# Desenvolvimento de um Sistema Operacional de Tempo Real para um Microcontrolador Básico

Hendrig W. M. S. Gonçalves, Fabrício Bortoluzzi
Laboratório de Redes
Universidade do Vale do Itajaí, UNIVALI
Itajaí, Brasil
{hendrig, fb}@univali.br

Cesar Albenes Zeferino
Laboratório de Sistemas Embarcados e Distribuídos
Universidade do Vale do Itajaí, UNIVALI
Itajaí, Brasil
zeferino@univali.br

***Abstract* — Este artigo apresenta o desenvolvimento de um sistema operacional de tempo real para um microcontrolador básico, visando o uso no ensino de conceitos de sistemas operacionais. É descrita uma análise realizada sobre sistemas operacionais de tempo real já existentes e o processo de criação de um novo sistema operacional baseado nos sistemas estudados. O artigo também apresenta a arquitetura desse microcontrolador e as extensões realizadas para suportar o sistema operacional.**

***Palavras-chave—Arquitetura e Organização de Computadores; Sistemas Operacionais; Sistemas Operacionais de Tempo Real***

## I. INTRODUÇÃO

O ensino da disciplina de sistemas operacionais frequentemente aborda um conjunto de conceitos elementares ou clássicos, a exemplo da gerência de processos, memória, entrada e saída e arquivos, como também pode incluir tópicos mais profundos e específicos: abordagens de escalonamento, critérios de tempo real, estruturação de máquinas virtuais entre outros.

Reflexões construtivistas, como as discutidas em [1], evidenciam que o modelo tradicional de ensino, baseado unicamente na apresentação de abstrações e teorias, pode ser melhorado quando se oferece ao estudante uma oportunidade prática para consolidação dos conceitos adquiridos. A prática melhora a assimilação porque permite a reorganização de idéias e a busca inerente por um equilíbrio entre o que foi discutido e o que é materializado por meio de tentativa e erro.

Um sistema operacional minimamente capaz de escalonar processos, constituindo uma Interface de Programação de Aplicação igualmente mínima, livre de qualquer complexidade desnecessária ao aprendizado, poderia servir como recurso didático valioso para materializar as abstrações da disciplina em um pequeno sistema embarcado capaz de acomodá-lo.

Nesse contexto, em um projeto de pesquisa da Universidade do Vale do Itajaí – Univali – foi especificada uma arquitetura de processador básico com o propósito de facilitar o aprendizado de conceitos introdutórios de arquitetura e organização de computadores em fases iniciais de cursos de graduação na área de Computação [2]. Com base nessa arquitetura, foram definidos cinco modelos com complexidade incremental, além de serem desenvolvidas ferramentas de apoio ao ensino (compilador, montador e simulador). Esse conjunto de processadores foi denominado BIP (*Basic Instruction-set Processor*) e sua família inclui quatro modelos de processador (BIP I a IV) e um microcontrolador (µBIP), modelo mais completo.

O BIP já foi aplicado como arquitetura de referência nas disciplinas "Algoritmos e Programação", "Circuitos Digitais" e "Arquitetura e Organização de Computadores" do curso de bacharelado em Ciência da Computação da Univali, servindo de importante instrumento pedagógico para o aprendizado seguindo uma abordagem interdisciplinar.

Além das disciplinas supracitadas, observou-se que o BIP teria potencial para uso na disciplina "Sistemas Operacionais". Porém, identificou-se a necessidade da realização de um estudo para confirmar essa possibilidade e identificar as modificações necessárias para suporte à implementação de sistemas operacionais.

Esse problema motivou a realização de uma pesquisa com intuito de verificar a viabilidade de se implementar um sistema operacional mínimo no µBIP para suporte ao escalonamento de tarefas baseado nos sistemas operacionais de tempo real. Para tal, foi feito um estudo sobre sistemas operacionais de tempo real e a análise de sistemas já existentes para microcontroladores comerciais com o objetivo de identificar quais recursos e conceitos seriam úteis para a construção de um sistema operacional para o microcontrolador µBIP.

Seguindo, foi feita a análise da arquitetura do µBIP a fim de verificar a necessidade de se estender a sua arquitetura para viabilizar a criação do sistema operacional. Após, foi realizada a implementação em linguagem de montagem do sistema operacional, o qual foi denominado BIP/OS.

Este artigo apresenta os aspectos relacionados ao desenvolvimento do BIP/OS. Ele é organizado em sete seções. A Seção II apresenta um revisão de conceitos básicos sobre sistemas operacionais de tempo real e uma breve análise dos sistemas para microcontroladores comerciais analisados nesta pesquisa. A Seção III descreve a arquitetura do microcontrolador µBIP e discute as extensões necessárias à sua arquitetura para suportar o sistema operacional. Continuando, a Seção IV descreve a arquitetura do BIP/OS, enquanto que a Seção V apresenta aspectos referentes à sua implementação. Na Seção VI, são apresentados os resultados

do trabalho, com a descrição dos procedimentos de teste e a caracterização do custo e do desempenho do BIP. Concluindo, na Seção VII são apresentadas as considerações finais.

## II. Sistemas Operacionais de Tempo Real

Um sistema operacional de tempo real ou RTOS (*Real-Time Operating System*) é sistema operacional cuja característica mais importante é a utilização e/ou controle do tempo por parte dos processos que executam sobre o mesmo. Um RTOS possui como características o determinismo com relação às tarefas que o sistema irá executar, responsividade, alto grau de controle do usuário, confiabilidade e suporte a operações de falha de software [3]. Um RTOS é usado em aplicações críticas quanto ao tempo, como sensores de monitoração e, geralmente, são sistemas pequenos, para uso em computadores embarcados [4].

As subseções a seguir apresentam conceitos e métricas utilizadas para caracterizar os RTOS comerciais e o BIP/OS.

### A. Atributos de um RTOS

Um sistema operacional de tempo real possui como característica a execução de suas tarefas dentro de um tempo específico e conhecido [5]. Os RTOS podem ser classificados em *hard* RTOS, que são sistemas em que o tempo de execução da tarefa é um elemento crítico, e *soft* RTOS, nos quais o tempo de execução é importante, mas não crucial.

Em um RTOS, também é importante conhecer o tempo que o sistema leva para trocar os contextos das tarefas, conhecido como tempo de chaveamento de tarefas.

Um RTOS apresenta um conjunto de funções que compõe a sua API (*Application Programming Interface*). Essas funções tornam possíveis, por exemplo, a criação de tarefas e a gestão de recursos do RTOS.

O tamanho do RTOS também é um fator a ser levado em consideração. Como os RTOS em análise executam em microcontroladores, que possuem pouca memória disponível, um RTOS que extrapolasse tais limites de memória acabariam utilizando espaços de memória que seriam destinados às tarefas que executariam sobre o mesmo.

O escalonador de tarefas é o coração de um RTOS porque é ele quem coordena o processo de troca de tarefas e a escolha de quais processos irão executar no processador, bem como em qual ordem cada tarefa irá executar suas atividades.

Diversos algoritmos de escalonamento podem ser utilizados em um RTOS. Os mais conhecidos são: FIFO, alternância circular (*round-robin*) e prioridade [4]. Existem, entretanto, vários outros algoritmos que podem ser aplicados em um RTOS. Em [6], são apresentados alguns deles, bem como técnicas para implementação dos mesmos.

### B. RTOS para microcontroladores comerciais

Atualmente o mercado oferece diversas soluções de RTOS para microcontroladores. Como a arquitetura BIP é baseada na arquitetura do microcontrolador PIC16 da Microchip [7], este trabalho realizou o comparativo entre RTOS desenvolvidos para esse microcontrolador e para microcontroladores semelhantes, como o PIC18. Dentre os RTOS estudados, destacam-se o OSA [8], o BRTOS [9] e o PICOS18 [10], cujos atributos estão resumidos na Tabela I, considerando o uso desses sistemas operacionais no PIC18. Como esse microcontrolador possui palavra de instrução de 16 bits, ressalta-se que um código de 1KB equivale a 512 instruções.

TABELA I. RTOS para o microcontrolador PIC18

| **Atributo** | **OSA**(1) | **BRTOS** | **PICOS18** |
|---|---|---|---|
| Tempo médio do chaveamento de tarefas (ciclos) | *N*×88(2) | *Informação não disponível* | 980 |
| Número de funções na API | 126 | 36 | 35 |
| Tamanho do código | 432 B a 1 KB | 2 KB a 8 KB | Até 5 KB |
| Memória de dados | *N*× 4 bytes | 100 B a 1 KB | 256 B |
| Algoritmo de escalonamento | Preemptivo por prioridades | Preemptivo por prioridades | Preemptivo por prioridade |
| Tarefas simultâneas | Até 512 tarefas | Até 32 tarefas | Até 8 tarefas |
| Linguagem de implementação | C | C / Assembly | C / Assembly |

Notas: (1) PIC18, Compilador CCS PICC, modo de prioridade Normal
(2) *N* : número de tarefas ativas

Após uma análise detalhada desses sistemas, concluiu-se que o PICOS18 era o mais apropriado para ser portado para o µBIP. Isso se justifica pelo fato do PICOS18 ser o RTOS de código-fonte mais legível dentre os sistemas analisados.

## III. Arquitetura do µBIP

A arquitetura BIP foi projetada visando facilitar o aprendizado de conceitos básicos de arquitetura e organização de computadores, desde a programação na linguagem de montagem ao projeto do processador. Por isso, sua arquitetura é altamente regular, o que simplifica sua organização [11] e facilita a sua implementação.

O BIP utiliza uma arquitetura orientada a acumulador, similar à dos microcontroladores PIC. Porém todas as suas instruções são baseadas em um único formato de instrução de 16 bits.

Os diferentes modelos da família BIP constituem em extensões arquiteturais dos modelos anteriores. O BIP I [12] oferece apenas instruções de aritmética e de transferência. O BIP II [7] acrescenta instruções de desvio condicional e incondicional, enquanto que o BIP III [13] adiciona operações de lógica bit-a-bit. Já o BIP IV [13] inclui suporte à chamada de procedimentos e à manipulação de vetores. Por fim, o µBIP [2] oferece suporte ao tratamento interrupções oriundas de periféricos (ex. Temporizador) e de dispositivos de entrada e saída.

A Tabela II, a seguir, apresenta um resumo dos atributos arquiteturais do µBIP. Ele é o modelo mais completo da família BIP e, por isso, foi selecionado para uso neste trabalho.

Para suportar um RTOS multitarefa, um processador precisa dispor de mecanismos para controle do tempo de execução das tarefas (ou seja, um temporizador) e para troca de tarefas (chaveamento de contexto). Ou seja, é necessário haver recursos de hardware para escalonar a execução das tarefas.

TABELA II. RESUMO DA ARQUITETURA DO µBIP

| Atributo | Descrição |
|---|---|
| Palavra de dados | 16 bits |
| Tipos de dados | Inteiro de 16 bits com sinal –32768 a +32767 |
| Formato de instrução | 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0<br>Cód. Operação \| Operando |
| Modos de endereçamento | Direto: o Operando é um endereço da memória<br>Imediato: o Operando é uma constante<br>Indireto: oOperando é um endereço base de um vetor que é somado ao INDR para o cálculo de um endereçoefetivo da memória de dados |
| Registradores | ACC: acumulador<br>PC: contador de programa<br>STATUS: registrador de Status<br>INDR: registrador de índice<br>SP: apontador do topo da pilha |
| Conjunto de instruções | Controle: HLT<br>Armazenamento: STO<br>Carga: LD e LDI<br>Aritmética: ADD, ADDI, SUB e SUBI<br>Desvio: BEQ, BNE, BGT, BGE, BLT, BLE e JMP<br>Lógica booleana: AND, OR, XOR, ANDI, ORI, XORI e NOT<br>Deslocamento lógico : SLL e SRL<br>Manipulação de vetor: LDV e STOV<br>Suporte a procedimentos : RETURN e CALL<br>Suporte a interrupções: RETINT |

O µBIP possui um temporizador e recursos para tratamento da interrupção gerada por esse periférico. Quando o temporizador envia uma interrupção ao processador, a instrução em execução é concluída, o endereço da instrução seguinte é salvo em uma pilha em hardware e o fluxo de execução é desviado para a rotina de atendimento da interrupção. Ao concluir essa rotina, o contador de programa (registrador PC) é restaurado e retorna para o endereço seguinte ao da instrução que estava sendo executada quando o temporizador gerou a interrupção. Esse recurso é suficiente para viabilizar a temporização que define as fatias de tempo a serem alocadas pelo escalonador às tarefas da aplicação.

Para o escalonador realizar o chaveamento de tarefas, é necessário dispor de recursos para salvar e restaurar o contexto de cada tarefa, o qual é constituído pelo estado dos registradores do processador (ACC, PC, STATUS, INDR e SP) e dos seus periféricos. O µBIP possui uma única pilha dedicada ao salvamento e restauração do PC durante uma chamada de procedimento e não dispõe de recursos para salvamento e recuperação dos demais registradores que compõem o contexto de uma tarefa.

A fim de contornar essa limitação, foram adicionadas três instruções ao conjunto de instruções do µBIP: PUSH, POP e JR, resumidas na Tabela III. A instrução PUSH copia o conteúdo do acumulador para o topo da pilha (ToS – *Top of Stack*) e incrementa em 1 o apontador do topo da pilha (SP – *Stack Pointer*), enquanto que a instrução POP faz a operação contrária, decrementando o SP no processo. Essas duas instruções permitem o salvamento e a recuperação do contexto (registradores) de uma tarefa. Porém, como elas são restritas a operações entre o acumulador e a pilha, qualquer outro registrador deve ser copiado para o acumulador com uma instrução de carga (LD) antes de ser armazenado na pilha com uma operação de PUSH. De forma similar, ao recuperar o contexto da tarefa, uma operação de POP deve ser seguida de uma instrução de armazenamento (STO) para copiar o conteúdo do acumulador para o registrador cujo conteúdo foi restaurado da pilha. Essas restrições são decorrentes de limitações da arquitetura do BIP que visam assegurar a regularidade do conjunto de instruções e a simplicidade da implementação do processador.

TABELA III. INSTRUÇÕES ADICONADAS AO µBIP

| Instrução | Operação |
|---|---|
| `PUSH` | `ToS ← ACC ; SP ← SP + 1` |
| `POP` | `ACC ← ToS ; SP ← SP - 1` |
| `JR` | `PC ← DM[operand]` |

Onde: ToS (Top of Stack): topo da pilha
`DM[operand]`: posição do espaço de endereçamento de dados
`SP (Stack Pointer)`: apontador da pilha de instrução

As instruções de desvio do BIP utilizam um endereço absoluto definido pelo operando imediato da instrução. Entretanto, para realizar o chaveamento de contexto, é preciso realizar o desvio para uma posição que varia conforme a tarefa a ser executada. Essa operação é suportada pela instrução JR (*Jump Register*), a qual permite carregar o conteúdo de uma posição de memória de dados (ex. endereço de uma tarefa) no contador de programa.

## IV. ARQUITETURA DO BIP/OS

A partir da análise do código do PICOS18 e das alterações realizadas no microcontrolador µBIP, foi realizado o projeto de um RTOS mínimo para o µBIP com suporte à execução de até 16 tarefas concorrentes. O sistema operacional criado foi batizado de BIP/OS e as subseções a seguir descrevem a sua arquitetura.

### *A. Organização da Memória*

A memória de programa do µBIP é dividida em duas partes, com 1024 endereços reservados para o sistema operacional e 1024 endereços disponibilizados para o código do usuário (aplicação).

### *B. Contexto da tarefa*

O contexto de uma tarefa no BIP/OS é composto pelos registradores de uso específico do processador e de seus periféricos e por informações sobre o estado da tarefa. Esse contexto é ilustrado na Fig. 1, a seguir.

### *C. API*

A API do BIP/OS possui sete funções relacionadas ao gerenciamento da execução das tarefas da aplicação, as quais são descritas na Tabela IV.

O BIP/OS possui ainda duas funções para escrita nos dispositivos de entrada e saída que impedem que ocorra uma interrupção enquanto o sistema está escrevendo nos mesmos. Além dessas, ele possui uma função para ordenação da tabela de tarefas e uma função para retornar o estado do registrador STATUS, totalizando 11 funções na API.

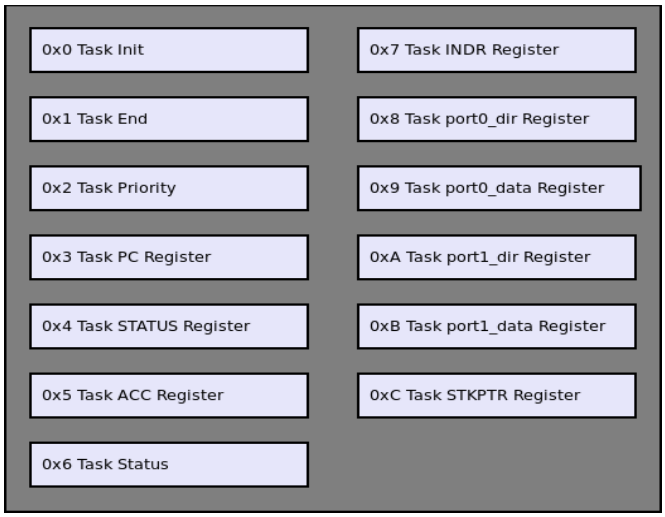


Fig. 1. Contexto de uma tarefa no BIP/OS

TABELA IV. API DO BIP/OS

| ***Função*** | ***Funcionalidade*** |
|---|---|
| OS_TSK_CREATE | Cria uma tarefa, define um identificador único para a mesma e cadastra a tarefa na tabela de tarefas |
| OS_TSK_PAUSE | Salva os valores dos registradores especiais na área de contexto da tarefa, além de salvar os dados da pilha de procedimentos na área de contexto da pilha da tarefa |
| OS_TSK_RETURN | Retorna os dados armazenados na área de contexto para os respectivos registradores, além de retornar o estado da pilha de procedimentos |
| OS_TSK_END | Encerra a execução da tarefa |
| OS_TSK_REMOVE | Remove a tarefa da tabela de tarefas |
| OS_END | Encerra a execução do BIP/OS |
| SCHEDULER | Realiza o escalonamento das tarefas do BIP/OS |
| SET_STATUS | Atualiza o estado do registrador STATUS |
| BUBBLE_SORT | Ordena a lista de tarefas |
| OS_WRITE_PORT0 | Desabilita interrupções para escrever na porta PORT0 |
| OS_WRITE_PORT1 | Desabilita interrupções para escrever na porta PORT1 |

### *D. Escalonador de tarefas*

O escalonador de tarefas do BIP/OS é um algoritmo que realiza um escalonamento do tipo *round-robin*, usando como base uma tabela de tarefas ordenada por ordem de prioridade e, em seguida, por ordem de chegada ao processador.

A Fig. 2 mostra um fluxograma do funcionamento do escalonador do BIP/OS. Ao iniciar suas atividades, o escalonador desliga a chave que permite interrupções do microcontrolador. Em seguida, ele verifica se a tarefa está reiniciando naquele momento ou se foi recém-criada. No primeiro caso, o escalonador paralisa a tarefa que está em execução e verifica qual é a tarefa seguinte na lista de tarefas prontas para serem executadas, carregando o próximo índice da tabela de tarefas. No segundo caso, o escalonador simplesmente escolhe a tarefa seguinte da lista de tarefas e inicia a mesma.

O BIP/OS mantém uma variável global que conta quantas tarefas estão em execução. Caso esse contador seja igual à zero, todas as tarefas do sistema já foram executadas e o BIP/OS pode encerrar a sua execução. No caso de ainda existir alguma tarefa pendente, o escalonador chama a próxima a ser executada.

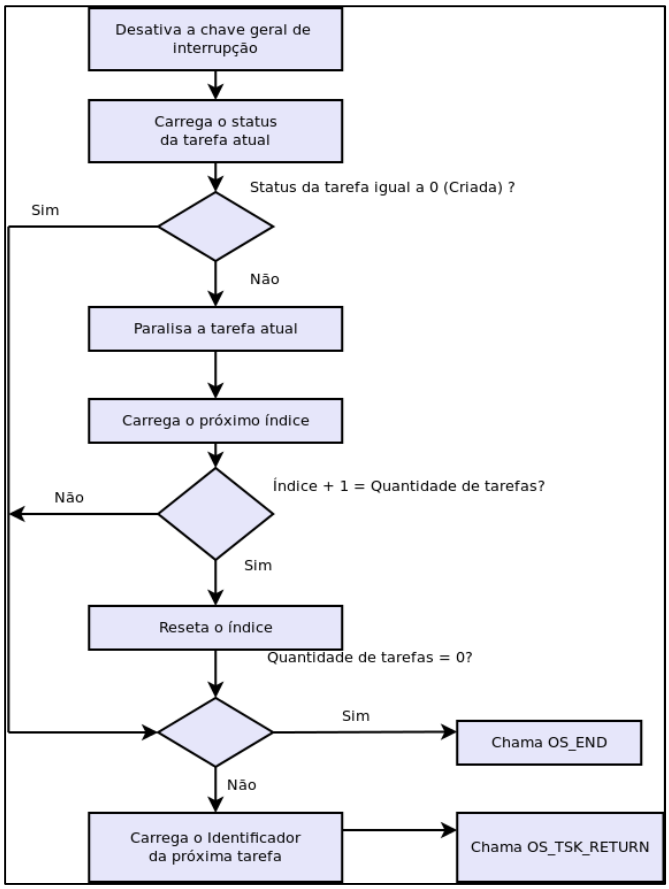


Fig. 2. Fluxograma descrevendo o funcionamento do escalonador do BIP/OS

Para verificar qual é a próxima tarefa a executar, o BIP/OS utiliza a tabela de tarefas, que é uma tabela ordenada de forma crescente, contendo o valor resultante da concatenação da prioridade da tarefa (que é estática, ou seja, não pode ser modificada durante a execução do RTOS) e do identificador da tarefa. A Tabela V exemplifica uma tabela de tarefas ordenada de acordo com a prioridade e a ordem de chegada (criação) das tarefas. Quanto menor o valor de prioridade, maior é o nível de prioridade da tarefa. No exemplo, a primeira tarefa da tabela é a de identificador 2 e possui a prioridade mais alta (1). As duas tarefas seguintes tem o mesmo nível de prioridade (2), mas a tarefa 1 aparece primeiro porque foi criada antes da tarefa 3.

TABELA V. EXEMPLO DE ORGANIZAÇÃO DA TABELA DE TAREFAS

| ***Prioridade da tarefa*** | ***Identificador da tarefa*** | ***Tabela de tarefas*** |
|---|---|---|
| 1 | 2 | 0x12 |
| 2 | 1 | 0x21 |
| 2 | 3 | 0x23 |

A tabela de tarefas é reordenada sempre que uma tarefa é criada (quando ocorre uma chamada à OS_TSK_CREATE) ou encerrada (quando ocorre uma chamada à OS_TSK_REMOVE).

O escalonador é chamado em três situações diferentes:

1. Na inicialização do sistema, quando o escalonador é chamado para iniciar a primeira tarefa da lista logo após a criação das tarefas;

2. Na ocorrência de uma interrupção do temporizador indicando o fim da fatia de tempo da tarefa em execução (essa é a situação mais comum para a chamada do escalonador); e

3. Na finalização de uma tarefa, quando então a função de encerramento desvia o escalonador para que ele chame a próxima tarefa cadastrada na tabela de tarefas.

O encerramento de uma tarefa ocorre quando a mesma chama a função OS_TSK_END. A tarefa não é finalizada automaticamente, sendo sempre necessário realizar a chamada desta função. Neste caso, a função OS_TSK_END funciona como a função OS_Yield() do OSA [8].

## V. IMPLEMENTAÇÃO

O BIP/OS foi codificado na linguagem de montagem do µBIP. A validação das suas funcionalidades foi realizada utilizando-se um modelo SystemC do µBIP gerado a partir da descrição em ADL (*Architecture Description Language*) do microcontrolador por meio da ferramenta ArchC [14]. Esse modelo ADL/SystemC do µBIP foi modificado para incluir as três instruções propostas neste trabalho (PUSH, POP e JR). Cabe ressaltar que a partir do modelo ADL, o ArchC também gera um montador e um ligador, os quais foram usados no desenvolvimento do BIP/OS.

O trecho de código-fonte na Fig. 3 ilustra uma parte da função OS_TSK_END do BIP/OS, a qual é responsável por encerrar a tarefa.

É possível ver no código-fonte a utilização das estruturas de dados do BIP/OS, representadas pelas instruções `LDV 0x5B0`, que carrega o valor contido no endereço `DM[0x5B0 + $indr]`, que neste caso, é a tarefa em execução, e pela instrução `STOV 0x0000`, que irá salvar o valor 3 no endereço representado por `DM[0x0000 + 0x7T6]`, onde `T` é o identificador único da tarefa.

## VI. RESULTADOS

A implementação do BIP/OS foi verificada por meio de testes de chaveamento de contexto e escalonamento por prioridades, os quais confirmaram o correto funcionamento do sistema. Após os testes, foram tomadas medidas relativas ao custo e ao desempenho do BIP/OS, os quais são resumidos na Tabela VI (que serve de referência para comparação com os sistemas operacionais descritos na Tabela I).

TABELA VI. RTOS PARA MICROCONTROLADORES PIC

| **Atributo** | **BIP/OS** |
|---|---|
| Tempo médio do chaveamento de tarefas | 260 ciclos |
| Número de funções na API | 11 |
| Tamanho do código | 1,08 KB |
| Memória de dados | 282 B |
| Algoritmo de escalonamento | Round-robin por prioridades |
| Tarefas simultâneas | Até 16 |
| Linguagem de implementação | Assembly |

Em comparação com o OSA, o BIP/OS ocupa mais espaço na memória mesmo tendo muito menos funções na API. Isso se dá pelo fato de o código-fonte do OSA ser escrito em C. No processo de compilação do OSA o compilador realiza diversas otimizações, o que minimiza o seu tamanho, deixando-o, entretanto, ilegível.

O escalonador do BIP/OS é baseado no escalonador do PICOS18. No BIP/OS algumas estruturas e verificações foram simplificadas, mas o funcionamento do escalonador destes dois sistemas é muito semelhante.

O tempo médio de chaveamento entre tarefas é relativamente pequeno, dependendo da fatia de tempo disponibilizada para cada tarefa.

O BIP/OS pode ser considerado um *soft* RTOS, visto que o tempo de execução das tarefas é conhecido, mas não configurável. A fatia de tempo disponibilizada para cada tarefa é uma fatia fixa, igual para todas as tarefas que executarão no BIP/OS.

Em comparação com os demais RTOS estudados para a criação do BIP/OS, o BIP/OS, em comparação com o OSA, é mais rápido quando o sistema executa mais do que 3 tarefas e 3,4 vezes mais rápido que o PICOS18 devido às simplificações feitas nas estruturas de dados do BIP/OS. Estes dados foram obtidos após simulações realizadas com o BIP/OS, comparadas às informações de tempo e desempenho disponibilizadas pelos desenvolvedores do OSA [8] e do PICOS18 [10].

Com relação ao número de tarefas, o BIP/OS permite no máximo 16 tarefas devido ao modo como são compostas as estruturas de dados do BIP/OS. Possibilitar a criação de mais do que 16 tarefas implicaria na utilização de mais memória para as estruturas que armazenam o contexto das tarefas.

```
01 OS_TSK_END:
02      LDI 0xFFE              # Desativa a chave de interrupção
03      AND $int_config        # zerando o ultimo bit
04      STO $int_config        # do registrador de configuração
05      LD current_tsk_ind     # Carrega o índice da tabela de tarefas
06      STO $indr              # Armazena no registrador índice
07      LDV 0x05B0             # Carrega o valor contido na tabela
08      ANDI 0x0007            # Obtém o identificador da tarefa
09      SLL 0x4                # Desloca em 4 posições o id da tarefa
10      ADDI 0x706             # Adiciona 0x706 p/ formar o endereço do status da tarefa (0x7T6)
11      STO $indr              # Armazena o endereço no registrador índice
12      LDI 0x3                # Carrega o status 3 = Tarefa encerrada
13      STOV 0x0000            # Salva o status na estrutura de dados
```

Fig. 3. Exemplo de função da API: a função de encerramento de tarefa (`OS_TSK_END`)

## VII. CONCLUSÕES

O BIP/OS é um sistema operacional básico que pode ser utilizado para explicar conceitos como chaveamento de contexto e escalonamento para alunos de disciplinas sobre Sistemas Operacionais. Assim como a arquitetura BIP, o BIP/OS é um sistema operacional voltado para o ensino, não se preocupando com questões como o desempenho do sistema. Logo, não foram realizadas otimizações de código-fonte, o que poderia, em alguns casos, dificultar a aprendizagem.

Diversos trabalhos de disciplina poderiam ser propostos para desenvolvimento utilizando o BIP/OS, como a criação de tarefas que executariam paralelamente e a adição de novos dispositivos e funcionalidades.

Como trabalho futuro, pretende-se avaliar o impacto do uso do BIP/OS no aprendizado dos conceitos abordados na disciplina "Sistemas Operacionais". Além disso, pretende-se realizar melhorias no algoritmo de escalonamento, implementar novos modos de tratamento para o encerramento de tarefas e avaliar a viabilidade de acrescentar o suporte à comunicação entre tarefas. Também considera-se como trabalho futuro tornar a fatia de tempo configurável individualmente para cada tarefa.

## VIII. REFERÊNCIAS


[1] C. MAZIERO, "Reflexões sobre o Ensino Prático de Sistemas Operacionais",*in* Anais do X Workshop sobre Educação em Computação da SBC, 2002. v. 1. p. 1-12.

[2] M. C. Pereira, C. A. Zeferino, "uBIP: a Simplified Microcontroller Architecture for Education in Embedded Systems Design",*in*Proc. of IP Based Electronic System Conference & Exhibition,Grenoble, Design and Reuse, 2008. p. 193-197.

[3] W. Stallings, Operating Systems: Internals and Design Principles. 7. ed. New Jersey: Prentice Hall, 2012. p. 430-473.

[4] H. M. Deitel, P. J. Deitel, D. R. Choffnes, Sistemas Operacionais. 3.ed. São Paulo, Prentice Hall, 2005, p. 225.

[5] A. C. Shaw, Sistemas e Software de Tempo Real, Porto Alegre: Bookman, 2001. p. 236.

[6] H. Marcondes, R. Cancian, M. Stemmer, A. A. Fröhlich, "Modelagem e Implementação de Escalonadores de Tempo Real para Sistemas Embarcados",*in*Anais do VI Workshop de Sistemas Operacionais. Bento Gonçalves: SBC, 2009, p. 2405-2416.

[7] D. Morandi, A. L. A. Raabe, C. A. Zeferino, "Processadores para Ensino de Conceitos Básicos de Arquitetura de Computadores",*in* Anais do 1ºWorkshop De Educação em Arquitetura de Computadores, Porto Alegre, SBC, 2006, p 17-24.

[8] OSA, OSA Documentation, 2008. Disponível em: <http://www.pic24.ru/doku.php/en/osa/ref/intro>. Acesso em: 15 out. 2012.

[9] BRTOS, Manual de referência do BRTOS, Versão 1.7x, Outubro de 2012. Disponível em: <http://brtos.googlecode.com/files/Manual%20de%20refer%C3%AAncia%20- %20BRTOS%201.7x.pdf>. Acesso em: 05 out. 2012.

[10] PRagmatec, PICos18: Real Time Kernel for PIC18,2006. Disponível em: <http://www.pragmatec.net/Download/PICos18/1.%20PICos/API/PICOS18_API_us.pdf>. Acesso em: 07 out.2012.

[11] D. A. Patterson, J. L. Hennessy, Organização e Projeto de Computadores: a Interface Hardware/Software. 2. ed. Rio de Janeiro, LTC, 2000.

[12] D. Morandi, M.C. Pereira, A. L. A. Raabe, C. A. Zeferino, "Um Processador Básico para o Ensino de Conceitos de Arquitetura e Organização de Computadores",Hífen, Uruguaiana, 2006, v. 30, p. 73-80.

[13] P. V. Vieira, P. R. M. Rech, R. C. Mensch, A. L. A. Raabe, C. A. Zeferino, "Estendendo a Arquitetura dos Processadores BIP para Ampliar o seu Potencial de Uso em Disciplinas de Introdução a Programação",International Journal of Computer Architecture Education, v. 1, p. 1-10, 2012.

[14] The ArchC Team, The ArchC Project Home Page. Disponível em: <http://www.archc.org >. Acesso em: 15 out 2012.