

 Universidade do Minho	Arquitectura de Computadores II L.E.S.I. – 3º ano	
	Módulo nº 1 A arquitectura Y86	

1. Introdução

Pretende-se com este módulo:

- fazer uma revisão das técnicas de programação em *assembly*;
- familiarizar os alunos com o conjunto de instruções Y86.

No final deste módulo os alunos deverão ser capazes de:

- converter pequenos segmentos de código expresso numa linguagem de alto nível (C) para *assembly* do IA32 ou Y86
- identificar as diferenças entre os conjuntos de instruções do IA32 e Y86, convertendo código *assembly* de um conjunto para o outro
- utilizar as ferramentas associadas ao Y86, nomeadamente, o *assembler yas* e o simulador *yis*.

2. Material de apoio

A bibliografia relevante para este módulo é a secção 4.1 do livro “Computer Systems: a Programmer’s Perspective”, de Randal E. Bryant e David O’Hallaron.

Associado a este livro é mantido pelos autores um web site com material de apoio: <http://csapp.cs.cmu.edu/>

Em anexo a este módulo encontra-se uma breve descrição da arquitectura do Y86.

O conjunto de ferramentas de apoio a este e próximos módulos, incluindo o *assembler* do Y86, *yas*, e o simulador do conjunto de instruções, *yis*, podem ser carregados a partir do web site do livro, no endereço <http://csapp.cs.cmu.edu/public/students.html>

Estão disponíveis as ferramentas em formato binário e também o código-fonte. Para este módulo são suficientes os programas *yas* e *yis*. No entanto, para os próximos módulos serão necessárias todas as ferramentas, pelo que se recomenda a sua instalação completa. Esta instalação necessita do tcl/tk. Se não os tiver correctamente instalados na sua máquina, pode encontrá-los em <http://tcl.sourceforge.net/>.

3. Caso de estudo: Acumulação dos valores de um vector

3.1 Assembly IA32

Gere o código *assembly* para o IA 32 do código C apresentado na tabela 1. Utilize o comando

```
gcc -c -S -O1 soma.c
```

soma.c
<pre>int soma (int *Start, int n) { int res=0; while (n) { res += (*Start); Start++; n--; } return (res); }</pre>

Tabela 1 – Soma dos elementos de um vector

Analise cuidadosamente o código apresentado no ficheiro soma.s. Identifique a função de cada instrução, bem como o registo ou endereço de armazenamento de cada variável.

3.2 Conversão para Y86

Converta manualmente o código em *assembly* do IA32 para *assembly* do Y86, explicando as diferenças que for encontrando.

3.3 Código máquina Y86

O código *assembly* deve agora ser convertido para código máquina. O *assembler yas* faz essa conversão gerando um ficheiro de texto¹ que contem os endereços das posições de memória das instruções e dados globais, contem as instruções máquina representadas em hexadecimal e contem ainda comentários que aumentam a legibilidade do ficheiro objecto.

Para que possamos simular a execução desta função falta ainda definir um ponto de entrada no programa e reservar espaço para o vector e a pilha. O ponto de entrada num programa em C é a função `main()`. No simulador do Y86 o ponto de entrada é identificado pela etiqueta `init` e deve corresponder ao endereço `0x00`. O endereço de memória a partir do qual o código é carregado é definido com a directiva `.pos`. A tabela 2 apresenta o código *assembly* Y86 quase completo, com um ponto de entrada, função `main()`, espaço para dados (correspondente ao nosso vector com 4 elementos) e espaço para a pilha. Complete-o inserindo o código de `soma()`.

¹ O ficheiro com um programa em linguagem máquina é tipicamente um ficheiro binário. Neste caso em particular estamos a lidar com um simulador de um processador. Este simulador lê o código máquina de um ficheiro em formato de texto (ASCII).

Y86
<pre> # Execução começa no endereço 0 .pos 0 init: irmovl Stack, %esp # iniciar pilha irmovl Stack, %ebp jmp main # Vector com 4 elementos .align 4 vector: .long 100 .long 25 .long -200 .long 125 main: irmovl \$4, %eax pushl %eax # 1º parâmetro irmovl vector, %eax pushl %eax # 2º parâmetro call Sum halt Sum: ... inserir código .pos 0x100 Stack: # Início da pilha </pre>

Tabela 2 – Programa completo para o Y86

Gere o código máquina usando o comando `yas soma.y86`

Analise cuidadosamente o conteúdo do ficheiro `soma.o` gerado no último passo. Responda às seguintes questões, lembrando-se que a representação das palavras em memória é *little endian*:

- Qual o endereço da posição de memória onde começa a pilha?
- Qual o endereço da posição de memória onde começa o armazenamento do elemento do vector com valor -200 e índice 2? Como é que o valor -200 está representado em memória?
- Qual o endereço correspondente à etiqueta Loop? Explique a codificação da instrução `jne Loop`.

3.4 Execução do código

O simulador `y86` permite executar o código gerado indicando:

- quantas instruções foram executadas, qual o valor do PC quando a execução terminou e a razão pela qual terminou;
- o valor dos códigos de condição no fim da execução;
- os registos alterados pelo programa, assim como os seus valores iniciais e finais;

- as posições de memória alteradas pelo programa, assim como os seus valores iniciais e finais.

Execute o comando `yis soma.yo`

Verifique, justificando:

- o valor final dos códigos de condição;
- os endereços e valores das posições de memória alteradas;
- a correcção do programa indicando onde se encontra o resultado da soma de todos os elementos do vector.

4. Exercícios adicionais

1. Repita todos os passos seguidos ao longo das secções anteriores para a seguinte função:

<code>conta_rec.c</code>
<pre>int conta (int *Start, int t, int n) { if (n<=0) return (0); else return ((*Start > t ? 1 : 0) + conta (Start+1, t, n-1)); }</pre>

2. O modo de endereçamento de memória do IA32 tem 4 argumentos: registo base (r_b), registo índice (r_i), factor de escala (s) e deslocamento imediato (d). O endereço final é dado por $r_b + r_i * s + d$. O Y86 apenas suporta o endereço base + deslocamento imediato. Converta a seguinte instrução para *assembly* do Y86

```
movl $12(%ebx, %esi, 4), %edx
```

3. Repita todos os passos seguidos ao longo das secções anteriores para a seguinte função:

<code>soma_idades.c</code>
<pre>typedef struct { int num, idade; } MEMBRO; int soma_idades (MEMBRO *Start, int n) { int soma=0; while (n) { soma += Start->idade; n--; Start++; } return (soma); }</pre>