Organização do Programa

Universidade Federal do Ceará - Campus Quixadá

Roberto Cabral
rbcabral@ufc.br

16 de Dezembro de 2020

Arquitetura e Organização de Computadores I

Organização do Programa

- Os programas escritos em C são organizados em funções.
- Cada função possui um nome exclusivo no programa. A execução do programa começa com a função chamada “main”.
- Exemplo função principal que não faz nada.

```c
/* doNothingProg1.c
 * The minimum components of a C program.
 * 2017-09-29: Bob Plantz
 */

int main(void)
{
  return 0;
}
```

Compilação

- **Preprocessamento** - processa as diretivas de pré-processamento. A compilação pode ser interrompida no final da fase de pré-processamento com a opção de linha de comando -E, que grava o código-fonte C resultante na saída padrão.
- **Compilador** - Traduz o código-fonte resultante do pré-processamento para a linguagem assembly. A compilação pode ser interrompida no final da fase de compilação com a opção de linha de comando -S, que grava o código-fonte da linguagem assembly em `filename.s`.

Compilação

- **Montador** - Tradução da linguagem assembly para código de máquina invocando o programa `as`. A compilação pode ser interrompida no final da fase de montagem com a opção de linha de comando -c, que grava o código de máquina em `filename.o`.
- **Lincador** - Combinar o código de máquina que resulta da fase de montagem com outro código de máquina das bibliotecas C padrão e outros módulos de código de máquina, e resolver os endereços. Isso é feito invocando o programa `ld`. O nome do arquivo executável pode ser especificado com a opção de linha de comando -o.

Gerando código Assembly

```
@ doNothingProg2.s
@ Minimum components of a C program, in assembly language.
@ 2017-09-29: Bob Plantz

@ Define my Raspberry Pi
        .cpu    cortex-a53
        .fpu    neon-fp-armv8
        .syntax unified         @ modern syntax

@ Program code
        .text
        .align  2
        .global main
        .type   main, %function
main:
        str     fp, [sp, -4]!   @ save caller frame pointer
        add     fp, sp, 0       @ establish our frame pointer

        mov     r3, 0           @ return 0;
        mov     r0, r3          @ return values go in r0

        sub     sp, fp, 0       @ restore stack pointer
        ldr     fp, [sp], 4     @ restore caller's frame pointer
        bx      lr              @ back to caller
```

Gerando código Assembly

- **Linguagem Assembly** - Um conjunto de mnemônicos que têm uma correspondência um a um com a linguagem de máquina.
- **Mnemônicos** - Um pequeno grupo de caracteres semelhantes ao inglês que sugere a ação da instrução.
- Exemplo
  - `0xe3a03000`
  - `mov r3, 0`
- A rigor, os mnemônicos são completamente arbitrários, desde que você tenha um programa assembler que os traduzirá nas instruções de máquina desejadas. No entanto, a maioria dos programas assembler usam mais ou menos os mnemônicos usados nos manuais fornecidos pelos fornecedores de CPU.

Linguagem Assembly

- A linguagem assembly é orientada por linhas.
- Isto é, há apenas uma instrução em linguagem assembly em cada linha e nenhuma das instruções se estende por mais de uma linha.
- Cada linha é organizada em quatro possíveis campos:

```
label:  operation operand(s) @ comment
```

- **Label** - permite dar um nome simbólico a qualquer linha do programa. Uma vez que cada linha corresponde a um local da memória no programa, outras partes do programa podem então referir-se ao local da memória pelo nome.

Linguagem Assembly

- **operation** - existem dois tipos de operações:
  - Um **mnemônico em linguagem assembly** é traduzido em instruções de máquina reais, que são copiadas para a memória quando o programa está para ser executado. Cada instrução de máquina ocupará quatro bytes de memória.
  - Uma **diretiva assembler** ou **pseudo operação** começa com o caractere ponto final (“.”). Eles são usados para direcionar a maneira pela qual o montador traduz o arquivo. Eles não se traduzem diretamente em instruções de máquina, embora alguns façam com que a memória seja alocada.

Linguagem Assembly

- **Operand(s)** - especifica os argumentos a serem usados pela operação. Os argumentos são especificados de várias maneiras diferentes:
  - Um nome que tem significado para o montador, por exemplo, o nome de um registro.
  - Um nome que é criado pelo programador, por exemplo, o nome de uma variável ou constante.
  - Um valor literal ou explícito, por exemplo, o inteiro 75.
- Uma instrução ARM pode ter de zero a três operandos. Todos, exceto a instrução de armazenamento, são especificados nessa ordem:

operation [destination[, source1[, source2[, source3]]]]

Linguagem Assembly

- **Comment** - é todo o texto imediatamente após um caractere “@”. Como a linguagem assembly não é tão fácil de ler quanto as linguagens de alto nível, bons programadores colocarão um comentário em quase todas as linhas.
- As regras para criação de identificadores são similares as do C.

Linguagem Assembly

- a diretiva assembler `.text` direciona o montador para colocar o que vem a seguir na seção de texto.
- O que significa "seção de texto"?
  - Quando um arquivo de código-fonte é traduzido em código de máquina, é produzido um arquivo-objeto, que possui um formato específico.
  - Em nosso ambiente, a organização do arquivo objeto segue o Executable and Linking Format (ELF).
  - O padrão ELF especifica muitos tipos diferentes de seções, cada uma dependendo do tipo de informação armazenada nela.

Linguagem Assembly

- GNU / Linux divide a memória em diferentes segmentos para propósitos específicos quando um programa é carregado do disco. As quatro categorias gerais são:
  - **Segmento de Texto** - onde as instruções do programa e os dados constantes são armazenados.
  - **Segmento de Dados** - onde as variáveis globais e as variáveis locais estáticas são armazenadas.
  - **Segmento de Pilha** - onde as variáveis locais e os dados que vinculam as funções são armazenados.
  - **Segmento de Heap** - local de memória disponível quando um programa C faz alocação dinâmica.

Linguagem Assembly

- A diretiva assembler `.align 2` faz com que o montador ARM garanta que os dois bits mais baixos do endereço inicial do código gerado sejam zero.
- Ou seja, o endereçamento é ajustado, incrementado se necessário, para ser um múltiplo de quatro.
- Cada instrução de máquina tem quatro bytes de comprimento, portanto, isso garante o alinhamento adequado das instruções na memória.

Organização do Programa

Universidade Federal do Ceará - Campus Quixadá

Roberto Cabral
rbcabral@ufc.br

16 de Dezembro de 2020

Arquitetura e Organização de Computadores I