Sistemas Digitais **Avaliação 4 – Assembler**

Nome: Mariana da Costa Zatta RA: 2079950

Nome: Kevin Azevedo Lopes RA: 2079909

1. Montar uma tabela, com duas colunas, estando o código em C `hello.c` a esquerda e o código em Assembler `hello.asm` a direita.

**#include <stdio.h>
#include<stdlib.h>
 	
int main()
 	
printf("Hello World!\n\n");
 	
return_EXITSUCCESS;**
	**LF equ 10
	EXIT_SUCCESS equ 0
	STDOUT equ 1
	SYS_WRITE equ 1
	SYS_EXIT equ 60
	
	section .text
	global _start
	
	_start:
	
	mov rax, SYS_WRITE
	mov rdi, STDOUT
	lea rsi, [msg]
	mov rdx, msglen
	syscall
	
	mov rax, SYS_EXIT
	xor rdi, rdi
	syscall
	
	section .rodata
	msg: db"Hello, World!", LF,
	LF
	msglen: eq $ - msg**

2. Comparar os códigos, e descrever o funcionamento em C e o equivalente em Assembler.

A forma como o programa objeto é gerada entre a linguagem C e o assembler são diferentes, em assembler o processo usado é a montagem, onde cada instrução (nmeumonico) é traduzida em uma instrução de linguagem de máquina, tendo como única vantagem não escrever diretamente em binário, mas o programador terá que usar e somente usar instruções disponíveis pelo processador alvo. Já na linguagem C, onde é utilizada uma linguagem compilada, um programa (compilador) surge como intermediador entre a linguagem da máquina e uma linguagem de alto nível e fácil abstração.

3. O comando `syscall` é uma instrução do Assembler 8086? Explicar o seu funcionamento para o S.O. Linux.

Syscall é uma chamada de sistema no Linux, onde o núcleo do sistema operacional irá executar o código. A chamada não recebe nenhum operando e a especificação de qual código irá executar e com quais argumentos, é definido por uma convenção de chamada assim como no caso das funções. A convenção para efetuar uma chamada de sistema em Linux x86-64 é bem simples, precisa-se definir RAX para o número da syscall que deseja-se executar e outros 6 registradores são usados para passar argumentos. O retorno da syscall também fica em RAX assim como na convenção de chamada da linguagem C.

4. Qual a importância de se usar o Assembler para se programar o Kernel do Linux?

O Assembly tem sua importância ao se programar o Kernel do Linux devido ao fato de ser uma linguagem que exige bem menos memória. Além disso, o Assembly dá acesso direto ao Hardware e executa suas funções com agilidade. Assim, o sistema operacional fica leve e rápido.

5. Quais são as desvantagens de se usar Assembler?

- **Tempo de desenvolvimento: escrever código em linguagem assembly leva muito mais tempo do que em uma linguagem de alto nível.**
- **Confiabilidade e segurança: É fácil cometer erros no código assembly. Existem tantas possibilidades de erros ocultos no código assembly que isso afeta a confiabilidade e a segurança do projeto, a menos que você tenha uma abordagem muito sistemática para teste e verificação.**
- **Depuração e verificação: o código assembly é mais difícil de depurar e verificar porque há mais possibilidades de erros do que no código de alto nível.**
- **Portabilidade: o código de montagem é muito específico da plataforma. A migração para uma plataforma diferente é difícil. A linguagem é portável apenas dentro de uma família de processadores.**

6. Qual é a aplicação da instrução `LEA`?

A instrução LEA coloca o *endereço* especificado pelo seu primeiro operando no registro especificado pelo seu segundo operando, Isso é útil para obter um ponteiro em uma região de memória ou para realizar operações aritméticas simples.

7. Descrever o funcionamento das instruções `CALL` e `RET` citados no artigo.

A instrução Call coloca o endereço de retorno na "pilha", isso permite que a instrução RET retorne a esse endereço.

8. Explicar o processo de compilação e geração de executável.

O processo de programação inicia-se com a edição de um programa-fonte e termina com a geração de um programa executável. A compilação é como um corretor de texto de várias dimensões, identificando falhas na estrutura de funcionamento do código, e quando totalmente compilado, significa que não há erros de concordância, no caso do assembler, sua estrutura da linguagem monta o código binário das funções definidas no texto em assembly, gerando o executável, que se comunica com as bibliotecas e realiza os processos definidos no programa em assembly, o executável é o programa que irá rodar na estrutura física do processador.

9. Por que é necessário usar um `shell script`?

Um shell script possibilita encadear comandos para solucionar tarefas complexas, como automatizar backups, limpar erros de um arquivo de texto, etc. Ele é o interpretador de comandos do Linux e realiza a execução do código Assembler, onde são pré definidos processos de montagem de arquivos e programas, automatizando partes do processo, fazendo com que o desempenho de acesso às bibliotecas seja melhorado, e também diminua o tempo de execução.

10. Qual a diferença entre o registrador `AX` e `RAX`?

AX é o registrador de tamanho "curto" de 16 bits. Foi adicionado em 1979 com a CPU 8086, mas é usado no código DOS ou BIOS até hoje. Acumulador, armazena operandos e resultados dos cálculos aritméticos e lógicos.

RAX é o registrador de tamanho "longo" de 64 bits. Foi adicionado em 2003 durante a transição para processadores de 64 bits. É possível armazenar valores para realizar cálculos aritméticos, somar, multiplicar e dividir.

11. Que utilitário o autor utiliza para depurar o programa em Assembler? Faça um resumo explicativo deste utilitário.

DDD debugger, depurador com apresentação de dados. Permite associar o programa sendo depurado com seu código-fonte, além de poder mostrar interativamente suas estruturas de dados como grafos. Durante a operação de depuração e execução do Assembly, é possível parar o processamento em um determinado momento e ver os valores obtidos nas variáveis e analisar o funcionamento do código, algo parecido é possível ser realizado em softwares como o matlab.