

## Trabalho Prático 1 - Montador

### 1. Descrição Geral

O principal objetivo deste trabalho é implementar um montador para a máquina *Swombat*, a ser executada no simulador CPUSim, ambos disponibilizados com este documento.

O CPUSim é desenvolvido em linguagem Java, e é capaz de simular uma máquina completa. Ele recebe como entrada um arquivo contendo os dados em linguagem de máquina, o qual é carregado na memória RAM da máquina *Swombat*, deixando-o pronto para execução.

Mais informações sobre o CPUSim podem ser encontradas na página oficial do simulador, em <http://www.cs.colby.edu/djskrien/CPUSim/>.

### 2. Informações Importantes

- O trabalho deve ser feito em duplas ou trios, podendo ser discutido entre os colegas desde que não haja cópia ou compartilhamento do código fonte.
- A data de entrega será especificada através de uma tarefa no Moodle.
- Os trabalhos poderão ser entregues até às 23:55 do dia especificado para a entrega. O horário de entrega deve respeitar o relógio do sistema Moodle. Haverá uma tolerância de 5 minutos de atraso, de forma que os alunos podem fazer a entrega até às 0:00. A partir desse horário, os trabalhos já estarão sujeitos a penalidades. A fórmula para desconto por atraso na entrega do trabalho prático é:

$$\text{Desconto} = 2^d / 0.32 \%$$

onde  $d$  é o atraso em dias úteis. Note que após 5 dias úteis, o trabalho não pode ser mais entregue.

- O trabalho deve ser implementado obrigatoriamente na linguagem C ou C++.
- Deverá ser entregue o código fonte com os arquivos de dados necessários para a execução e um arquivo *Makefile* que permita a compilação do programa nas máquinas Linux do DCC.
- Além disso, deverá ser entregue uma pequena documentação contendo todas as decisões de projeto que foram tomadas durante a implementação, sobre aspectos não contemplados na especificação, assim como uma justificativa para essas decisões. Esse documento não precisa ser extenso (mínimo 3 e máximo de 6 páginas). A documentação deve indicar o nome dos alunos integrantes do grupo.

- A ênfase do trabalho está no funcionamento do sistema e não em aspectos de programação ou interface com o usuário. Assim, não deverá haver tratamento de erros no programa de entrada.
- Todas as dúvidas referentes ao trabalho serão esclarecidas por meio do fórum disponível no ambiente Moodle da disciplina.
- A entrega do trabalho deverá ser realizada pelo Moodle, na tarefa criada especificamente para tal. A entrega deverá ser feita no seguinte formato:
  - O trabalho a ser entregue deverá estar contido em um único arquivo compactado, em formato “.zip”, com o nome no formato “*tp1\_aluno1\_aluno2\_aluno3.zip*”
  - O arquivo .zip definido deverá ter três pastas:
    - “*assembler*”: Essa pasta deverá conter o código-fonte do montador implementado, juntamente do arquivo *Makefile* (OBS.: Não devem ser incluídos arquivos .o nem executáveis nessa pasta.)
    - “*tst*”: Essa pasta irá conter os arquivos “.a” de entrada desenvolvidos para testar o montador. Mais detalhes desse arquivo estão definidos na Seção 5.
    - “*doc*”: Essa pasta deverá conter o arquivo da documentação, em formato PDF. Caso o grupo julgue necessário incluir quaisquer outros arquivos a parte, esses deverão ser justificados em um arquivo texto com o nome README.
  - **Atenção:** Trabalhos que descumprirem o padrão definido acima serão penalizados.

### 3. Especificação da Máquina *Swombat*

A máquina a ser utilizada é a *Swombat*, projetada para uso no simulador CPUSim. Seguem as especificações:

- A menor unidade endereçável nessa máquina é uma palavra de 16 bits (um inteiro).
- Os tipos de dados tratados pela máquina são somente inteiros.
- A máquina possui uma memória principal de 256 posições, numeradas de 0 a 255.
- O endereço 254 (última palavra de 16 bits) é um endereço especial para E/S (Entrada e Saída). Quaisquer valores armazenados nesse endereço (através da instrução *storei*, descrita no Anexo 1 deste documento) serão impressos na tela. Por outro lado, toda instrução de carregamento nesse endereço (*loadi*, também descrita no Anexo 1) será interpretada como uma solicitação de dados ao usuário, o qual deverá digitar um valor inteiro através de um console do CPUSim. O registrador de destino definido na instrução deverá receber o valor informado.
- A memória é dividida em células de 8 bits. Dessa forma, uma palavra de 16 bits situada no endereço de memória X será seguida por outra palavra no endereço X+2, e assim por diante.
- Existe também uma pilha (127 elementos de 8 bits) acessada por instruções específicas (*push* e *pop*)
- Cada registrador pode armazenar uma palavra de 16 bits.
- Existem 8 registradores de uso geral, os quais são nomeados de “R0” a “R7”.
- Os registradores de propósito específico são:
  - **stackpt** (Stack Pointer): O ponteiro de pilhas aponta para o topo da pilha (inicializado com 0).
  - **sdr**: Armazena um dado recém-lido ou a ser escrito na pilha.
  - **pc**: (Program Counter): Armazena o contador de programa, ou seja, o endereço de memória da próxima instrução a ser executada. O *pc* inicia a execução do programa

na posição 0 (zero) da memória e é automaticamente incrementado a cada ciclo de instrução de forma que as instruções são normalmente executadas sequencialmente a partir da memória. O *pc* é afetado também pelas instruções de desvio e chamadas de procedimentos.

- **buffer1** e **buffer2**: Registradores que recebem operandos a serem tratados, como por exemplo, os operandos da instrução *add*.
- **ir**: Armazena a instrução em execução. É a partir desse registrador que a instrução é decodificada.
- **mar**: Armazena um endereço a ser lido ou escrito na memória.
- **mdr**: Armazena um dado recém-lido (por exemplo, uma instrução recém-carregada) ou a ser escrito na memória.
- **status**: Armazena uma flag chamada *halt-bit* que, se escrita, interrompe o programa imediatamente, gerando uma mensagem de encerramento na tela.
- Cada instrução pode ter 0, 1 ou 2 operandos, e sempre possui 16 bits de tamanho. Algumas instruções são especificadas com campos não utilizados, e portanto, devem ser escritos 0's (zeros) nessa região. O Anexo 1 deste documento mostra a ordem dos operandos (incluindo os campos não utilizados) em cada instrução de máquina, bem como o número de bits de cada operando. Essa ordem deve ser respeitada pelo montador, caso contrário, o programa não funcionará corretamente.

## 4. Especificação do Montador

- O montador a ser implementado é um montador de 2 passos, conforme descrito no capítulo 7 do livro utilizado na disciplina.
- O conjunto de instruções é o especificado pela máquina *Swombat*, e está descrito em mais detalhes no Anexo 1 deste documento.
- Cada linha da linguagem de montagem da máquina *Swombat* possui o seguinte formato:
  - `_[rótulo:] operador [operando(s)] [:comentário]`

Ou seja:

- Se houver algum rótulo (*label*), ele será definido no início da linha. Todo rótulo deverá iniciar com um *underscore* (“\_”) e finalizar com dois-pontos (“:”).

Exemplo: `_label1: add r0 r1 r2 ;soma de dois números`

- A presença do operador é obrigatória, pois o mesmo identifica a instrução de máquina a ser executada.
- A presença ou não de operandos depende da instrução, tendo em vista que o número de operandos varia de instrução para instrução. Se houverem dois operandos, estes serão separados por espaços (“ ”).
- Um comentário pode ser incluído opcionalmente no fim da linha, devendo necessariamente começar por um ponto-e-vírgula (;) e devendo ser ignorado pelo montador.
- O endereço de memória indicado nos operandos das instruções, inclusive nas de desvio, é a posição absoluta da memória, ou seja, nenhum pré-tratamento deve ser feito sobre esse endereço.
- Rótulo, operador, operandos e comentário deverão ser separados por espaços (“ ”), assim como dois operandos mencionados anteriormente. Poderá haver mais de um espaço, o que não deve afetar o funcionamento do montador.

- Não deve haver linhas vazias e linhas contendo apenas comentários ou rótulos.
- A pseudo-instrução *.data* (inclui ponto no nome) deverá ser tratada pelo montador, e sua função será a de reservar uma posição da memória da máquina *Swombat*.
  - Formato: *label: .data num\_bytes valor\_inicial*
  - A instrução servirá para alocar uma região de memória de tamanho '*num\_bytes*', com "*valor\_inicial*". Essa região de memória será identificada pelo rótulo '*label*', o qual pode ser usado ao longo do código Assembly para acessar aquela região de memória.
  - A política de alocação de variáveis na memória deverá ser definida pelo montador. A documentação deve explicar e justificar como foi feita a alocação (por exemplo o início da memória de dados, se a alocação cresce para cima ou para baixo, entre outras definições).
- **IMPORTANTE:** o programa a ser carregado na máquina deverá sempre iniciar na posição 0 (zero) de memória, uma vez que o contador de programa (PC) da máquina *Swombat* é inicializado por padrão com este valor.

## 5. Formato de Entrada de Dados no Montador

O programa a ser traduzido pelo montador deverá ser escrito em um arquivo texto simples com formato *“.a”*, sendo que as instruções devem ser dispostas uma por linha no arquivo, e deverão ser lidas pelo montador. Um arquivo exemplo será disponibilizado no Moodle. **Atenção:** O arquivo de exemplo não deve ser utilizado como teste oficial do montador implementado.

Deverão ser escritos ao menos **dois** programas em Assembly que, juntos, executem ao menos **dois terços** das instruções da máquina *Swombat* e ao menos uma chamada de função ou procedimento.

A implementação desses programas de teste será avaliada e, portanto, os seus códigos não devem ser compartilhados entre os grupos. Como mencionado na Seção 2, os arquivos de teste *“.a”* deverão ser incluídos no diretório *“tst”* do arquivo de entrega do trabalho, e devidamente explicados na documentação (o que se propõe fazer, qual o resultado esperado e outras informações que possam ser relevantes).

## 6. Formato de Saída de Dados do Montador

O grupo deverá optar por um dos formatos de saída abaixo, os quais são suportados pelo CPUSim:

1. Saída em formato *“.hex”* (Intel HEX): Para a máquina *Swombat*, este será um arquivo texto que irá separar as instruções de máquina byte-a-byte, uma vez que cada célula de memória possui um byte. Em outras palavras, cada linha do arquivo nesse formato define uma célula de memória que possuirá um dos dois bytes de uma instrução, em formato hexadecimal. Além desse dado, outras informações são definidas segundo o padrão Intel HEX, como o tipo do dado, o número de bytes da linha (nesse caso, sempre 1 byte), bem como o *checksum* (soma de verificação). Mais informações de como esse arquivo deve ser escrito podem ser encontrados no site da Wikipédia em [https://en.wikipedia.org/wiki/Intel\\_HEX#Format](https://en.wikipedia.org/wiki/Intel_HEX#Format)

2. Saída em formato *“.mif”*: Arquivo texto que define cada célula de memória (1 byte no caso da máquina *Swombat*), em formato binário (Caracteres 1 e 0 em ASCII).

Como forma de auxiliar os grupos na escolha do formato mais adequado, bem como ter uma referência na hora de implementar o montador, os grupos podem utilizar o montador interno do CPUSim para gerar as instruções de máquina em um dos formatos acima. Para isso, execute os seguintes passos:

1. Com o CPUSim aberto, vá em *File->Open Machine...* Selecione a máquina *Swombat*.
2. Vá em *File->Open Text...* e abra o arquivo “.a” desejado (faça isso inicialmente com o arquivo exemplo fornecido).
3. Vá em *Execute->Assemble & load*. Esse comando irá executar o montador interno do CPUSim e irá carregar o programa para a memória RAM, situada na direita do simulador. (OBS: Serão apresentadas mensagens de erro caso o CPUSim encontre problemas no código Assembly fornecido, e consequentemente o mesmo não será carregado para a memória RAM.)
4. Por fim, vá em *File->Save RAM->from Main...* Escolha o formato (“.hex” ou “.mif”) e escolha a pasta de destino. **Atenção:** apenas a memória principal (*Main*) será avaliada, já que a pilha estará vazia na inicialização de qualquer programa.
5. O arquivo gerado pode ser aberto em qualquer editor de texto. (OBS: O formato do arquivo gerado pelo grupo deve ser o mesmo que o gerado pelo montador do CPUSim. Por isso, o grupo pode utilizar este arquivo como uma forma de orientação na resolução de quaisquer problemas no código fonte do montador.)

## 7. Formato de Chamada do Simulador

A saída do montador (arquivo no formato “.hex” ou “.mif”) deve ser executada na máquina *Swombat* do CPUSim para garantir que o programa Assembly foi traduzido corretamente. Para isso, execute os seguintes passos:

1. Com o CPUSim aberto, vá em *File->Open Machine...* Selecione a máquina *Swombat*.
2. Vá em *File->Open RAM->into Main....* Selecione o arquivo (“.hex” ou “.mif”) gerado pelo montador.
3. Com o arquivo carregado na memória RAM, vá em *Execute->Run*. O programa carregado na memória RAM será executado.

## 8. Sobre a Documentação

- Deve conter todas as decisões de projeto.
- Deve conter informações sobre cada programa testado, sobre o que ele faz, entradas de dados, saída esperada, etc.
- Deve conter elementos que comprovem que o montador foi testado (Ex.: imagens das telas de montagem e execução no CPUSim). Quaisquer arquivos relativos a testes devem ser enviados no pacote do trabalho, como mencionado na Seção 2. A documentação deve conter referências a esses arquivos, explicação do que eles fazem e dos resultados obtidos.
- O código fonte não deve ser incluído no arquivo PDF da documentação.

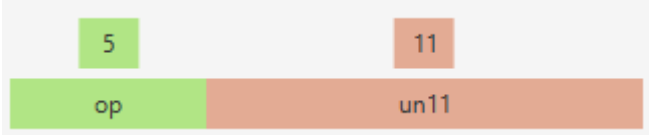
## 9. Considerações Finais

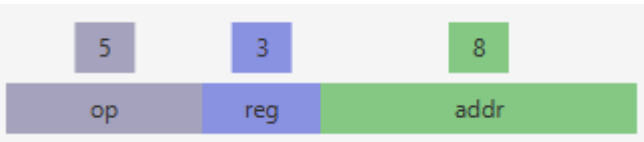
É obrigatório o cumprimento fiel de todas as especificações descritas neste documento. As decisões de projeto devem fazer parte apenas da estrutura interna do montador, não podendo afetar a interface de entrada e saída.

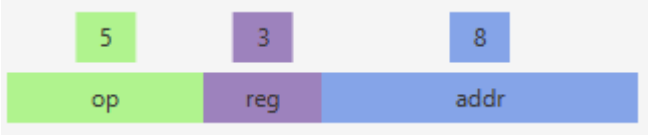
## ANEXO 1 - Conjunto de Instruções da Máquina *Swombat*

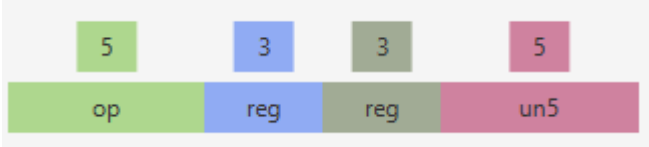
É possível ver uma representação gráfica das instruções dentro do CPUSim. Para tanto, vá em *Modify -> Machine Instructions...* A ordem dos operandos de cada instrução (de cima para baixo) é definida a seguir.

Algumas instruções possuem bits não utilizados, os quais devem ser preenchidos com o valor zero, caso contrário, o programa não funcionará corretamente. A posição desses campos (se houverem) também é definida em cada instrução.

<b>Código da operação (5 bits)</b>	<b>00 - exit</b>
<b>Significado</b>	Encerra o programa, através da escrita da flag <i>halt-bit</i> .
<b>Operandos</b>	<ul style="list-style-type: none"> <li>Bits não utilizados (valor zero) – 11 bits</li> </ul>
<b>Ação</b>	N/A
<b>Exemplo</b>	<ul style="list-style-type: none"> <li>exit</li> <li>label: exit</li> </ul>
<b>Representação gráfica</b>	

<b>Código da operação (5 bits)</b>	<b>01 - loadi</b>
<b>Significado</b>	Carrega um dado da memória para um registrador. (OBS.: Caso o endereço especificado seja o de E/S, ou seja, 254, será solicitado ao usuário que digite um valor.)
<b>Operandos</b>	<ul style="list-style-type: none"> <li>Registrador – 3 bits</li> <li>Endereço de Memória – 8 bits</li> </ul>
<b>Ação</b>	Registrador << Memória[endereço]
<b>Exemplo</b>	<ul style="list-style-type: none"> <li>loadi R0 150</li> <li>loadi R0 var ;‘var’ alocada na memória com a pseudo-instrução <i>.data</i></li> <li>label: loadi R0 var</li> </ul>
<b>Representação gráfica</b>	

<b>Código da operação (5 bits)</b>	<b>02 – storei</b>
<b>Significado</b>	Escreve o valor de um registrador em um endereço de memória. (OBS.: Caso o endereço especificado seja o de E/S, ou seja, 254, o valor contido no Registrador 1 será impresso na tela.)
<b>Operandos</b>	<ul style="list-style-type: none"> <li>• Registrador – 3 bits</li> <li>• Endereço de Memória – 8 bits</li> </ul>
<b>Ação</b>	Registrador >> Memória[endereço]
<b>Exemplo</b>	<ul style="list-style-type: none"> <li>• storei R0 150</li> <li>• storei R0 var ;‘var’ alocada na memória com a pseudo-instrução .data</li> <li>• label: storei R0 var</li> </ul>
<b>Representação gráfica</b>	

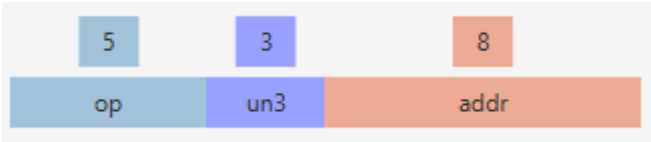
<b>Código da operação (5 bits)</b>	<b>03 - add</b>
<b>Significado</b>	Faz a soma dos valores de dois registradores e escreve o resultado no primeiro.
<b>Operandos</b>	<ul style="list-style-type: none"> <li>• Registrador 1 – 3 bits</li> <li>• Registrador 2 – 3 bits</li> <li>• Bits não utilizados (valor zero) – 5 bits</li> </ul>
<b>Ação</b>	Registrador 1 << Registrador 1 + Registrador 2
<b>Exemplo</b>	<ul style="list-style-type: none"> <li>• add R0 R1</li> <li>• label: add R0 R1</li> </ul>
<b>Representação gráfica</b>	

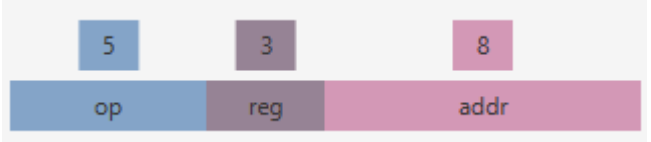
<b>Código da operação (5 bits)</b>	<b>04 - subtract</b>
<b>Significado</b>	Faz a subtração dos valores de dois registradores e escreve o resultado no primeiro.
<b>Operandos</b>	<ul style="list-style-type: none"> <li>• Registrador 1 – 3 bits</li> </ul>



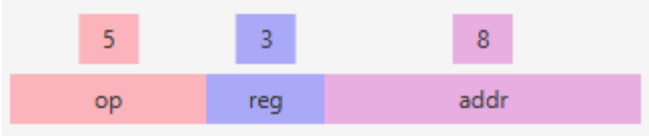
	<ul style="list-style-type: none"> <li>Registrador 2 – 3 bits</li> <li>Bits não utilizados (valor zero) – 5 bits</li> </ul>
<b>Ação</b>	Registrador 1 << Registrador 1 - Registrador 2
<b>Exemplo</b>	<ul style="list-style-type: none"> <li>subtract R0 R1</li> <li>label: subtract R0 R1</li> </ul>
<b>Representação gráfica</b>	

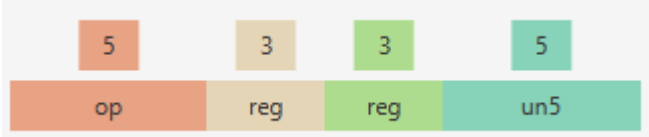
<b>Código da operação (5 bits)</b>	<b>05 - multiply</b>
<b>Significado</b>	<ul style="list-style-type: none"> <li>Faz a multiplicação dos valores de dois registradores e escreve o resultado no primeiro.</li> </ul>
<b>Operandos</b>	<ul style="list-style-type: none"> <li>Registrador 1 – 3 bits</li> <li>Registrador 2 – 3 bits</li> <li>Bits não utilizados (valor zero) – 5 bits</li> </ul>
<b>Ação</b>	Registrador 1 << Registrador 1 * Registrador 2
<b>Exemplo</b>	<ul style="list-style-type: none"> <li>multiply R0 R1</li> <li>label: multiply R0 R1</li> </ul>
<b>Representação gráfica</b>	
<b>Código da operação (5 bits)</b>	<b>06 - divide</b>
<b>Significado</b>	Faz a divisão dos valores de dois registradores e escreve o resultado no primeiro.
<b>Operandos</b>	<ul style="list-style-type: none"> <li>Registrador 1 – 3 bits</li> <li>Registrador 2 – 3 bits</li> <li>Bits não utilizados (valor zero) – 5 bits</li> </ul>
<b>Ação</b>	Registrador 1 << Registrador 1 / Registrador 2
<b>Exemplo</b>	<ul style="list-style-type: none"> <li>divide R0 R1</li> <li>label: divide R0 R1</li> </ul>
<b>Representação gráfica</b>	

<b>Código da operação (5 bits)</b>	<b>07 - jump</b>
<b>Significado</b>	Pula para a instrução contida no endereço de memória especificado.
<b>Operandos</b>	<ul style="list-style-type: none"> <li>• Bits não utilizados (valor zero) – 3 bits</li> <li>• Endereço de Memória – 8 bits</li> </ul>
<b>Ação</b>	pc << endereço
<b>Exemplo</b>	<ul style="list-style-type: none"> <li>• jump 50</li> <li>• jump label</li> <li>• label: jump 50</li> </ul>
<b>Representação gráfica</b>	

<b>Código da operação (5 bits)</b>	<b>08 - jmpz</b>
<b>Significado</b>	Pula para a instrução contida no endereço de memória especificado, caso o valor contido no registrador seja igual a zero.
<b>Operandos</b>	<ul style="list-style-type: none"> <li>• Registrador – 3 bits</li> <li>• Endereço de Memória – 8 bits</li> </ul>
<b>Ação</b>	Se (Registrador = 0) pc << endereço
<b>Exemplo</b>	<ul style="list-style-type: none"> <li>• jmpz R0 50</li> <li>• jmpz R0 label</li> <li>• label: jmpz R0 50</li> </ul>
<b>Representação gráfica</b>	

<b>Código da operação (5 bits)</b>	<b>09 - jmpn</b>
<b>Significado</b>	Pula para a instrução contida no endereço de memória especificado, caso o valor contido no registrador seja menor que zero.

<b>Operandos</b>	<ul style="list-style-type: none"> <li>Registrador – 3 bits</li> <li>Endereço de Memória – 8 bits</li> </ul>
<b>Ação</b>	Se (Registrador < 0) pc << endereço
<b>Exemplo</b>	<ul style="list-style-type: none"> <li>jmpn R0 50</li> <li>jmpn R0 label</li> <li>label: jmpn R0 50</li> </ul>
<b>Representação gráfica</b>	

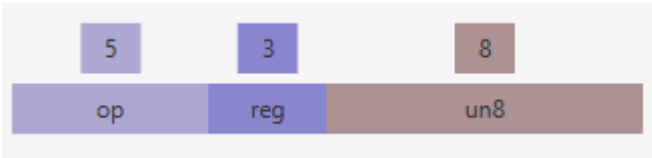
<b>Código da operação (5 bits)</b>	<b>10 - move</b>
<b>Significado</b>	Copia o conteúdo do Registrador 2 para o Registrador 1.
<b>Operandos</b>	<ul style="list-style-type: none"> <li>Registrador 1 – 3 bits</li> <li>Registrador 2 – 3 bits</li> <li>Bits não utilizados (valor zero) – 5 bits</li> </ul>
<b>Ação</b>	Registrador 1 << Registrador 2
<b>Exemplo</b>	<ul style="list-style-type: none"> <li>move R0 R1</li> <li>label: move R0 R1</li> </ul>
<b>Representação gráfica</b>	

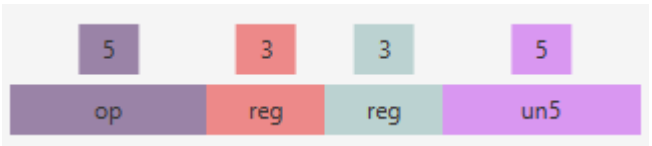
<b>Código da operação (5 bits)</b>	<b>11 - load</b>
<b>Significado</b>	Carrega o endereço de memória contido no Registrador 2 para o Registrador 1. (OBS.: Caso o endereço especificado seja o de E/S, ou seja, 254, será solicitado ao usuário que digite um valor.)
<b>Operandos</b>	<ul style="list-style-type: none"> <li>Registrador 1 – 3 bits</li> <li>Registrador 2 – 3 bits</li> <li>Bits não utilizados (valor zero) – 5 bits</li> </ul>
<b>Ação</b>	Registrador 1 << Memória[Registrador 2]
<b>Exemplo</b>	<ul style="list-style-type: none"> <li>load R0 R1</li> </ul>

	<ul style="list-style-type: none"> <li>label: load R0 R1</li> </ul>
<b>Representação gráfica</b>	

<b>Código da operação (5 bits)</b>	<b>12 - store</b>
<b>Significado</b>	Armazena o valor contido no Registrador 1 no endereço de memória contido no Registrador 2. (OBS.: Caso o endereço especificado seja o de E/S, ou seja, 254, o valor contido no Registrador 1 será impresso na tela.)
<b>Operandos</b>	<ul style="list-style-type: none"> <li>Registrador 1 – 3 bits</li> <li>Registrador 2 – 3 bits</li> <li>Bits não utilizados (valor zero) – 5 bits</li> </ul>
<b>Ação</b>	Registrador 1 >> Memória[Registrador 2]
<b>Exemplo</b>	<ul style="list-style-type: none"> <li>store R0 R1</li> <li>label: store R0 R1</li> </ul>
<b>Representação gráfica</b>	

<b>Código da operação (5 bits)</b>	<b>13 - loadc</b>
<b>Significado</b>	Carrega uma constante de 8 bits (com sinal) em um registrador. A constante é armazenada nos 8 bits menos significativos do registrador.
<b>Operandos</b>	<ul style="list-style-type: none"> <li>Registrador – 3 bits</li> <li>Constante – 8 bits</li> </ul>
<b>Ação</b>	Registrador 1[8-15] << Constante
<b>Exemplo</b>	<ul style="list-style-type: none"> <li>loadc R0 150</li> <li>label: loadc R0 150</li> </ul>
<b>Representação gráfica</b>	

<b>Código da operação (5 bits)</b>	<b>14 - clear</b>
<b>Significado</b>	Zera o valor contido em um registrador.
<b>Operandos</b>	<ul style="list-style-type: none"> <li>• Registrador – 3 bits</li> <li>• Bits não utilizados (valor zero) – 8 bits</li> </ul>
<b>Ação</b>	Registrador $\ll 0$
<b>Exemplo</b>	<ul style="list-style-type: none"> <li>• clear R0</li> <li>• label: clear R0</li> </ul>
<b>Representação gráfica</b>	

<b>Código da operação (5 bits)</b>	<b>15 - negate</b>
<b>Significado</b>	Inverte o sinal do valor de um registrador e armazena o resultado em outro.
<b>Operandos</b>	<ul style="list-style-type: none"> <li>• Registrador 1 – 3 bits</li> <li>• Registrador 2 – 3 bits</li> <li>• Bits não utilizados (valor zero) – 5 bits</li> </ul>
<b>Ação</b>	Registrador 1 $\ll$ - Registrador 2
<b>Exemplo</b>	<ul style="list-style-type: none"> <li>• negative R0 R1</li> <li>• label: negative R0 R1</li> </ul>
<b>Representação gráfica</b>	

<b>Código da operação (5 bits)</b>	<b>16 - push</b>
<b>Significado</b>	Insere um valor contido em um registrador na pilha.
<b>Operandos</b>	<ul style="list-style-type: none"> <li>• Registrador – 3 bits</li> <li>• Bits não utilizados (valor zero) – 8 bits</li> </ul>
<b>Ação</b>	Pilha[ <i>stackpt</i> ] $\ll$ R0 <i>stackpt</i> $\ll$ <i>stackpt</i> + 2

<b>Exemplo</b>	<ul style="list-style-type: none"> <li>• push R0</li> <li>• label: push R0</li> </ul>
<b>Representação gráfica</b>	

<b>Código da operação (5 bits)</b>	<b>17 - pop</b>
<b>Significado</b>	Remove o topo da pilha e o coloca em um registrador.
<b>Operandos</b>	<ul style="list-style-type: none"> <li>• Registrador – 3 bits</li> <li>• Bits não utilizados (valor zero) – 8 bits</li> </ul>
<b>Ação</b>	$stackpt \ll stackpt - 2$ $R0 \ll Pilha[stackpt]$
<b>Exemplo</b>	<ul style="list-style-type: none"> <li>• pop R0</li> <li>• label: pop R0</li> </ul>
<b>Representação gráfica</b>	

<b>Código da operação (5 bits)</b>	<b>18 - addi</b>
<b>Significado</b>	Adiciona ao valor de um registrador uma constante (positiva ou negativa).
<b>Operandos</b>	<ul style="list-style-type: none"> <li>• Registrador de destino – 3 bits</li> <li>• Constante (com sinal) – 8 bits</li> </ul>
<b>Ação</b>	Registrador $\ll$ Registrador + constante
<b>Exemplo</b>	<ul style="list-style-type: none"> <li>• addi R0 2</li> <li>• addi R0 -2</li> <li>• label: addi 2</li> </ul>
<b>Representação gráfica</b>	

<b>Código da operação (5 bits)</b>	<b>19 - call</b>
<b>Significado</b>	Chama o procedimento que está contido no endereço de memória especificado. Empilha o endereço da próxima instrução a ser chamada (pc, antes do redirecionamento) para ser usado pelo <b>return</b> posteriormente.
<b>Operandos</b>	<ul style="list-style-type: none"> <li>• Bits não utilizados (valor zero) – 3 bits</li> <li>• Endereço de memória – 8 bits</li> </ul>
<b>Ação</b>	<b>push(pc)</b> $pc \ll \text{Memória}[\text{endereço}]$
<b>Exemplo</b>	<ul style="list-style-type: none"> <li>• call proc ;Procedimento identificado pelo label ‘proc’</li> <li>• label: call proc</li> </ul>
<b>Representação gráfica</b>	

<b>Código da operação (5 bits)</b>	<b>20 - return</b>
<b>Significado</b>	Encerra um procedimento e retorna para endereço especificado pelo valor no topo da pilha. ATENÇÃO: uma função chamada que insere elementos na pilha deve obrigatoriamente removê-los antes do <b>return</b> .
<b>Operandos</b>	<ul style="list-style-type: none"> <li>• Bits não utilizados (valor zero) – 11 bits</li> </ul>
<b>Ação</b>	$pc \ll \text{pop}()$
<b>Exemplo</b>	<ul style="list-style-type: none"> <li>• return</li> </ul>
<b>Representação gráfica</b>	