Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Computação e Sistemas
CSI 203 - Organização e Arquitetura de Computadores I
Trabalho Prático 2019/1

Prof. Carlos Henrique Gomes Ferreira

1. Descrição Geral:

O principal objetivo deste trabalho é implementar um montador para a máquina especificada, 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 especificada, 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 grupo de 4 pessoas, devendo ser discutido somente entre os colegas do mesmo grupo, não sendo permitido cópia ou compartilhamento do código fonte.
  - Em caso de plágio, todos os grupos envolvidos receberão nota 0.
- 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. A partir desse horário, os trabalhos já estarão sujeitos a penalidades. A fórmula para desconto por atraso na entrega do trabalho prático é:

$$Desconto = 2^d/0.32\% \quad (1)$$

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 nas linguagem 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 em uma máquina com o Linux Ubuntu.
  - Sugestão: Teste o seu montador em um máquina linux antes da submissão, pois algumas bibliotecas diferem no windows e no linux. Códigos que não executarem terão, inicialmente, nota zero atribuída.
- 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.
- Todas as dúvidas referentes ao trabalho serão esclarecidas por meio do monitor da disciplina ou pelo fórum aberto no Moodle especificamente para este propósito.
  - Não é obrigação do monitor lhe auxiliar no desenvolvimento do projeto. Ele estará disponível apenas para dúvidas pontuais.
- 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-aluno4.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 deverão ser incluídos outros arquivos nessa pasta.)
    - "tst": Essa pasta irá conter os arquivos ".a" de entrada, desenvolvidos para testar o montador. Um arquivo de exemplo é fornecido junto a esta especificação. 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.
    - A integridade do arquivo submetido no sistema é de inteira responsabilidade do aluno, ou seja, trabalhos com arquivos corrompidos terão nota zero.
  - **Atenção**: Trabalhos que descumprirem o padrão definido acima serão fortemente penalizados.

3. Especificação da Máquina ECEE R19.1:

A máquina a ser utilizada é a Máquina ECEE R19.1, 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 128 posições, numeradas de 0 a 127.
- 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 (256 elementos de 8 bits) acessada por instruções específicas (push e pop)
- Cada registrador pode armazenar uma palavra de 16 bits.
- Existem 4 registradores de uso geral, os quais são nomeados de "A0" a "A3". Apenas estes registradores estão disponível para o uso direto.
- Os registradores de propósito específico são:
  - **top** (Stack Top): O ponteiro de pilhas aponta para o topo da pilha (inicializado com 0). Uso interno, não podendo ser diretamente referenciado.
  - **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. Uso interno, não podendo ser diretamente referenciado.
  - **buffer1** e **buffer2**: Registradores que recebem operandos a serem tratados, como por exemplo, os operandos da instrução *add*. Uso interno, não podendo ser diretamente referenciado.
  - **ir**: Armazena a instrução em execução. É a partir desse registrador que a instrução é decodificada. Uso interno, não podendo ser diretamente referenciado.
  - **mar**: Armazena um endereço a ser lido ou escrito na memória. Uso interno, não podendo ser diretamente referenciado.
  - **mdr**: Armazena um dado recém-lido (por exemplo, uma instrução recém-carregada) ou a ser escrito na memória. Uso interno, não podendo ser diretamente referenciado.
  - **status**: Armazena uma flag chamada halt-bit que, se escrita, interrompe o programa imediatamente, gerando uma mensagem de encerramento na tela. Uso interno, não podendo ser diretamente referenciado.

- Cada instrução pode ter 0, 1 ou 2 operandos, e sempre possui 16 bits de tamanho. Alguns operandos são especificados com mais bits do que o necessário. Neste caso, sempre é utilizado os n bits menos significativos que são necessários. 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 apêndice A.2 - Montador (ou Assembler) do livro: *Organização e Projeto de Computadores: A Interface Hardware/Software*. David Patterson e John Hennessy.
  - Atenção: Faz parte do trabalho entender a ideia de ter dois passos no processo de montagem. Considere também materiais da *Web*, disponibilizados gratuitamente para lhe auxiliar na compreensão de tais conceitos.
- O conjunto de instruções é o especificado pela Máquina ECEE R19.1, descrito em mais detalhes no Anexo 1 deste documento.
- Cada linha da linguagem de montagem da máquina ECEE R19.1 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 A0 A1 ;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 ECEE 19.1.

  - 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 ECEE R19.1 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 ECEE R19.1, ao **menos uma chamada de procedimento utilizando a pilha em algum momento.**

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 ECEE R19.1, 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

2. Saída em formato ".mif": Arquivo texto que define cada célula de memória (1 byte no caso da máquina Swombat R3.0), 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 ECEE R19.1.
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 Entrada no Simulador:

A saída do montador (arquivo no formato ".hex"ou ".mif") deve ser executada na máquina ECEE R19.1 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 ECEE R19.1.
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.

Por fim, é importante destacar que apresentação de como o projeto foi desenvolvido poderão ocorrer, sendo avaliada da seguinte forma:

$$NotaFinal = N_A * N_{TP} \tag{2}$$

onde $0 \leq N_A \leq 1$ e $0 \leq N_{TP} \leq 1$.

ANEXO 1 - Conjunto de Instruções da Máquina ECEE R19.1

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 houver) também é definida em cada instrução[1].

Código da operação (5 bits)	0x0 – stop
Significado	Encerra o programa, através da escrita da flag halt-bit.
Operandos	Bits não utilizados (valor zero) – 16 bits
Ação	N/A
Exemplos de uso	stop label: stop

Código da operação (5 bits)	0x01 – load
Significado	Carrega uma palavra da memória para um registrador.
Operandos	Registrador – 2 bits Endereço de Memória – 9 bits
Ação	Registrador = Memória[endereço]
Exemplos de uso	load A0 150 load A0 var; 'var' pode ser uma variável estática pré-definida na memória com a pseudo-instrução .data label: load A0 var

Código da operação (5 bits)	0x02 – store
Significado	Armazena uma palavra de um registrador em um endereço de memória.
Operandos	Registrador – 2 bits Endereço de Memória - 9 bits
Ação	Memória[Endereço] = Registrador
Exemplos de uso	store A0 150 store A0 var; 'var' pode ser uma variável estática pré-definida na memória com a pseudo-instrução .data label: store var

[1] O formato 0x representa notação hexadecimal.

Código da operação (5 bits)	0x03 – read
Significado	Armazena o valor da entrada digitada no Registrador 1.
Operandos	Registrador 1 – 11 bits (Use apenas os 2 bits menos significativos [14-15] para endereçar o registador)
Ação	Registrador 1 = Input
Exemplos de uso	load A0 label: load A0

Código da operação (5 bits)	0x04 - write
Significado	Imprime o valor do registrado no console.
Operandos	Valor a ser impresso.
Ação	Output = Registrador 1
Exemplos de uso	write A0 label: write A0

Código da operação (5 bits)	0x05 - add
Significado	Faz a soma dos valores de dois registradores e armazena no primeiro registrador.
Operandos	Registrador 1 – 2 bits Registrador 2 – 9 bits (Use somente os 2 bits menos significativos [14-15] para endereçar o registador 2.)
Ação	Registrador 1 = Registrador 1 + Registrador 2
Exemplos de uso	add A0 A1 label: add A0 A1

Código da operação (5 bits)	0x06 - subtract
Significado	Faz a subtração dos valores de dois registradores e armazena no primeiro registrador.
Operandos	Registrador 1 – 2 bits Registrador 2 – 9 bits (Use os 2 bits menos significativos [14-15] para endereçar o registrador 2.)
Ação	Registrador 1 = Registrador 1 - Registrador 2
Exemplos de uso	subtract A0 A1 label: subtract A0 A1

Código da operação (5 bits)	0x07 - multiply
Significado	Faz a multiplicação dos valores de dois registradores e armazena no primeiro registrador.
Operandos	Registrador 1 – 2 bits Registrador 2 – 9 bits (Use os 2 bits menos significativos [14-15] para endereçar o reg. 2.)
Ação	Registrador 1 = Registrador 1 * Registrador 2
Exemplos de uso	multiply A0 A1 label: multiply A0 A1

Código da operação (5 bits)	0x08 - divide
Significado	Faz a divisão dos valores de dois registradores e armazena no primeiro registrador.
Operandos	Registrador 1 – 2 bits Registrador 2 – 9 bits (Use os 2 bits menos significativos [14-15] para endereçar o reg. 2.)
Ação	Registrador 1 = Registrador 1 / Registrador 2
Exemplos de uso	divide A0 A1 label: divide A0 A1

Código da operação (5 bits)	0x09 - jump
Significado	Salta para a instrução contida no endereço de memória especificado.
Operandos	Endereço de Memória – 11 bits (Use os 9 bits menos significativos [7-15].)
Ação	pc = endereço
Exemplos de uso	jump 50 label: jump 50

Código da operação (5 bits)	0A - jmpz
Significado	Salta para a instrução contida no endereço de memória especificado, caso o valor contido no registrador seja igual a zero.
Operandos	Registrador – 2 bits Endereço de Memória – 9 bits
Ação	Se (Registrador == 0): pc = Endereço
Exemplos de uso	jmpz A0 50 jmpz A0 label label: jmpz A0 50

Código da operação (5 bits)	0x0B - jmpn
Significado	Salta para a instrução contida no endereço de memória especificado, caso o valor contido no registrador seja menor que zero.
Operandos	Registrador – 2 bits Endereço de Memória – 9 bits
Ação	Se (Registrador <0): pc = endereço
Exemplos de uso	jmpn A0 50 jmpn A0 label label: jmpn A0 50

Código da operação (5 bits)	0x0C - move
Significado	Move o valor de um registrador para o outro.
Operandos	Registrador 1 – 2 bits Registrador 2 – 9 bits (Use somente os 2 bits menos significativos [14-15])
Ação	Registrador 1 = Registrador 2
Exemplos de uso	move A0 A1 label: move A0 A1

Código da operação (5 bits)	0x0D - push
Significado	Insere um valor contido em um registrador na pilha.
Operandos	Registrador – 11 bits (Use os 2 bits menos significativos [14-15])
Ação	Stack[top] = Registrador top = top + 2
Exemplos de uso	push A0 label: push A0

Código da operação (5 bits)	0x0E - pop
Significado	Remove o topo da pilha e o coloca em um registrador.
Operandos	Registrador – 11 bits (Use os 2 bits menos significativos [14-15])
Ação	top = Top - 2 Registrador 1 = Pilha[top]
Exemplos de uso	pop A0 label: pop A0

Código da operação (5 bits)	0x0F - call
Significado	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 return posteriormente.
Operandos	Endereço de Memória – 11 bits (Use os 9 bits menos significativos [7-15])
Ação	push(pc) pc = Memória[endereço]
Exemplos de uso	call proc ;Procedimento identificado pelo label 'proc' label: call proc

Código da operação (5 bits)	0x8000 - return
Significado	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 return.
Operandos	Bits não utilizados (valor zero) – 16 bits
Ação	pc = pop()
Exemplos de uso	return

Código da operação (5 bits)	0x13 – load_c (Load Constant)
Significado	Carrega uma constante de 9 bits (Notação Complemento 2) em um registrador. A constante é armazenada em um registrador.
Operandos	Registrador 1– 2 bits Endereço de Memória – 9 bits
Ação	Registrador 1 = Constante
Exemplos de uso	load_c A0 -1 label: load_ A0 150

Código da operação (5 bits)	0x16 – copytop
Significado	Carrega no registrador 1 o endereço do topo da pilha (ou seja, o endereço contido no ponteiro superior, não os dados no topo da pilha).
Operandos	Registrador 1 – 2 bits Registrador 2 – 11 bits (Use os 2 bits menos significativos [14-15])
Ação	Registrador 1 = top
Exemplos de uso	copyto A0 label: copyto A0