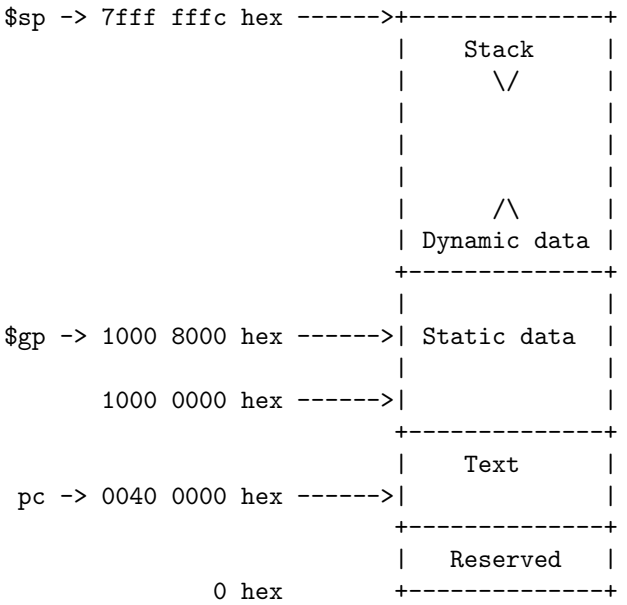
Montador e sua relação com a Memória

Servente

16 de outubro de 2015

pg 118 Uma variável C é genericamente uma localização na memória, e sua interpretação depende de seu tipo classe de armazenamento. Exemplos incluem inteiros, caracteres, etc. C tem duas classes de armazenamento: automático

pg 121 **Alocando Espaço para Novos Dados na Heap**: Quando adicionamos variáveis automáticas que são locais no escopo de procedimentos, os programadores C precisarão de espaço em memória para variáveis estáticas e para estruturas de dados dinâmicos. **A pilha** começa no maior endereço e aumenta em direção ao menor endereço. A primeira



Imediatamente acima da área estática é a área dinâmica de dados. Essa área, pelo próprio nome já dá para entender, implica dados alocados pelo programa em tempo de execução. Um programa.c a rotina `malloc` usa essa área. Por

causa que segmento de dados começa muito longe do programa, no endereço 10000000 hex, instruções `load` e `store` não podem referenciar diretamente objetos de dados com seus 16-bit de offset.

Como a flecha para cima (`/\`) indica, `malloc` expande área dinâmica de dados com a chamada de sistema `sbrk`, a qual diz para o sistema operacional adicionar mais páginas para o espaço virtual do programa, imediatamente acima do espaço dinâmico de dados.

Por exemplo, para carrega o valor do endereço de dados 10010020 hex em um registrador $v0 requer duas instruções:

```
lui $s0, 0x1001      # 0x1001 significa 1001 base 16
lw  $v0, 0x0020($s0) # 0x10010000 + 0x0020 = 0x10010020
```

139 O compilador transforma código C em linguagem de montagem, uma forma simbólica para que a máquina possa processar. A máquina entende linguagem de montagem, não entende código em alto nível. Nós utilizamos código de alto nível para obter maior rendimento e precisão em desenvolvimento de software.

140 A transformação hierárquica para um programa em C acontece da seguinte forma: Um código.c passa pelo compilador e vira um programa em linguagem de montagem que passa então pelo montador e vira uma objeto, um módulo em linguagem de máquina juntamente a esse objeto temos outros objetos da biblioteca local (já compiladas e montadas) então o ligador associa ambas e vira um executável em linguagem de máquina e então passa pelo carregador e aí então vai para a memória.

141 O montador traduz pseudo-instruções para instruções válidas pelo sistema como: `move $t0, $t1` vira `add $t0, $zero, $t1`. O assembler MIPS também converte `blt` em duas instruções **slt** e **bne**. Traduz também: branch para uma distância muito longe em um par de branch mais jump para poder saltar longas distâncias. Pseudo-instruções dão uma flexibilidade a mais para quem precisa realmente programar em linguagem de montagem. O arquivo objeto para uma sistema UNIX, contem tipicamente:

(a) Um cabeçalho (tamanhos e posições).

(b) Um segmento de texto (programa, lógica).

(c) Uma área de dados estáticos (dados para o programa).

(d) Informações de realocação (identificadores das palavras de dados que dependem de endereço absoluto).

(e) Uma tabela de símbolos (rótulos restantes que não foram definidos).

(f) Informações de deputação (contém informações de como foi compilado, para o depurador associar instruções de máquina com `código.c`).

142 O ligador faz os seguintes passos:

(a) Insere simbolicamente na memória, módulos de código e dados.

(b) Determina endereços de rótulos de dados e rótulos de instruções.

(c) Empacota endereços externos mais endereços internos.

O ligador usa informação de realocação e a tabela de símbolos em cada módulo objeto para resolver todas as referências indefinidas. Se todas as referências externas foram resolvidas, então o ligador poderá determinar locais na memória que cada módulo irá ocupar. O ligador produz então o arquivo executável que pode rodar no computador. Linguagem de montagem é linguagem de programação, a diferença entre ela e as linguagens de alto nível como java e c é que assembler provê apenas poucos tipos de dados e de controle de fluxo.

O objeto é a seguinte estrutura:

Cabeçalho	Texto	Dados	Realocação	Tabela de Símbolos	Depuração

Tabela 1: arquivo.o

O carregador faz os seguintes passos:

(a) Le as informações no arquivo executável para determinar o tamanho dos segmentos de dados e texto.

(b) Cria um novo espaço de endereçamento para o programa.

(c) Copia as instruções e os dados do arquivo executável para o novo espaço de endereçamento.

(d) Copia argumentos passados para o programa para dentro da pilha.

(e) Inicializa os registradores da máquina (Java não usa registradores, por isso é que Java não inicializa as variáveis automaticamente, já C faz isso).

(f) Pula para uma rotina de inicialização

~~**2.21.1**~~ → **2.21.1a** Traduzir o código em C abaixo para linguagem de montagem, respeitando as convenções e padrões, etc.

Código em linguagem C—alto nível:

```
int my_global = 100;
main() {
  int x = 10;
  int y = 20;
  int z;
  z = my_function(x, y)
}

int my_function(int x, int y) {
  return x - y + my_global;
}
```

Tradução para MIPS—linguagem de montagem:

```
my_global: .word 100

main: addi $t0, $0, 10
      addi $t1, $0, 20
      addi $t3, $0, 0

      addi $a0, $0, $t0
      addi $a1, $0, $t1           # (a0, a1) <-- (x, y)

      jal my_function             # salta para rotina folha
      addi $t3, $0, $v0

my_function: addi $sp, $sp, -12
             sw   $ra, 0($sp)
             sw   $s0, 4($sp)
             sw   $s1, 8($sp)     # salva contexto

             la $s1, my_global
             lw $s1, 0($s1)       # busca my_global

             sub $s0, $a0, $a1    # x - y
             add $v0, $s0, $s1    # x - y + my_global

             lw   $s1, 8($sp)     # restaura contexto
             lw   $s0, 4($sp)
             lw   $ra, 0($sp)
             addi $sp, $sp, 12

             jr $ra
```

~~2.21.2~~ Acho que não preciso fazer essa questão (olhando na quarta edição revisada, pode ser que na edição não revisada seja uma questão diferente).

~~2.30.1~~ Na edição 4 revisada essa questão é uma de traduzir pseudo instruções.