de 12 teclas pode ser usado como no simulador SimuS. Quando uma tecla é pressionada, o status "pronto" do keypad é definido, e o valor da última tecla pressionada é salvo.

[7]https://github.com/adafruit/Adafruit_SSD1306

C. Interação com o simulador

Sem a utilização da interface *web*, o envio de arquivos pode ser feito pela interface *serial*, com a placa de desenvolvimento conectada ao computador pela USB. Para o envio do código do programa a ser executado, deve ser utilizado um *dump* de memória no formato Intel HEX do código de máquina Sapiens, que pode ser obtido através do simulador SimuS.

Ao ser iniciado, o ESP8266 tentará se conectar à rede Wi-Fi utilizando o nome e senha definidos no código, no início do arquivo "web.ino". Após se conectar, ou depois de alguns segundos sem sucesso na tentativa de conexão, será ativado o modo em que o envio de um arquivo é aguardado, com a mensagem "Aguardando o envio do arquivo Intel HEX..." exibida no *display*. Neste momento, o envio do arquivo pode ser feito pela interface *serial*. Como o formato Intel HEX possui um identificador de fim de arquivo, é possível saber que o envio terminou, e o simulador identifica isso automaticamente. No *display*, isso poderá ser identificado pela exibição dos registradores e do *banner*.

Após o envio do código, é possível controlar o fluxo de execução através dos quatro botões. São eles, da esquerda para a direita: iniciar execução contínua, passo a passo, pausar execução e reiniciar.

VI. Interface Web

Nesta seção apresentamos a interface *web* do simulador. São detalhadas suas várias funcionalidades, muitas delas não disponíveis na interação através da *protoboard*. A seguir, é discutida a arquitetura e comunicação da interface *web* com o ESP8266, com detalhes sobre protocolos utilizados e formato de mensagens enviadas. Por fim, são apresentadas as dificuldades encontradas durante seu desenvolvimento, assim como soluções utilizadas para superá-las.

A. A interação através da interface web

A interface *web* é a maneira mais completa de interagir com o simulador. Para acessá-la, basta acessar o endereço IP do ESP8266 através de um computador conectado à mesma rede WiFi. Na Figura 5, é possível ver a interface *web* assim que a página é acessada. O desenvolvimento foi feito em HTML e JavaScript.

À esquerda no topo há um campo para carga do arquivo com o código de máquina no formato Intel HEX. Na esquerda embaixo estão os dispositivos virtuais de entrada e saída. No centro, são exibidos os registradores e a pode-se visualizar toda a memória. À direita no topo, temos os botões de ações como: executar, parar e passo a passo, para controlar o fluxo de execução; e embaixo, um *disassembler*, que exibe o código em linguagem de máquina gerado a partir do código de máquina do arquivo enviado.

O endereço 192.168.1.32 acessado foi o endereço IP atribuído ao ESP8266 pela rede. Este IP é exibido no *display*, logo que o programa é iniciado. Ao ser iniciado, é exibida a mensagem "Conectando..." no display, e é feita a tentativa de conexão na rede com os dados informados no código (nome e senha), no início do arquivo "web.ino". Depois de alguns

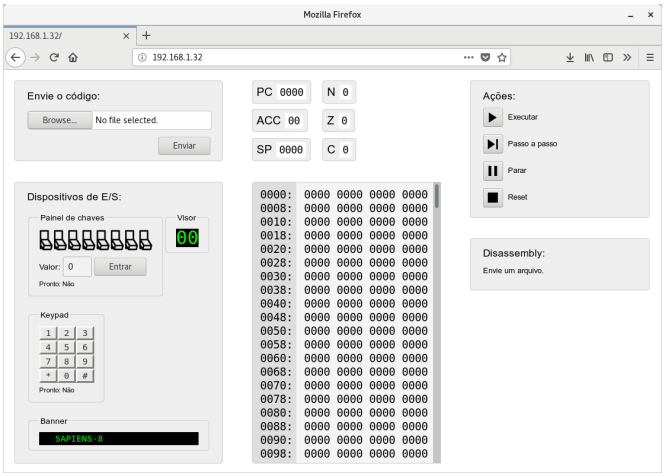

Fig. 5: Interface *web* do simulador

segundos, caso a conexão seja bem sucedida, será exibida a mensagem "Conectado." e o IP do servidor *web* para ser acessado, como mostrado na Figura 6.

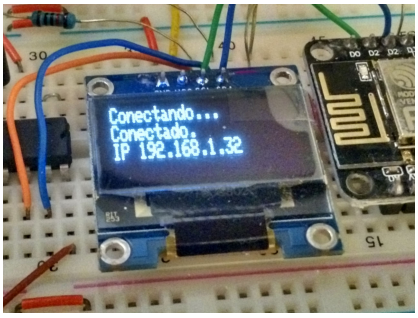

Fig. 6: Display exibindo informações da conexão Wi-Fi

Todos os recursos do simulador estão disponíveis na interface *web*. São eles:

- Todos os dispositivos virtuais de E/S: painel de chaves, um visor hexadecimal, um teclado de 12 teclas e um *banner* de 1 linha com 16 caracteres;
- Toda a memória pode ser examinada;
- *Disassembler* do código: a partir do conteúdo de memória carregado no simulador, é possível ver as instruções às quais os valores correspondem;
- Registradores;
- *Breakpoints*: no *disassembler*, é possível clicar numa linha de código para adicionar um breakpoint;
- *Upload* do código de forma simples.

A Tabela IV compara os recursos do simulador e os dispositivos de entrada/saída disponíveis fisicamente na *protoboard* e na interface *web*. Os dispositivos de entrada/saída são discutidos com mais detalhes na sessão seguinte.

B. Breakpoints

No canto inferior direito há o *disassembler*, que mostra os mnemônicos em linguagem de montagem gerados a partir do código de máquina do arquivo de entrada, com uma instrução por linha. É possível clicar em uma linha para definir um *breakpoint*. Um *breakpoint* é indicado por um pequeno círculo à esquerda da linha, como mostra a Figura 7. Para removê-lo, basta clicar na linha novamente. Quando um *breakpoint* está definido, a execução contínua é interrompida logo antes da execução daquela instrução.

	Protoboard	**Interface web**
Registradores	Sim	Sim
Memória	Não	Sim
Disassembler do código	Não	Sim
Breakpoint	Não	Sim
Painel de chaves	Não	Sim
Visor hexadecimal	Não	Sim
Teclado de 12 teclas	Sim	Sim
Banner de 16 caracteres	Sim	Sim

Tabela IV: Comparação de recursos do simulador na interface *web* e na *protoboard*

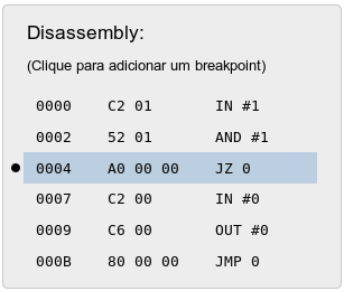

Fig. 7: Exemplo do *disassembler* com um *breakpoint* inserido

C. Arquitetura

Quando o usuário acessa a interface *web* através do navegador de um computador na rede, uma requisição é feita para o servidor HTTP do ESP8266, na porta 80. O servidor HTTP irá então responder à requisição com o código HTML/JavaScript da página da interface *web*. Ao carregar esta página será executado o código JavaScript, que abrirá uma conexão com o servidor WebSockets na porta 81. Esta conexão é bidirecional e ficará aberta o tempo todo. Toda a comunicação entre a interface *web* e o ESP8266 ocorrerá através dela: sempre que o simulador alterar algum dado por conta da simulação, uma mensagem com as alterações será enviada para a interface *web*; sempre que o usuário interagir de alguma forma na interface *web* (clicando num botão ou enviando um arquivo, por exemplo), uma mensagem será enviada para o ESP8266. Na Figura 8 é possível ver um diagrama da arquitetura utilizada para a interface *web* e sua comunicação com o ESP8266.

Portanto, para o funcionamento da interface *web*, são utilizados dois servidores:

- Servidor HTTP, na porta 80, através da biblioteca *ESP8266WebServer*[8], para servir a página HTML;

[8]https://github.com/esp8266/Arduino/tree/master/libraries/ESP8266WebServer

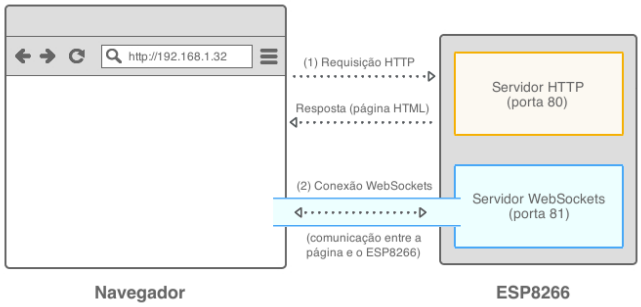

Fig. 8: Diagrama da arquitetura utilizada para a interface *web*

- Servidor WebSockets, na porta 81, através da biblioteca *arduinoWebSockets*[9], para a comunicação bidirecional entre a página e o ESP8266.

A seguir, serão exibidos trechos do código necessário para a utilização desses servidores, com explicações detalhadas sobre seu funcionamento. A proposta foi desenvolvida com o foco em apenas um cliente. Apesar de não haver problemas com a conexão de mais de um cliente para a visualização da simulação, a interação com o simulador deve ser feita por apenas um único cliente.

D. Comunicação da interface web *com o ESP8266*

Toda a comunicação entre a interface *web* e o ESP8266 é feita através do protocolo WebSockets. O protocolo WebSockets mantem uma conexão aberta, e os dois lados podem enviar mensagens. Dessa forma, não há a necessidade de fazer *polling* a cada segundo, por exemplo, para perguntar se há dados novos [7]. No protocolo WebSockets, a conexão fica aberta, e sempre que há alguma alteração no valor de algum registrador ou posição de memória, o ESP8266 envia para a interface *web*.

Da mesma forma, sempre que alguma ação for feita na página *web*, como a inserção de um dado num dispositivo de entrada/saída ou um clique no botão de iniciar ou pausar a execução, uma mensagem é enviada para o ESP8266, para que a ação seja refletida no simulador. As mensagens que a página envia para o microcontrolador seguem um protocolo próprio, detalhado a seguir.

1) Mensagens enviadas da interface web *para o ESP8266:* São enviadas mensagens da interface *web* para o ESP8266 sempre que o usuário interage de alguma forma. Alguns exemplos são exibidos a seguir.

a) Adicionar breakpoint *:* `b` `p` `,` byte1 byte2 `\0`

b) Remover breakpoint *:* `b` `p` `n` `\0`

c) Upload *de* dump *de memória :* `e` `n` `v` `i` `a` `,` conteúdo do arquivo

d) Iniciar execução (contínua) : `p` `l` `a` `y` `\0`

e) Parar execução : `s` `t` `o` `p` `\0`

2) Mensagens enviadas do ESP8266 para a interface web*:* Todas as mensagens que o ESP8266 envia para a página são no formato JSON, formato tipicamente usado em aplicações de IoT. Este formato tem a facilidade de poder ser transformado num objeto nativo do JavaScript, linguagem utilizada na interface *web*, e, assim, ser manipulado de maneira bem simples. É mostrado um exemplo a seguir.

```
{
  "registradores": {
    "pc": "00AA",
    "acc": "BB",
    "sp": "00CC",
    "n": "0",
    "z": "1",
    "c": "0"
  },

  "memoria": {
    "4": "15",
    "9": "C6",
    "7": "16"
  }
}
```

Sempre que o simulador faz alguma alteração na memória, apenas as posições alteradas da memória são enviadas. Quando há apenas alterações em registradores, o valor do campo "memoria" é vazio.

No JavaScript, o JSON pode ser transformado em um objeto nativo, e o acesso aos elementos fica bastante fácil. Por exemplo: caso o objeto se chame `dados`, para acessar a posição 4 de memória é usado apenas `dados.memoria[4]`, e para acessar o registrador PC, `dados.registradores.pc`.

E. Otimizações

Algumas otimizações foram feitas, por conta de complicações que surgiram durante o desenvolvimento.

Inicialmente, a troca de mensagens da interface *web* com o ESP8266 era feita com requisições HTTP comuns, através de *polling*. A cada segundo, a página enviava uma requisição, e o simulador retornava como resposta o conteúdo dos registradores e de todas as posições de memória diferentes de 0. Isso acabava sendo ruim por diversos motivos: primeiramente, era preciso fazer uma conexão HTTP por segundo, o que inclui todo o *overhead* de abrir a conexão no início, e fechar no final. Além disso, muitas vezes a conexão era feita à toa: se não houvesse nenhuma modificação, o conteúdo seria enviado de novo. Em terceiro lugar, o envio da memória inteira adicionava um grande custo. Em programas maiores, acabava sendo uma quantidade não tão pequena de dados, para ser enviada a cada segundo.

Para contornar estes problemas, primeiramente foi utilizado o protocolo WebSockets. Neste protocolo, uma conexão é mantida aberta entre o servidor e cliente, e os dois lados podem enviar mensagem quando algum evento ocorrer [7]. Dessa forma, a conexão pôde ficar aberta sem transmitir nada a

[9]https://github.com/Links2004/arduinoWebSockets

menos que seja necessário. Quando alguma mudança no valor do registrador ou da memória é feita no simulador, ele envia as alterações para a interface *web*. Não é mais preciso que a conexão seja iniciada pela interface *web*. Além disso, outra mudança foi o envio apenas das modificações da memória. Assim, a memória é enviada apenas quando há alteração, e apenas os bytes alterados são enviados. A interface *web* recebe esses dados alterados e substitui na memória salva. Somente no início é enviada a memória inteira.

VII. Trabalhos Futuros

A. Novos dispositivos

Uma possível adição ao simulador seria a inclusão de novos dispositivos físicos de entrada e saída na *protoboard*, como interruptores (*switches*) físicos para o painel de chaves e *displays* de 7 segmentos para o visor hexadecimal.

B. Uso do ESP32

Como trabalho futuro a se considerar está o porte o simulador para o ESP32. O ESP32 é sucessor do ESP8266, e conta com um processador melhor, mais memória, e algumas outras adições [3] [2]. Como o ESP32 possui 520 KB de RAM, muito mais do que os 64 KB de memória endereçados pelo Sapiens, não haveria os problemas de memória que existiram com o ESP8266. Toda a memória poderia ser alocada. Além disso, o ESP32 também possui Bluetooth, abrindo um leque de novas possibilidades de dispositivos de entrada e saída. Foram feitos testes em um ESP32, mas algumas bibliotecas utilizadas ainda não haviam sido portadas para o ESP32.

C. Melhorias na interface web

Algumas adições à interface *web* que seriam relevantes são, primeiramente, a possibilidade da edição da memória, como existe no simulador SimuS, através de um clique na posição de memória. Ao clicar numa posição, um campo seria exibido, para que fosse possível inserir um novo valor para aquele endereço.

D. Novos tipos de TRAP

Além das TRAPs desenvolvidas, como leitura e escrita de pinos digitais, seria interessante a implementação de novos tipos de TRAP, tais como:

- a leitura de valor analógico, para que fosse possível ler o valor de sensores;
- utilização da interface I2C, para que fosse possível o uso do display, que atualmente só é usado caso a interação não seja feita pela interface web;
- controle de dispositivos específicos, como um motor servo, especificando o pino e a posição desejada.

E. Implementação do protocolo MQTT

Uma facilidade adicional interessante seria a inclusão no menu do simulador de uma janela de configuração MQTT. Nesta janela seriam fornecidos os parâmetros de forma a possibilitar o envio dos dados coletados na placa de desenvolvimento NodeMCU para um servidor de MQTT. Desta forma, também de uma forma transparente ao estudante, os dados coletados nos pinos poderiam ser enviados para um *broker* MQTT, mediante a programação de tópicos adequados, onde seriam armazenados, permitindo o seu posterior tratamento por outras ferramentas desenvolvidas com outras linguagens, em disciplinas ministradas em fases mais avançadas do curso.

F. Avaliação do Uso do Simulador

Está em curso uma avaliação do uso prático do simulador em uma disciplina do curso de Bacharelado de Ciência da Computação. Os experimentos incluem o desenvolvimento de código em linguagem de montagem do Sapeins para realizar tarefas simples com piscar e mudar a intensidade de um *led*, ativar motores, implementar contadores e temporizadores, entre outras. Contudo, ao tempo da redação do presente artigo a mesma ainda não havia sido concluída.

VIII. Conclusões

O ensino de internet das coisas apresenta novos desafios para o educador, principalmente por abordarem temas disciplinares tão distintos entre si. Acreditamos que conjugar o seu ensino com o de arquitetura de computadores nos primeiros semestres do curso, com o apoio de ferramentas de simulação, pode ser uma alternativa promissora e efetiva para a solução deste problema.

Neste artigo, apresentamos um simulador do processador hipotético Sapiens, executado no microcontrolador ESP8266, voltado para o ensino de Arquitetura de Computadores e Internet da Coisas. É uma ferramenta que pode ser utilizada em disciplinas iniciais de Arquitetura de Computadores, com exemplos práticos que envolvam a coleta de dados em sensores e o acionamento de dispositivos como leds, *displays* e motores.

Foram apresentados detalhes minuciosos de sua implementação, para auxiliar aqueles que desejem o desenvolvimento de ferramentas similares. O uso das instruções de TRAP permite ocultar do estudante a complexidade de acesso aos dispositivos de IoT, apresentando-se uma interface de fácil programação, com parâmetros simples e bem definidos. A possibilidade de simulação dos programas passo-a-passo, colocação de pontos de parada, o exame dos registradores e a visualização do conteúdo da memória são diferenciais desta proposta sobre outras ferramentas similares.

Pode-se observar, no desenvolvimento da proposta, a união entre as áreas de Arquitetura de Computadores e Internet das Coisas, sendo possível utilizar código de máquina para a interação com um circuito na *protoboard*, além da possibilidade da utilização da interface *web* para a interação com o simulador, sendo também o próprio simulador um excelente exemplo de aplicação IoT. O protocolo WebSockets e o formato JSON, utilizados na comunicação entre o ESP8266 e a interface *web*, são soluções bem utilizadas no contexto de IoT [11] [20] [8]. Além disso, o envio apenas dos dados que foram alterados, ao invés de reenviar sempre tudo, é bastante comum em aplicações deste contexto.

Com este simulador, espera-se despertar um maior interesse dos alunos em disciplinas de Arquitetura de Computadores e de Circuitos Lógicos. Agora, é possível interagir com dispositivos físicos na *protoboard* a partir do código de máquina do processador Sapiens, e os limites vão além do mundo virtual do computador.

O baixíssimo custo do sistema de desenvolvimento utilizado permite que ele possa ser adquirido com facilidade, tanto pela universidade como pelos próprios estudantes, e que sejam utilizados em sala de aula ou em tarefas fora de sala, sem a necessidade de equipamentos sofisticados.

A partir de modificações no circuito proposto, é simples adicionar ao simulador outros dispositivos físicos e utilizar programas simples escritos em linguagem de montagem para interagir com eles. Todos os códigos do simulador desenvolvidos estão disponíveis no GitHub[10], e são livres para serem modificados e utilizados.

REFERÊNCIAS

[1] ESP-12E WiFi Module. Disponível em: <https://www.kloppenborg.net/images/blog/esp8266/esp8266-esp12e-specs.pdf>. Acesso em: 06 maio 2018.

[2] ESP32 Datasheet: Version 2.3. 2018. Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>. Acesso em: 06 maio 2018.

[3] ESP32 Overview. 2018. Disponível em: <https://www.espressif.com/en/products/hardware/esp32/overview>. Acesso em: 06 maio 2018.

[4] ESP8266EX Datasheet. 2018. Disponível em: <https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf>. Acesso em: 06 maio 2018.

[5] NodeMCU DEVKIT V1.0. 2018. Disponível em: <https://github.com/nodemcu/nodemcu-devkit-v1.0/blob/master/README.md>. Acesso em: 06 maio 2018.

[6] SSD1306. 2008. Disponível em: <https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf>. Acesso em: 06 maio 2018.

[7] WebSockets. 2018. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API>. Acesso em: 06 maio 2018.

[8] ADA, L. All the Internet of Things - Episode Two. 2017. Disponível em: <https://learn.adafruit.com/alltheiot-protocols?view=all#whats-the-difference>. Acesso em: 06 maio 2018.

[9] CHEN, X.-Y.; JIN, Z.-G. Research on Key Technology and Applications for Internet of Things. *Physics Procedia 33* (2012), 561–566.

[10] INTEL. Hexadecimal Object File Format Specification. 1998. Disponível em: <https://web.archive.org/web/20160607224738/http://microsym.com/editor/assets/intelhex.pdf>. Acesso em: 15 abr. 2018.

[11] MADAN, D. Unleashing the power of HTML5 WebSocket for Internet of Things. 2015. Disponível em: <https://www.hcltech.com/blogs/unleashing-power-html5-websocket-internet-things-iot>. Acesso em: 06 maio 2018.

[12] ESP8266. Arduino/pins_arduino.h - esp8266/Arduino. 2018. Disponível em: <https://github.com/esp8266/Arduino/blob/master/variants/nodemcu/pins_arduino.h#L37-L59>. Acesso em: 06 maio 2018.

[13] MOREIRA, A. A.; MARTINS, C. A. P. S. R2DSim: Simulador Didático do RISC Reconfigurável. In: WORKSHOP SOBRE EDUCAÇÃO EM ARQUITETURA DE COMPUTADORES, 2009, São Paulo. Anais... [S.l.: s.n.], 2009.

[14] RANGANATHAN, A. Gnusim8085. 2018. Disponível em: <https://launchpad.net/gnusim8085>. Acesso em: 29 abr. 2018.

[15] SEEED TECHNOLOGY CO., L. NodeMCU v2 - Lua based ESP8266 development kit. 2017. Disponível em: <https://www.seeedstudio.com/NodeMCU-v2-Lua-based-ESP8266-development-kit-p-2415.html>. Acesso em: 29 abr. 2018.

[16] SILVA, G. P.; BORGES, J. A. DOS S. Neanderwin - Um Simulador Didático para uma Arquitetura do Tipo Acumulador. 2006. Disponível em: <http://dcc.ufrj.br/~gabriel/WEAC2006.pdf>. Acesso em: 15 abr. 2018.

[17] SILVA, G. P.; BORGES, J. A. DOS S. Simus: Um Simulador Para o Ensino de Arquitetura de Computadores. 2016. Disponível em: <http://www2.sbc.org.br/ceacpad/ijcae/v5_n1_dec_2016/IJCAE_v5_n1_dez_2016_paper_2_vf.pdf>. Acesso em: 15 abr. 2018.

[18] SILVA, G. P.; BORGES, J. A. DOS S. SimuS: Um Simulador Didático para Arquitetura de Computadores. Rio de Janeiro: Ed. do Autor, 2017.

[19] VERONA, A. B.; MARTINI, J. A.; GONÇALVES, T. L. SIMAEAC: Um simulador acadêmico para ensino de arquitetura de computadores. 2009. Disponível em: <https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvb%20WFpbnxub3ZvbWlsZW5pbzIwMTQwMXxneDo0MTE2YzA3M2J%20jMzYzYjhi>. Acesso em: 29 abr. 2018.

[20] VISWANATH, A. How JSON and Big Data Will Shape the Internet of Things. 2013. Disponível em: <http://java.sys-con.com/node/2881856>. Acesso em: 06 maio 2018.

[21] ZILLER, R. Microprocessadores: Conceitos Importantes. 2. ed. Florianópolis: R. M. Ziller, 2000.

[10] https://github.com/edunmc/tcc-ufrj