

TPC4

Resultados dos exercícios propostos

1. Formato dos ficheiros `prog.c` e `soma.c`

Os ficheiros estão no formato "texto". Usando o comando (hex dump) `xxd soma.c`:

```
00000000: 696e 7420 6163 6375 6d3d 303b 0a76 6f69  int accum=0;.voi
00000010: 6420 736f 6d61 2028 696e 7420 7029 0a7b  d soma (int p).{
00000020: 0a09 6163 6375 6d20 2b3d 2070 3b0a 7d0a  ..accum += p;..}
```

Cada um dos *bytes* do ficheiro representa um carácter codificado em ASCII e termina normalmente em Linux com o carácter mudança de linha (ASCII `0x0a`, mas depende do editor de texto usado).

2. Tamanho dos ficheiros `prog.c` e `soma.c`

Basta contar os caracteres introduzidos (não esquecer o carácter de mudança de linha) para obter uma estimativa do tamanho do ficheiro. Existem vários comandos do Linux (por ex., `ls -l` ou `wc`) para confirmar o tamanho exato do ficheiro.

3. Formato do ficheiro `soma.s`

Tal como o ficheiro fonte que lhe deu origem, este continua no formato "texto". O facto de conter código *assembly* pelo meio, é uma interpretação nossa do significado do texto nele contido.

4. Informação para além de puro código *assembly* no ficheiro `soma.s`

Viu-se em aulas anteriores que o código *assembly* é constituído por um conjunto de mnemónicas que identificam as operações que a PU executa (por ex., `mov`, `sub`, `and`, `push`) seguida da identificação dos operandos fonte e/ou destino (seu valor ou localização). Os valores são normalmente numéricos, e as localizações são números (endereços de memória), nomes de registos (começados por "%" na notação da GNU), ou especificação de endereços de memória usando constantes e 1 a 2 nomes de registos entre parênteses.

Assim, todo o texto extra neste ficheiro é informação pertinente para o *assembler*.

Estão presentes pelo menos 3 tipos de informação extra:

- linhas que começam com ".", que são diretivas (comandos) para o *assembler*, quando for montar o código binário; especificam, por exemplo, onde começa o bloco de informação contendo o código do módulo (".text") ou contendo as variáveis globais (".data"), ou ainda que uma determinada sequência de caracteres é única no programa e foi definida neste módulo, podendo ser acedida a partir de qualquer outro módulo que seja posteriormente ligado a este e que lhe faça referência; por ex. ".global" seguido do nome de uma função codificada no ficheiro original em C que deu origem a este ficheiro, ou de uma variável global declarada no mesmo ficheiro;
- linhas que terminam em ":" são etiquetas (*labels*) que indicam a localização (i) dentro do bloco .text de um dado pedaço de código ou (ii) dentro do bloco com as variáveis globais, de uma dada variável (cujo nome está imediatamente antes dos ":");
- sequências de caracteres no meio do código *assembly*, associadas normalmente ao nome de uma variável ou função, indicando a localização em memória onde ela irá tomar os diversos valores ao longo da execução de um programa (as "etiquetas" referidas antes).

5. Nível de abstração do ficheiro `soma.s`

Este é ainda um ficheiro de texto, no nível *assembly*, ainda longe de poder ser interpretado pela unidade de descodificação de qualquer processador.

6. Visualização de parte do conteúdo do ficheiro `soma.o`

Este é um ficheiro objeto, em que parte do seu conteúdo corresponde ao código máquina de `soma`. Com o comando `(gdb) x/23xb soma` o que se pretende é ver, em hexadecimal, o conteúdo dos 23 bytes do bloco `.text` a partir do local onde estava colocada a etiqueta com o nome da função `soma`, i.e., os 23 bytes iniciais com o código em linguagem máquina da função `soma`. Contudo, aparece uma mensagem de aviso de algo inesperado a partir de certa posição, devido ao facto de o código máquina da função `soma` não ter 23 bytes.

7. Nível de abstração do ficheiro `soma.o`

Tal como escrito antes, este é um ficheiro objeto, em que parte do seu conteúdo é já o código do programa em linguagem máquina do IA-32. Contudo, olhando para os endereços de cada uma das instruções (que começam no endereço "0") pode-se concluir que este código ainda não está pronto para ser executado (para além de se saber que lhe falta ainda o `main`).

8. Informação simbólica em `soma.o`

Este ficheiro tem informação simbólica (`.text` e/ou `.data`) para além do código máquina da função `soma`. Além disso, a informação está incompleta: o endereço da função `soma` é 0, assim como o endereço da variável `accum` também é 0. A tabela de símbolos neste ficheiro pode ser visualizada com `objdump -t soma.o`.

9. Representação da variável `accum`

Esta variável deixou de aparecer como uma *"string"*: já foi convertida no número "0". No entanto o seu endereço ainda terá que ser redefinido, tal como pode ser visto usando `objdump -dr soma.o`. (2: R_386_32 accum e c: R_386_32 accum indica que os valores do executável na posição 2 e c terão que ser redefinidos).

10. Análise do código em `soma.o`

Na sequência do comando de `objdump` executado (com a opção `-d`, de "desmontagem") é possível verificar quantos bytes ocupa o código desta função em linguagem máquina, como está codificada cada uma das instruções e se cada uma destas ocupa 1, 2, 3 ou 5 bytes.

11. Formato do ficheiro `prog`

Este ficheiro é executável, pois contém toda a informação necessária: código em linguagem máquina de todo o programa, incluindo o código das funções de bibliotecas do sistema operativo (e do C e outros), e os valores inicializados das variáveis globais.

12. Localização da variável `accum` no ficheiro `prog.dump`

Esta variável deixou de aparecer localizada no endereço "0" bem como o do código da função `soma`. Estão já em endereços distintos, como seria de esperar.

13. Representação em memória do endereço de `accum`

Analisando lado a lado o código binário e o código *assembly* da função verifica-se como o endereço de `accum` aparece no formato de instrução: o seu byte menos significativo está no endereço de menor valor. O que mostra que a implementação desta arquitetura é *little endian*.

14. Invocação da função `soma` pela `main`

No código da função `main`, encontra-se a instrução de `call` seguida de um endereço de memória, que corresponde ao início da função `soma`. Contudo, o valor que foi colocado no código máquina (9 em *little endian*) dá-nos, não o endereço, mas a distância a que o código da função `soma` se encontra o valor do IP aquando da execução da instrução `call`; é, pois, um modo relativo (ao IP) de indicar um endereço.