INSTITUTO FEDERAL
Espírito Santo
Campus Serra

Curso: Bacharelado em Sistemas de Informação
Disciplina: Arquitetura e Organização de Computadores
Período: 2023/1
Professor: Flávio Giraldeli

Turma: BSI3
CH: 60 h

Resolvendo problemas em C e Assembly

Objetivo

Implementar três pequenos programas, usando a linguagem **Assembly x86** (16 bits) e **C**. Para cada caso, após a implementação em alto e baixo nível, uma análise comparativa deverá ser feita a fim de evidenciar os detalhes de Arquitetura de Computadores que são ocultados pela linguagem de alto nível, C.

Grupos

Deverão ser formados grupos com 5 a 6 integrantes, a serem escolhidos livremente (contando que pelo menos um domine os aspectos básicos de programação em C). Casos particulares deverão ser comunicados ao professor imediatamente e a decisão deverá ficar a critério do professor.

Pontuação

O referido trabalho será avaliado de 0 a 100 e corresponderá a 20% da nota semestral. A nota do trabalho será calculada da seguinte forma:

$$Nota\ do\ Trabalho = (0{,}15 * P_1 + 0{,}20 * P_2 + 0{,}25 * P_3) + 0{,}4 * Relatório$$

Todas as notas sempre são avaliadas de 0 a 100, mas com pesos diferentes conforme a fórmula acima. Cada P corresponde a um problema a ser resolvido (próxima página). Portanto, 60% da nota envolve as implementações e 40% para o Relatório. E, em cada problema, a nota é composta por 75% (Assembly) + 25% (C). O código em Assembly vale bem mais, claro!

Implementação

Os três problemas abaixo foram cuidadosamente propostos para explorarem aspectos básicos da programação em baixo nível. Cada um deles deverá ser resolvido separadamente, com arquivos com a seguinte nomenclatura:

`Problema1.c`	`Problema2.c`	`Problema3.c`
`Problema1.asm`	`Problema2.asm`	`Problema3.asm`

Em cada uma das implementações, o comportamento do programa executável deverá ser indistinto, ou seja, quando executados, tanto o executável originado do código C, quanto o provindo do Assembly têm que se comportar **exatamente da mesma forma** do ponto de vista do usuário.

A implementação em C poderá ser feita no compilador que o grupo desejar (ou que já esteja acostumado a usar). Ou, se quiser uma dica, pode começar a usar uma IDE profissional como o ***Microsoft Visual Studio Community*** (Gratuito).

Já a implementação em Assembly deverá fazer uso do **emu8086**. O formato do arquivo Assembly poderá ser um `.com`, cuja estrutura "mono-segmento" é bem mais simples que um `.exe`. Nenhum procedimento/macro pré-pronto poderá ser chamado diretamente. No entanto, você poderá **abrir o arquivo inc\emu8086.inc** e **copiar uma ou outra rotina básica** e inserir diretamente no seu código. As rotinas (macro/procedimento) deverão ser traduzidas e plenamente compreendidas pelo grupo. A qualidade da sua implementação, usando corretamente os recursos aprendidos (como chamadas a procedimento, macros, uso da pilha, saltos, variáveis etc.) será avaliada.

Faça uso extensivo do emulador. Lembre-se que o objetivo primário é o aprendizado de AOC, e não a programação em Assembly. O emulador possui simulação passo a passo da execução, sendo possível visualizar todas as informações de estado (registradores, memória, pilha, variáveis, etc...) em tempo real.

Vamos falar de Plágio e IA?

Então... todo professor honesto deseja que seus(suas) alunos(as) também tenham honestidade no processo de aprendizagem. Essa honestidade é fundamental para que o professor identifique possíveis deficiências na aprendizagem e possa intervir para tentar solucionar. Em termos mais simples, eu, Giraldeli, preciso saber se você aprendeu ou não os conceitos abordados nesse trabalho. E estou mensurando isso através de três pequenas tarefas (onde você constrói de um algoritmo que soluciona cada um desses pequenos problemas). Se você abre mão de exercitar seu cérebro e busca soluções mais "fáceis", perde-se completamente o sentido do trabalho.

Hoje há duas formas básicas de você cometer essa desonestidade nesse trabalho: copiando trechos significativos da internet ou de outros colegas (plágio direto) e/ou consultando diretamente ferramentas de IA (como o chatGPT). Quanto ao primeiro caso, é preciso deixar claro: **Não serão admitidos, em hipótese alguma, trechos de código comuns ou semelhantes entre grupos. Isto será rigorosamente observado.** Assim sendo, evitem a "cooperação" entre os grupos. A mesma observação vale "pesquisas na internet" e em trabalhos feitos por grupos em turmas anteriores em eventuais problemas semelhantes (o professor tem **TODOS** os trabalhos da história da disciplina arquivados para consulta).

Já a questão do uso indiscriminado de **ferramentas de IA**, a coisa é mais complicada, já que é difícil afirmar que você a **usou extensivamente no seu trabalho**, o que, acho quase desnecessário dizer, **é proibido**! O que farei nesse caso é simples: **caso eu desconfie**, por alguma razão (experiência na disciplina, talvez?), que a sua solução não partiu do seu esforço direto, **o grupo inteiro será convocado para uma entrevista presencial**. Nessa entrevista eu farei perguntas e possivelmente solicitarei algumas modificações "ao vivo" no código apresentado. Caso seja observada a incapacidade do grupo diante desse cenário, a nota será considerada zero. Caso a um ou mais membros se neguem a comparecer nessa entrevista, a mesma decisão será tomada: nota zero.

TODOS os membros do grupo são pessoalmente responsáveis pela observância a respeito dessa regra. Se vocês eventualmente "dividirem" o trabalho e um membro resolver cometer essa grave infração, TODO O GRUPO será punido! "E isso é justo, professor?" Sim. O trabalho é assinado por todos os membros do grupo, não é individual.

LEMBREM-SE: Não existem "coincidências" em código Assembly. É altamente improvável que existam trechos semelhantes entre grupos (ou da internet ou de turmas anteriores) sem que isso tenha sido INTENCIONAL (PLÁGIO). Já houve plágio em semestres anteriores e isso foi severamente punido. A nota final será **zero** e o nome de cada aluno(a) será encaminhado para o conselho de ética da escola. **NÃO TESTE SEU PROFESSOR**.

Professor... agora você me deixou preocupado(a)! 👀 Respondo: Se você não está sendo desonesto(a), você não tem com que se preocupar. Prometo.

Por fim... uma última coisa: isso não é um desafio. Mas, se assim você quiser, eu não vou fugir.

Relatório

É necessária a confecção de um relatório escrito do trabalho, correspondendo a 40% da nota, conforme o tópico anterior. É um Relatório Simples e, apesar de haver certo grau de liberdade na sua elaboração, os seguintes tópicos devem ser abordados:

- Todos os aspectos da linguagem de baixo nível que a implementação em C simplesmente "oculta", ou seja, você sequer precisa saber que aquilo existe em Assembly ao implementar em C. Dê exemplos.
- Comparação da quantidade de instruções relevantes presentes em C/Assembly;
- Explicação do funcionamento básico de todos os macros/procedimentos que foram copiados do inc\emu8086.inc. Você pode descrever textualmente o funcionamento de cada um deles. Um em cada parágrafo.
- Avaliação do grupo do tempo necessário para cada implementação. Tome a implementação em C como referência e estime o percentual de tempo adicional necessário para a implementação em Assembly.
- Conclusão, incluindo uma autoavaliação do aprendizado adquirido.

Entrega

A data limite para entrega do trabalho é o dia **04/07/2023 (terça-feira)** até as **23:55**.

O trabalho deverá ser postado por APENAS UM membro do grupo, no local correspondente no AVA, e deverá ter, obrigatoriamente, o nome:

[AOC 2023-1] Trabalho 2 - Aluno1, Aluno2, Aluno3, Aluno4, Aluno5, Aluno6.zip

O pacote comprimido (zip) deverá conter os **6 arquivos mencionados anteriormente** e um **relatório em formato PDF** conforme recomendações acima citadas (`relatorio.pdf`). **Siga atentamente essas regras!**

PROBLEMAS A SEREM RESOLVIDOS

Competências envolvidas: Comparações, Desvios condicionais, operações aritméticas, input/output de caracteres.

1) Volte a sua disciplina de Programação 1[1]. Olhe os primeiros problemas que você fez. Encontre um que use apenas comandos de decisão (if's) e operações aritméticas (ou seja, nada de loops e nem de funções). **Especifique o problema no relatório** e o resolva em C/Assembly conforme as especificações desse trabalho. Mas, atenção: A escolha de um exercício trivial (`if (a>0) printf("positivo"); else printf("negativo"); //` ¬¬) fará com que sua pontuação seja significativamente mais baixa. #DeGraçaEssa

Competências envolvidas: Anterior + *Loops* (iterações/comandos de repetição).

2) Uma PA (Progressão Aritmética) é uma sequência onde o n-ézimo elemento é formado pelo elemento base ($\mathbf{a_0}$) somado ao produto da razão (**r**) por **n**. Solicite ao usuário que digite o elemento base, a razão e um número inteiro maior que zero denominado **k**. A seguir, imprima todos os números da progressão aritmética formada por esses parâmetros que sejam menores que **k**.

Competências envolvidas: Anterior + Procedimentos.

3) A soma dos termos de uma PA (Progressão Aritmética) é dada por:

$$S_n = \frac{(a_1 + a_n) * n}{2}$$

onde temos, no numerador, respectivamente, o primeiro termo, o último termo e o número de termos. Sabendo disso, elabore um programa que solicita ao usuário esses três parâmetros, repassa os mesmos para uma função (procedimento, em Assembly) que calculará a soma da PA e retornará esse resultado para o programa principal, que se encarregará de exibi-lo na tela.

ATENÇÃO:

- Você deve fazer exatamente como está se pedindo, usando um procedimento (Assembly) e uma função (em C) já que esse é o conhecimento principal a ser testado neste problema.
- Os parâmetros a serem passados em Assembly podem ser por registrador (jeito mais fácil) ou por pilha (mais genérico, porém um pouco mais difícil). Não use variáveis para este fim (porque esse não é o jeito certo).
- O procedimento/função não deve imprimir absolutamente nada. Deve apenas fazer a conta e retornar o resultado para o algoritmo principal. Quaisquer impressões e leitura de valores na tela devem ser feitos exclusivamente pelo algoritmo principal (em C ele é o *main*). Em Assembly é o corpo principal do código, que está fora de outros procedimentos).

[1] Ou Algoritmos e Estruturas de Dados, no caso de eventuais alunos(as) de Eng. de Controle e Automação

UM APÊNDICE...

A essa altura, faltando poucos dias para terminar o período, você pode estar se perguntando: Professor, como eu começo? Pois bem... te darei uma dica de começo e uma sequência de estudos que pode dar certo (baseado em experiências de colegas de vocês anteriormente):

1. Comece pelo começo! Sim. Identifique o começo. E o começo está em ESTUDAR o básico de Assembly x86-16 através dos **slides** e dos **vídeos** postados. O primeiro passo é compreender o que é uma linguagem de montagem e saber que ela abstrai muito pouco da arquitetura da CPU a qual se relaciona. Logo, é impossível aprender Assembly x86 de 16 bits sem conhecer as características principais do Intel 8086, a primeira CPU a implementar essa arquitetura. Para isso, deve ficar claro na sua cabeça:
    i. Os **8 registradores de propósito geral** (todos de 16 bits, daí dizemos que a CPU é de 16 bits, entendeu?): AX, BX, CX, DX, SP, BP, SI, DI. Bem como a forma como AX/BX/CX/DX se dividem em **parte alta** (AH, BH, CH, DH) e **parte baixa** (AL, BL, CL, DL). Esses registradores "parciais" são todos de 8 bits e são parte direta dos maiores, não são registradores "a parte". Por exemplo, ao alterar AL ou AH, você também estará alterando AX.
    ii. **Ignorar**, a princípio, **os registradores de segmento** (CS/DS/ES/SS). Isso porque faremos um programa pequeno, que cabe dentro de um único segmento de 64 KB. Segmentos foram ~~gambiarras~~ recursos técnicos criados pela Intel para que o 8086 enxergasse mais do que 64 KB de memória (Ele enxerga 1 MB no total, como se tivesse registradores de memória de 20 bits).
    iii. Aprender a **acessar a memória** através das variáveis e dos registradores BX, BP, SI, DI. Esses só podem se combinar na forma $\begin{vmatrix} \boldsymbol{BX} \\ ou \\ \boldsymbol{BP} \end{vmatrix} + \begin{vmatrix} \boldsymbol{SI} \\ ou \\ \boldsymbol{DI} \end{vmatrix} + deslocamento$. Isso tudo está explicado nos slides e logo nas primeiras aulas. É **impossível** avançar sem esse conhecimento.
    iv. Aprender o que são as **interrupções** de software que servem quase como "funções básicas" dentro do Assembly x86-16. Através delas imprimimos ou lemos um caractere, por exemplo. São todas muito, muito básicas como tudo em Assembly é. Através desse monte de funcionalidade básica é que construímos algo mais complexo. Todas as possíveis interrupções estão documentadas no arquivo "*8086 bios and dos interrupts (IBM PC).htm*" postado no AVA.
    v. Aprender as **instruções lógicas e aritméticas**. Não se preocupe em decorar todas. Você não fará prova desse assunto. Você pode consultar a qualquer momento que precisar. Por falar em consulta, no AVA há um excelente documento sobre o assunto ("*8086 instructions.htm*") postado no AVA.
    vi. Aprender a construir e usar os **jumps condicionais e incondicionais**. Pois é, não há "IF" em Assembly. Há instruções (especialmente a CMP) que afetam flags e isso é levado em conta pelas diversas instruções de jump condicional.
    vii. Por fim, entender **procedimentos e macros**. Especialmente porque o último problema desse trabalho exige o uso de procedimentos.
2. Com ou sem pandemia, todas as turmas antes de você tiveram pouco tempo. Mas a maioria que persiste e se dedica consegue ir bem. E até gostar... pasme!
3. Concluída a primeira fase de estudos, **instale o emulador** e **rode os três exemplos** que são encontrados no AVA, na seção correspondente a esse trabalho. Rode no modo passo-a-passo entendendo como cada instrução afeta os registradores e o que acontece na tela. Gaste tempo aqui. Compreenda o que está acontecendo. Colegas podem ajudar. O professor também!
4. **Pense em pequenas modificações nos exemplos práticos**. Tente fazer essas modificações funcionarem.
5. Por fim, agora sim: baseado nos exemplos práticos, **comece o trabalho pelo primeiro problema**. Eles estão em ordem crescente de dificuldade. *NÃO DIVIDA OS PROBLEMAS ENTRE OS MEMBROS DO GRUPO*. Além de não ser justo, tornará tudo muito mais difícil.

Bom Trabalho!