Ciências ULisboa | Informática

Arquiteturas de Sistemas Computacionais

Guião de laboratório
x86 – Operações aritméticas e lógicas

1 Exercícios fundamentais

Descarregue o arquivo de suporte a esta aula, `Lab_x86_Operacoes.tar.gz`, que se encontra no moodle junto a este Guião. Guarde o arquivo no diretório de `Downloads` (no caso de o Linux estar em português, o diretório chama-se `Transferências`).

Execute os seguintes comandos no terminal para criar o diretório de trabalho para esta aula, mover o arquivo comprimido para o diretório pretendido e obter os ficheiros de suporte a este guião:

```
$ mkdir -p ~/2023-24/ASC/11-operacoes
$ mv ~/Downloads/Lab_x86_Operacoes.tar.gz ~/2023-24/ASC/11-operacoes
$ cd ~/2023-24/ASC/11-operacoes
$ tar xvzf Lab_x86_Operacoes.tar.gz
$ rm Lab_x86_Operacoes.tar.gz
```

Para a realização deste Guião serão usadas as aplicações contidas no container de ASC. O container de ASC é um ambiente de software que pode ser temporariamente aberto e utilizado proporcionando novas aplicações, ou aplicações mais ajustadas, às necessidades de ASC do que o ambiente proporcionado pelas aplicações instaladas nativamente no Linux.

Neste container encontra-se uma aplicação com interface gráfica (GUI) que é o SASM. Vamos usá-lo para editar, executar e depurar os programas deste Guião.

1.1 *Overflow* na soma de inteiros

Considere o programa `alu_inteiros.asm`. Este programa efetua algumas somas de números inteiros que, potencialmente, podem gerar transbordo (*overflow*).

Corra o programa em modo de depuração e **execute-o passo-a-passo**.

Observe o resultado de cada operação e em particular o valor das *flags* de transporte (CF) e transbordo (OF).

Alguma operação de soma gerou transbordo (*overflow*)? Qual o significado desta ocorrência?

❑ Sim
❑ Não

Ocorreu também transporte (*carry*)? Qual o significado desta ocorrência? O que fazer?

❑ Sim
❑ Não

Se ocorreu transbordo (*overflow*), **faça as alterações necessárias** para que deixe de ocorrer.

1.2 Comparação de valores

Considere o programa `notas.asm`. Este programa compara uma nota obtida com as várias gamas definidas e escreve uma avaliação qualitativa de **Insuficiente**, **Suficiente**, **Bom**, **Muito bom** ou **Excelente**.

Corra o programa em modo de depuração e **execute-o passo-a-passo**.

Observe o resultado de cada comparação e em particular o comportamento das instruções de salto condicional.

Teste o programa com diferentes valores iniciais para a variável `nota`.

Observe que em cada bloco de escrita da mensagem são repetidas 3 instruções …

```
mov rax, 1
mov rdi, 1
…
syscall
```

Este programa pode ser simplificado, colocando estas 3 linhas apenas a seguir ao rótulo _next (antes da chamada ao sistema SYS_EXIT). Desta forma só vai existir uma linha com a chamada ao sistema SYS_WRITE e cada bloco da A estrutura `if-then-elseif-…-else` apenas prepara os dois registos, `rsi` e `rdx`, que definem o endereço da mensagem a escrever e o respetivo tamanho.

Faça esta otimização no código e volte a testar o programa.

Existe outra otimização que pode ser feita ao programa. O teste da expressão é constituído por 3 linhas …

```
    jae _?
    jmp _if?
_?:
```

E estas podem ser reduzidas a <u>apenas uma instrução de salto condicional</u>.
Que instrução de salto condicional usaria em sua substituição? Concretize especificamente o operando que usaria quando esta instrução é colocada logo a seguir `cmp al, EXCEL`.

Desta forma, não estamos a usar exatamente a estrutura de controlo de fluxo tal como definida nas aulas teóricas, mas estamos a adaptá-la de modo a melhorar o tamanho e desempenho do programa.

Faça esta otimização no código e volte a testar o programa.

Consegue identificar alguma outra otimização?

❑ Sim
❑ Não

Existe outra técnica de otimização que poderia ser aqui utilizada e é muito popular que é baseada em vetores indexados pelo valor da nota e que contêm o endereço e o tamanho da mensagem a utilizar. Esta técnica torna o programa mais simples e configurável. Será estudada mais tarde.

1.3 Operações aritméticas de vírgula-flutuante

Considere o programa `fpu_vflutuante.asm`. Este programa calcula a circunferência de um círculo com raio de 5 cm usando valores representados em IEEE 754 com precisão simples (32 bits) e as operações da FPU. A fórmula de cálculo da circunferência de um círculo é $C = 2 \times \pi \times r$, onde r é o raio do círculo. (O código completo pode ser encontrado no ficheiro fornecido).

Corra o programa em modo de depuração e **execute-o passo-a-passo**.

Observe o conteúdo das variáveis `PI_single` e `r`. Contêm os valores esperados? Escreva abaixo os cálculos que fez para confirmar que correspondem aos valores iniciais definidos no programa.

Continue a execução até ao rótulo `_fim`.

Qual o resultado obtido para a circunferência do círculo? Como se comparam os valores contidos, no final do programa, nas variáveis `Resultado_single` e `Resultado`?

Como poderia evitar a conversão realizada no início do programa e em seu lugar apenas carregar diretamente o valor da variável `r` para o registo XMM0?

Adicione a este programa as instruções de controlo de fluxo para armazenar o resultado inteiro só quando o valor da variável booleana *arredondar* é 1.

Corra novamente o programa em modo de depuração, **execute-o passo-a-passo** e confirme que a alteração foi feita corretamente.

Modifique o programa `vflutuante.asm` para este calcular a área do círculo, ao invés da circunferência. Lembre-se que a área de um círculo é dada pela fórmula $A = \pi \times r^2$.

Corra este novo programa em modo de depuração, **execute-o passo-a-passo** e confirme que a alteração foi feita corretamente.

1.4 Operações lógicas com máscaras

Considere o programa `mascaras.asm`. Este programa realiza várias operações lógicas com máscaras de modo a testar, ativar, desativar e trocar determinados bits de uma variável `ascii` que contém um caractere ASCII.

Corra o programa em modo de depuração e **execute-o passo-a-passo**.

Altere as máscaras definidas na secção `.data` de modo a conseguir trocar o conteúdo da variável `ascii` para os seguintes valores e indique que máscaras usou para cada caso:

Novo valor de ascii	Máscara de Teste	Máscara Ativar	Máscara Desativas	Máscara Trocar
"A"				
"b"				
"Z"				
"9"				

Conseguiu usar todas as máscaras para alcançar os resultados? Se não conseguiu então escolha um dos valores e tente agora usar as máscaras que ainda não usou para alcançar o resultado.

Caso não tenha conseguido obter algum dos valores pretendidos, execute o programa passo-a-passo e observe o resultado das operações lógicas e compare o byte obtido com os códigos da tabela ASCII.

1.5 Escrita de dados de 32 bits na consola em hexadecimal

Considere o programa `hex2ascii.asm`. Este programa escreve na consola a representação hexadecimal de um registo ou variável de, até, 32 bits. Isto é feito convertendo cada conjunto de 4 bits (algarismo hexadecimal) para a sua representação em ASCII. A técnica usada baseia-se na consulta a um vetor de caracteres "hexadecimais" usando indexação.

Nota: Esta técnica de indexação foi mencionada no final do exercício de Comparação de valores deste Guião. A técnica é usada neste exercício, mas apenas será estudada mais tarde.

Corra o programa em modo de depuração e **execute-o passo-a-passo**.

Teste o programa com diferentes valores iniciais para a variável `valor`.

Agora verifique que também pode usar esta técnica para observar os endereços em que se encontram os dados. Na linha 13, **altere** o operando que obtém o conteúdo da variável `valor` para passar o obter o endereço das seguintes variáveis:

- `hex_lut`
- `msg`
- `lf`
- `valor`

Compare as diferenças entre os endereços de `msg` e `lf`, e também entre os endereços de `lf` e `valor`. Porquê estas diferenças?

2 Exercícios Complementares

1. Considerando o programa contido no ficheiro `alu_add.asm` (o código completo pode ser encontrado no ficheiro fornecido):

a) Indique o endereço de memória correspondente a cada uma das variáveis X, Y e Z, e o valor em hexadecimal do conteúdo da memória nesses endereços no início da execução.

b) Indique agora o valor em hexadecimal do conteúdo das variáveis X, Y e Z após a execução da última instrução do programa acima.

c) Coloque pontos de paragem (*breakpoints*) após a execução de cada instrução aritmética, isto é, nas instruções imediatamente a seguir a cada instrução. Foram levantadas *flags*? Quais?

d) Substitua a primeira instrução *nop* por instruções de salto para desviar o fluxo normal da execução e terminar o programa com o rótulo *_fim* no caso de a flag CF ser levantada. Agora defina o valor mínimo na variável *X* que faria com que a flag CF fosse levantada após a execução da instrução de adição, volte a fazer Debug e verifique se as instruções de salto inseridas terminam efetivamente o programa.

e) Reponha a variável *X* para o valor original. Substitua a segunda e a terceira instruções *nop* por instruções de controlo de fluxo para ir para o rótulo ***_ajuste*** só se a flag OF ser levantada após a instrução de multiplicação.

2. Considere o programa `empacota.asm` que codifica informação sobre um par de registos a usar numa operação hipotética de transferência de dados. Um dos registos assume o papel de origem dos dados enquanto o outro assume o papel de destino dos dados. Estes registos são designados através de um identificador, de `0` a `15`, que correspondem, respetivamente aos registos:

```
rax,rbx,rcx,rdx,rsi,rdi,rbp,rsp,r8 ,r9 ,r10,r11,r12,r13,r14,r15
```

Assim o registo `rax` é designado pelo identificador `0` enquanto `rsp` é designado por `7`.

Definindo o par de registos, `Origem` e `Destino`, cada um na sua variável do tipo byte, o programa empacota esta informação num único byte, `OrigemDestino`. Como demonstração, a informação volta a ser desempacotada e os nomes dos registos são escritos na consola.

a) Experimente o programa e verifique que são escritos na consola os nomes dos registos identificados nas variáveis `Origem` e `Destino`.

b) Alterando apenas o conteúdo das variáveis `Origem` e `Destino`, faça com que o programa escreva `Origem: rdi` e `Destino: r10`. Quais os valores que colocou nas variáveis?

c) Troque a instrução `nop` por uma outra que troque as posições de origem e destino dentro da variável `OrigemDestino`, i.e. nos bits menos significativos passe a ficar o destino e vice-versa.

d) Esta operação de empacotamento reduziu a necessidade de armazenamento para metade. É possível compreender o impacto desta técnica em volumes grandes de dados que possam ser empacotados. No entanto, também existiu um custo. Qual foi? Esta técnica é sempre vantajosa?

3 Material de Apoio

Manual do Netwide Assembler (NASM): http://www.nasm.us/doc/

Intel Assembler 80x86 CodeTable (referência rápida): http://www.jegerlehner.ch/intel/IntelCodeTable.pdf