Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Disciplina Software Básico (DCC008) - 2016/1

Trabalho Prático 1 - Emulador

1. Descrição Geral

Este trabalho envolve a implementação de um emulador para uma máquina básica, capaz de realizar um pequeno conjunto de operações.

Atenção: todos os trabalhos práticos da disciplina dependerão deste emulador. Implemente-o com cuidado, pois erros na máquina podem impactar nas notas de todos os trabalhos.

2. Informações Importantes

- O trabalho deve ser feito em duplas ou individualmente, podendo ser discutido entre os colegas, mas o código fonte não pode ser compartilhado.
- A data de entrega será especificada através de uma tarefa no Moodle.
- Os trabalhos poderão ser entregues até a meia-noite do dia especificado para a entrega. O horário de entrega deve respeitar o relógio do sistema Moodle, ou seja, a partir de 00:01 do dia seguinte à entrega no relógio do Moodle, os trabalhos já estarão sujeitos a penalidades. A fórmula para desconto por atraso na entrega do trabalho prático ou resenha é:

  $$Desconto = 2^{d-1}/0.32\ \%$$

  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 e 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 (entre 3 e 5 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. As instruções de submissão, alguns arquivos de teste e o esqueleto da organização dos arquivos estão presentes no arquivo "tp1_login1_login2.tar.gz", disponível para download no Moodle.
- **Atenção**: trabalhos que descumprirem esse padrão serão penalizados.

3. Especificação da Máquina Simple86

A máquina a ser emulada é a Simple86, projetada exclusivamente para a disciplina. Seguem as especificações:

- A menor unidade endereçável nessa máquina é uma palavra de 16 bits (um inteiro): O primeiro byte da memória tem endereço zero, o terceiro o endereço um, e assim sucessivamente.
- Os tipos de dados tratados pela máquina também são somente inteiros.
- A máquina possui uma memória de 1000 posições.
- Cada posição da memória armazena uma palavra (16 bits). Cada registrador também possui tamanho de 16 bits.
- Os registradores de uso geral são:
    - AX: acumulador em instruções aritméticas e lógicas;
    - BX: usado por algumas instruções para armazenar ponteiros para dados armazenados na memória
    - CX: usado em instruções especiais, como o controle de loop
    - Os registradores podem ser acessados por inteiro, os 16 bits ou por apenas os bits mais significativos H ou os menos significativos L são codificados da seguinte forma:

Código	Registrador
0	AL (8 bits menos significativos de AX)
1	AH (8 bits mais significativos de AX)
2	AX (16 bits)
3	BH (8 bits mais significativos de BX)
4	BL (8 bits menos significativos de BX)
5	BX (16 bits)
6	CL (8 bits menos significativos de CX)

7	CH (8 bits mais significativos de CX)
8	CX (16 bits)

- Os registradores de propósito específico são:
    - BP (BASE POINTER): é usado para registrar o endereço de memória da base da pilha. Como a máquina Simple86 possui 1000 posições de memória, BP inicia a execução com o endereço da posição de memória 1001.
    - SP (STACK POINTER): o ponteiro de pilhas, aponta para o topo da pilha. O SP é inicializado com o mesmo valor de BP, na posição 1001. Quando há a requisição para empilhar valor, esse será empilhado na posição de memória (SP - 1). Nesse caso o primeiro valor será armazenado na posição 1000, o segundo na 999, o terceiro na posição 998, e assim por diante, até que sejam desempilhados e o valor de SP volte ao valor original.
    - IP (INSTRUCTION POINTER): Armazena o endereço da próxima instrução a ser executada. O IP inicia a execução do programa em 0 (zero) e é automaticamente incrementado a cada ciclo de instrução de forma que as instruções são normalmente executadas sequencialmente a partir da memória. O IP é afetado também pelas instruções de desvio e chamadas de procedimentos.
- Registradores de Flags (1 bit):
    - ZF (Zero Flag): indica se o resultado de uma operação é igual a zero (1) ou diferente de zero (0).
    - SF (Sign Flag): indica se o resultado de uma operação com sinal é positivo (0), caso contrário (1).
- Cada instrução pode ter 0, 1 ou 2 operandos.
- As instruções são codificadas da seguinte forma:
    - o primeiro byte indica a operação (instrução)
    - o segundo byte indica o tipo dos operandos conforme a seguinte tabela:

Código	Operando
0	Nenhum operando
1	Registrador
2	Memória
3	Registrador e memória
4	Memória e Registrador
5	Registrador e Registrador
6	Memória e imediato (número)

7	Registrador e imediato (número)
8	Imediato (número)

- Os dois próximos bytes da instrução dependerão de haver ou não operandos e o tipo de operando definido, sendo que cada um dos operandos serão representados por uma palavra de 2 bytes. Esses dois bytes representarão o registrador, ou a posição de memória ou um imediato (número). Assim serão possíveis instruções com 16 bits, 32 bits e 48 bits.
- Os imediatos são valores em hexadecimal. Para indicar isso, todo valor começa com a string 0x. Por exemplo, 0xABCD é um valor hexadecimal.
- Os registradores podem armazenar tanto valores negativos quanto positivos. Os valores negativos são representados no formato de complemento de dois, ou seja, você pode usar as variáveis inteiras de 8 e 16 bits do C do seu PC para fazer as operações sem nenhuma conversão. Dica: use os tipos int8_t e int16_t. O tamanho dos tipos padrões do C (*short* e do *int*) pode variar de arquitetura para arquitetura, mas o int8_t e o int16_t não variam.

O conjunto de instruções da Máquina Simple86 está no anexo 1 deste documento.

4. Descrição da Tarefa

Sua tarefa é implementar um programa que emule a máquina Simple86, ou seja, implementar uma máquina virtual que execute programas em linguagem de máquina conforme definido na seção anterior.

O trabalho pode ser visto como possuindo duas partes: a primeira é o interpretador da máquina propriamente dito, que contém uma representação da memória da máquina, e o interpretador do programa na memória.

A segunda parte é o carregador do programa, que consiste em ler de um arquivo um programa em linguagem de máquina contendo as instruções.

Essas duas "peças" são fundamentais para a continuidade dos trabalhos práticos da disciplina. Os outros trabalhos dependerão delas.

5. Formato da Entrada de Dados

O arquivo binário sempre será lido de 2 em 2 bytes. Lidos os 2 bytes iniciais, será possível saber a instrução e quantos bytes compõem a instrução (2, 4 ou 6 bytes). Caso a instrução contenha operandos, os 2 bytes seguintes à instrução serão do primeiro operando e os outros 2 bytes, o segundo operando (caso exista).

O programa inicialmente deverá ser carregado na memória da máquina, a partir da posição 0 (zero) e o valor de IP deverá ser inicializado com zero de forma que a execução do programa se iniciará na posição de memória zero.

OBS1: Assuma que uma codificação little endian será utilizada. Se você usar uma máquina de CPU Intel, não deverá ter problemas. Caso contrário, pode ser necessário tratar a conversão de codificação de sua máquina. Nesse caso, em vez de tratar a conversão, provavelmente será mais fácil utilizar uma das máquinas dos laboratórios do DCC.

Exemplo de executável:

Linguagem de montagem	**Arquivo Binário (2 em 2 bytes)**	**Explicação**	**AX**	**BX**	**CX**	**SP**	**BP**	**IP**	**ZF**	**SF**
READ AX	0x1201	0x12 identifica a instrução READ e 0x1 a opção de operandos Registrador. Instrução READ é executada								
	0x0002	Identificamos que o registrador será o AX. O número 0 foi digitado. ZF = 1, SF = 0. AX=0000;	0	0	0	3E9	3E9	2	1	0
SUB AX, 0xA	0x0307	Instrução SUB (0x3), Opção de operandos Registrador / Número (0x7)								
	0x0002	Identifica o registrador AX								
	0x000A	Número 10 em decimal	FFF6	0	0	3E9	3E9	5	0	1
JS exit	0x0C02	Instrução JS (0x0C), Opção de operandos Memória (0x02)								
	0x0009	Identifica posição de memória 09d. A instrução JS avalia o SF para realizar o desvio, nesse caso igual a 1.	FFF6	0	0	3E9	3E9	9	0	1
WRITE AX	0x1301	Instrução WRITE (0x13), opção de operandos Registrador (0x1)								
	0x0002	Identifica o registrador AX								
exit: HLT	0x1400	Instrução HLT, execução finalizada	FFF6	0	0	3E9	3E9	A	0	1

O arquivo binário correspondente a esse programa de exemplo está disponível no pacote do trabalho prático, em tst/tp1ex1.sa. Ao executá-lo no emulador, o programa deve ler um número e subtrair 10d. No caso do resultado ser maior do que zero, o programa finalizará, caso contrário irá imprimir

na tela o resultado da operação antes de finalizar. Os programas que serão disponibilizados de exemplo não testam todas as instruções, então outros programas devem ser obrigatoriamente implementados com o objetivo de cobrir todo o conjunto de instruções. A qualidade dos testes implementados será levada em conta na correção.

6. Formato de Saída de Dados

A saída das instruções WRITE e DUMP deve ser sempre numérica, ocupando uma única linha. No caso da instrução WRITE, uma linha é impressa contendo um único número (o valor do acumulador em hexadecimal). No caso da instrução DUMP, uma linha é impressa contendo o valor dos registradores em hexadecimal, separados um do outro por um espaço em branco, conforme especificado no Anexo 1.

No exemplo da seção anterior, supondo que o valor recebido por AX na instrução READ foi de 0x106A, ao final da execução, a saída do programa dada pela instrução DUMP deverá ser:

```
AX   BX   CX   SP   BP   IP   ZF   SF
1060 0000 0000 03E9 03E9 000A 0000 0000
```

A impressão da instrução DUMP será feita alinhada em 4 bytes, para facilitar a legibilidade dos registradores. Deverá ser impressa a primeira linha com o nome dos registradores, e em seguida a segunda linha com o valor dos mesmos.

7. Formato de Chamada do Emulador

O único parâmetro recebido pelo emulador é o nome do arquivo contendo o programa a ser executado pela máquina virtual. Exemplo de chamada:

./emulador teste1.sa

8. Sobre a Documentação

- Deve conter todas as decisões de projeto.
- Deve conter informações sobre como executar o emulador. Obs: é necessário cumprir os formatos definidos acima para a execução, mas tais informações devem estar presentes também na documentação.
- Deve conter elementos que comprovem que o emulador foi testado (ex: imagens das telas de execução). Os arquivos relativos a testes devem ser enviados no pacote do trabalho. 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 de documentação.

9. Considerações Finais

Os testes de funcionamento da máquina virtual possivelmente serão automatizados, o que torna necessário o cumprimento fiel de todas as especificações de interface descritas neste documento. As decisões de projeto devem fazer parte apenas da estrutura interna do Emulador, não podendo afetar os resultados esperados de cada instrução.

ANEXO 1 - Conjunto de Instruções da Máquina Simple86

Código	**01 - MOV**
Operandos	● Registrador e memória; ● Memória e Registrador; ● Registrador e Registrador; ● Memória e imediato (número); ● Registrador e imediato (número)
Tamanho	48 bits
Significado	Copia o operando 2 para o 1
Ação	operando1 ← operando2
Flags afetados	Nenhum
Exemplo	● MOV AX, 0xB800 ; AX recebe o valor B800h ● MOV BX, AX ● MOV BX, 0x15E

Código	**02 - ADD**
Operandos	● Registrador e memória; ● Memória e Registrador; ● Registrador e Registrador; ● Memória e imediato (número); ● Registrador e imediato (número)
Tamanho	48 bits
Significado	Soma ao valor do operando1 o valor do operando2.
Ação	operando1 ← operando1 + operando2
Flags afetados	ZF e SF
Exemplo	● MOV AL, 0x5 ; AL = 5 ADD AL, 0x3 ; AL = 8

Código	**03 - SUB**
Operandos	● Registrador e memória; ● Memória e Registrador; ● Registrador e Registrador;

	• Memória e imediato (número); • Registrador e imediato (número)
Tamanho	48 bits
Significado	Subtrai do valor do operando1 o valor do operando2.
Ação	operando1 ← operando1 - operando2
Flags afetados	ZF e SF
Exemplo	MOV AL, 0x5 SUB AL, 0x1 ; AL = 4

Código	**04 - MUL**
Operandos	• Registrador; • Memória;
Tamanho	32 bits
Significado	Multiplicação sem sinal
Ação	se o operando for *byte:* AX = AL * operando. se o operando for *word*: AX = AX * operando.
Flags afetados	Nenhum

Exemplo	MOV AL, 0xC8H ; AL = 200
	MOV BL, 0x4
	MUL BL ; AX = 0320H (800)

Código	**05 - DIV**
Operandos	• Registrador; • Memória;
Tamanho	32 bits
Significado	Divisão sem sinal
Ação	se o operando for *byte*: AL = AX / operando AH = recebe o resto da divisão

	se o operando for *word*: AX = AX / operando BX = recebe o resto da divisão
Flags afetados	Nenhum

Exemplo	MOV AX, 0x00CB ; AX = 203
	MOV BL, 0x4
	DIV BL ; AL = 50 (32h), AH = 3

Código	**06 - AND**
Operandos	● Registrador e memória; ● Memória e Registrador; ● Registrador e Registrador; ● Memória e imediato (número); ● Registrador e imediato (número)
Tamanho	48 bits
Significado	Operação lógica AND bit a bit de dois operandos. O resultado é colocado no operando 1.
Ação	1 AND 1 = 1 1 AND 0 = 0 0 AND 1 = 0 0 AND 0 = 0
Flags afetados	ZF e SF
Exemplo	● MOV AL, 0x61 ; AL = 01100001b AND AL, 0xCF ; AL = 01000001b

Código	**07 - NOT**
Operandos	● Registrador; ● Memória
Tamanho	32 bits
Significado	Operação lógica NOT bit a bit. Inverte cada bit do operando.
Ação	se o bit for igual a 1 transforma em 0 e vice versa.
Flags afetados	ZF e SF

Exemplo	• MOV AL, 0x1B NOT AL ; AL = 0xC4

Código	**08 - OR**
Operandos	• Registrador e memória; • Memória e Registrador; • Registrador e Registrador; • Memória e imediato (número); • Registrador e imediato (número)
Tamanho	48 bits
Significado	Operação lógica OR bit a bit de dois operandos. O resultado é colocado no operando 1.
Ação	1 OR 1 = 1 1 OR 0 = 1 0 OR 1 = 1 0 OR 0 = 0
Flags afetados	ZF e SF
Exemplo	• MOV AL, 0x41 ; AL = 01000001b OR AL, 0x83 ; AL = 11000011b

Código	**09 - CMP**
Operandos	• Registrador e memória; • Memória e Registrador; • Registrador e Registrador; • Memória e imediato (número); • Registrador e imediato (número)
Tamanho	48 bits
Significado	Compara dois valores.
Ação	Algoritimo: operando1 - operando2 O resultado não é salvo em nenhum lugar, apenas os valores dos registradores de flags são alterados.
Flags afetados	ZF e SF
Exemplo	MOV AL, 0x5 MOV BL, 0x5

	CMP AL, BL ; AL = 5, ZF = 1(significa que é igual!)

Código	**10 - JMP**
Operandos	• Memória
Tamanho	32 bits
Significado	Desvio incondicional. Transfere o controle para outra parte do programa.
Ação	Sempre desvie para a posição de memória indicada.
Flags afetados	Nenhum.

Exemplo	MOV AL, 0x5
	JMP label1 ; Desvio de 1 linha
	MOV AL, 0x0
	label1:
	WRITE AL ; Irá escrever na tela 5

Código	**11 - JZ**
Operandos	• Memória
Tamanho	32 bits
Significado	Desvio se igual a zero.
Ação	Se ZF = 1 então desvie. ZF é afetado pelas instruções CMP, SUB, ADD, AND e OR.
Flags afetados	Nenhum.

Exemplo	MOV AL, 0x5
	CMP AL, 0x5
	JZ label1
	SUB AL, 0x1 ;AL é diferente de 5.
	JMP exit
	label1:
	ADD AL, 0x1 ; AL é igual a 5.
	exit:
	HLT

Código	**12 - JS**

Operandos	• Memória
Tamanho	32 bits
Significado	Desvio se negativo.
Ação	Se SF = 1 então desvie. SF é afetado pelas instruções CMP, SUB, ADD, AND, e OR.
Flags afetados	Nenhum.

Exemplo	MOV AL, 0x80b ; AL = -128
	OR AL, 0x0
	JS label1
	SUB AL, 0x1 ; sem sinal (positivo)
	JMP exit
	label1: ADD AL, 0x1 ; com sinal (negativo)
	exit: HLT

Código	**13 - CALL**
Operandos	• Memória
Tamanho	32 bits
Significado	Realiza a chamada à um procedimento (ou "função")
Ação	Transfere o controle para um procedimento em outro ponto da memória. Nesse momento é necessário que o endereço de retorno seja colocado na pilha. O endereço de retorno é a posição de memória da próxima instrução a ser executada após a chamada da função.
Flags afetados	Nenhum

Exemplo	CALL p1 ;Empilha o IP da próxima instrução e transfere o IP para p1
	ADD AX, 0x1
	...
	p1 PROC ; declaração de um procedimento.
	MOV AX, 0x1234
	RET ; retorna para a posição indicada no endereço de retorno.
	p1 ENDP

Código	**14 - RET**
Operandos	Nenhum
Tamanho	16 bits

Significado	Retorno de um procedimento (ou “função”).
Ação	Desempilha da memória o endereço de retorno que foi empilhado pela chamada do procedimento CALL e a execução do programa retorna exatamente na instrução seguinte à aquela em que foi realizada a chamada da função.
Flags afetados	Nenhum

Exemplo	CALL p1
	ADD AX, 0x1 ; O comando RET retornará a execução para essa linha.
	...
	p1 PROC ; declaração de um procedimento.
	MOV AX, 0x1234
	RET ; retorna para a posição indicada no endereço de retorno.
	p1 ENDP

Código	**15 - PUSH**
Operandos	● Registrador; ● Memória; ● Imediato (número)
Tamanho	32 bits
Significado	Empilha um valor na memória.
Ação	SP = SP - 1 O local da memória apontada por SP recebe o valor do operando.
Flags afetados	Nenhum
Exemplo	MOV AX, 0x1234 PUSH AX POP BX ; BX = 0x1234

Código	**16 - POP**
Operandos	● Registrador; ● Memória;
Tamanho	32 bits
Significado	Desempilha um valor da memória.
Ação	operando = O valor no topo da pilha SP = SP + 1

Flags afetados	Nenhum
Exemplo	MOV AX, 0x1234 PUSH AX POP BX ; BX = 0x1234

Código	**17 - DUMP**
Operandos	Nenhum
Tamanho	16 bits
Significado	Imprime os valores atuais de todos os registradores
Ação	Imprimi o estado de todos os registradores separados por um espaço " " na seguinte ordem: AX, BX, CX, SP, BP, IP, ZF e SF.
Flags afetados	Nenhum

Exemplo	DUMP ; será impresso:
	`AX BX CX SP BP IP ZF SF`
	`1060 0000 0000 03E9 03E9 0009 0000 0000`

Código	**18 - READ**
Operandos	● Registrador; ● Memória;
Tamanho	32 bits
Significado	Leitura de dado do input padrão.
Ação	Lê um hexadecimal do input padrão (teclado) e atribui à um registrador ou local da memória.
Flags afetados	ZF e SF
Exemplo	READ AX ; Foi digitado "107A". AX = 0x107A

Código	**19 - WRITE**
Operandos	● Registrador; ● Memória;

Tamanho	32 bits
Significado	Escrita de um valor no output padrão.
Ação	Imprime o valor do registrador ou da posição de memória, em hexadecimal, no output padrão (monitor).
Flags afetados	Nenhum
Exemplo	MOV AX, 0xCB ; AX = 203 WRITE AX ; Imprime na tela: 00CB

Código	**20 - HLT**
Operandos	Nenhum
Tamanho	16 bits
Significado	Interrompe a execução.
Ação	Interromper
Flags afetados	Nenhum
Exemplo	HLT