Linguagem de Montagem

Controle de Fluxo de Execução
Aula 07

Edmar André Bellorini

Ano letivo 2016

Execução TOP → DOWN

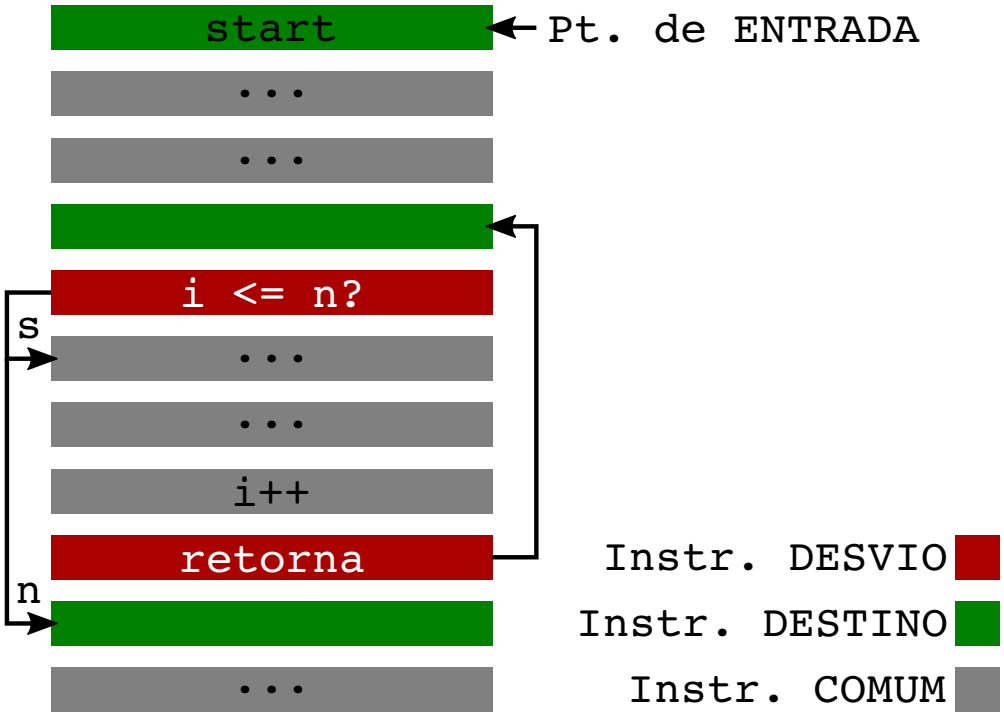
```
1     ...
2   section .data
3     fileName: db "a06e02.txt"
4
5   section .bss
6     texto: resb 25
7     fileHandle: resd 1
8
9   section .text
10    global _start
11
12  _start:
13    mov rax, 5    ; open file
14    mov rbx, fileName
15    mov rcx, openrw
16    mov rdx, userWR
17    int 0x80
18
19    mov [fileHandle], eax
20  leitura:
21    xor rbx, rbx
22    mov ebx, [fileHandle]
23    mov rax, 3    ; leitura do arquivo
24    mov rcx, texto
25    mov rdx, maxChars
26    int 0x80
27    ...
```

- Fluxo de Execução do exemplo `a06e02.asm`
  1. _start
  2. leitura
  3. escrita
  4. fecha
  5. fim
- E se for necessário realizar diversas leituras?

Fluxo de Execução

- É a ordem pela qual as instruções são executadas
- TOP → DOWN
  - Executa as instruções na ordem que estas aparecem, de forma sequêncial, do início (`_start`) até a finalização do código
  - `PC` ← `PC+1`
- Ordem determinada por desvios
  - Executa as instruções na ordem que estas aparecem, porém existem Instruções de Desvios que alteram a execução para alguma outra Instrução não sequêncial.
  - `PC` ← `DESTINO`

Fluxo de Execução TOP-DOWN com Desvios



Definições importantes

- *Labels*
  - São rótulos/apelidos para posições de memória
  - Indicam variáveis e **trechos de códigos**

```
_start:  ...
leitura: ...
escrita: ...
fecha:   ...
fim:     ...
```

- Instruções de Controle de Fluxo de Execução
  - Desvios Incondicionais
  - Desvios Condicionais
    - Registrador `Flags`

Desvio Incondicional

- São desvios que não dependem de estado

  `JMP l/r64/m64 ; destino deve conter endereco`

  - Altera o PC para destino
  - Não depende do estado do processador, resultado de operação ou qualquer outro fator

- Laço infinito em C

```c
int main(){

  while (1) {
    // play dead!
  }

  return 0;
}
```

- Laço infinito em Assembly

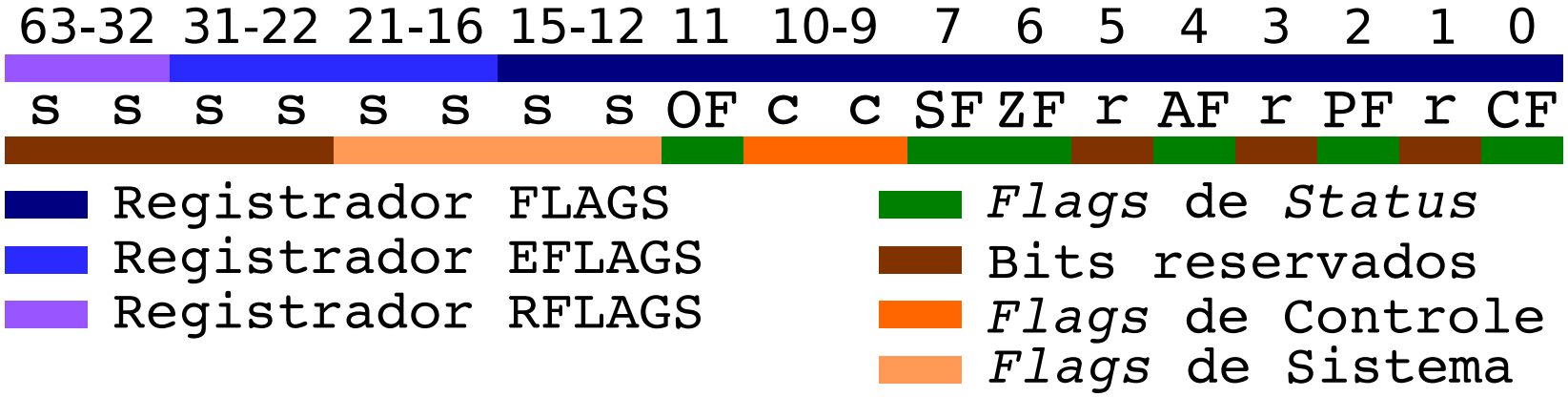
```asm
section .text
    global _start

_start:
    ; play dead!
    jmp _start

fim:
    mov rax, 1
    mov rbx, 0
    int 0x80
```

Desvio Condicional

- São desvios que dependem do estado do processador
  - Registrador EFLAGS
    - *Flags* de *Status*



- 00: CF → Carry
- 02: PF → Parity
- 04: AF → Adjust/Auxiliar
- 06: ZF → Zero
- 07: SF → Sign
- 11: ZF → Overflow

Flags de Status

- CF: *Carry-out* ou *Borrow-in* em operações aritméticas
- PF: Determina se no. de "1"s na palavra é par
- AF: *Carry-out* ou *Borrow-in* em operações BCD8421
  - Considera *carry* ou *borrow* no nibble mais baixo
- ZF: Indica resultado nulo
- SF: Indica resultado negativo na representação C2
- OF: Indica se operação resultou em *Overflow*

Observações

- *Flags* são ativadas em "1"
- No gdb, ao usar `info registers eflags`, são apresentadas somente as flags ativas

Desvio Condicional

- A partir do estado do processador, é possível escolher entre executar o desvio para um novo alvo, ou continuar com o fluxo TOP-DOWN
- Exemplo de desvio (a07e02.asm)
  - JNZ → *Jump if Not Zero*

```
1  section .text
2      global _start
3  _start:
4      mov r8, 10
5  repete:
6      dec r8
7      jnz repete
8  fim: ...
```

- Instrução DEC

  `DEC r/m`

  destino = destino - 1
- JNZ

  `JNZ l/r64/m64`

  Efetua o desvio se $ZF = 0$

Desvios Condicionais

- JConditions
  - São instruções que baseiam-se nas *flags* para determinar se alteram, ou não, o fluxo de execução.
  - 30 instruções gerais
    - Normalmente em "pares"

```
JNZ > Jump if Not Zero ; Desvia se ZF = 0
JZ  > Jump if Zero     ; Desvia se ZF = 1
```

  - 02 instruções que utilizam CX e/ou ECX

```
JCondition l/r64/m64 ; destino deve conter endereco
```

Instruções de Desvios Condicionais

mnemônico	descrição	mnemônimo	descrição
JA	Jump if Above	JNA	Jump if Not Above
JAE	Jump if Above or Equal	JNAE	Jump if Not Above or Equal
JB	Jump if Below	JNB	Jump if Not Below
JBE	Jump if Below or Equal	JNBE	Jump if Not Below or Equal
JG	Jump if Greater	JNG	Jump if Not Greater
JGE	Jump if Greater or Equal	JNGE	Jump if Not Greater or Equal
JL	Jump if Less	JNL	Jump if Not Less
JLE	Jump if Less or Equal	JNLE	Jump if Not Less or Equal
JE	Jump if Equal	JNE	Jump if Not Equal
JZ	Jump if Zero	JNZ	Jump if Not Zero
JS	Jump if Sign	JNS	Jump if Not Sign
JC	Jump if Carry	JNC	Jump if Not Carry
JO	Jump if Overflow	JNO	Jump if Not Overflow
JP	Jump if Parity	JNP	Jump if Not Parity
JPO	Jump if Odd	-	-
JPE	Jump if Parity or Equal	-	-
JCXZ	Jump if CX is Zero	JECXZ	Jump if ECX is Zero

Instrução de Comparação

- CMP
  - Subtração implícita
  - Compara dois operandos e ajusta *flags*
    `OF, ZF, SF, AF, PF e CF`

    `cmp r, r/m/i`

    `cmp m, r/i`

    `cmp operando1, operando2`

    Compara *operando*1 `com` *operando*2

Instrução de Comparaçao - Exemplo A07e03.asm

C

```c
if (v1 == v2)
  printf("v1 eh igual a v2");
else
  if (v1 < v2){
    printf("v1 eh menor do que v2");
  else
    if (v1 > v2)
      printf("v1 eh maior do que v2");
```

NASM

```
...
mov al, [v1]
cmp al, [v2]
je lIguais ; v1 = v2
jl lMenor  ; v1 < v2
jg lMaior  ; v1 > v2
...
```

Exercícios de Fixação

- EF0701 - *Parrot Code*:
  Considere o código EF0701.asm em anexo (semelhante ao a06e01.asm) que contém dois *bugs*:
  - Se a quantidade de caracteres lidos for maior do que *maxChars*, ao finalizar o programa, esses caracteres extras são lançados no terminal, pois foram armazenados no *buffer* do teclado.
  - Se a quantidade de caracteres for menor do que *maxChars*, será impresso duas quebras de linhas. A primeira é o "Enter" da finalização da edição, e a segunda é a impressão da variável `strLF`

- O código EF0701.asm deve ser corrigido de modo a eliminar esses dois *bugs*
  - Deve imprimir, **sempre**, apenas uma quebra de linhas
  - Deve esvaziar o *buffer* do teclado antes de encerrar

Exercícios de Fixação

- EF0702 - *Endless Stamina Parrot Code*:
  Altere o código EF0701 (corrigido) de modo a:
  - Pedir para digitar *algo*
    Se *algo* for diferente de "quit", deve repetir o que foi digitado e solicitar nova entrada.
    Se *algo* for igual à "quit", deve encerrar a execução.

    Observações:
  - Não se esqueça de evitar os *bugs* do exercício EF0701
  - string são vetores de caracteres

Relatório

- Somente relatório
  - O modelo de relatório para a disciplina de LM está disponível em anexo da Aula 01
    - Arquivo **modeloRelatorioLM.odt**
    - A data do relatório é a data de entrega (ver moodle)
  - Somente serão aceitos os relatórios em formato .pdf com nome do arquivo seguindo o padrão: **TY.PXX.nome.sobrenome.pdf**
    - TY é o número da turma prática (1, 2, 3 ou 4)
    - PXX é o número da prática, neste caso: P07

~~Exercícios de Fixação~~ Desafio - pt01 ...

- EF0703 - *Persistent Endless Stamina Parrot Code*: Adicione ao código EF0702 o requisito de persistência em arquivos.
  - Deve ser gravado em arquivo todas as entradas digitadas
    - O programa deve continuar emitindo o texto no terminal
    - O arquivo deve ser criado caso este não exista
  - Para cada execução, deve ser criado uma marcação "=====”
    - O arquivo deve ser aberto com O_APPEND
      Ver aula 06
  - O texto de encerramento de execução deve ser “q!”
  - Entradas vazias (apenas “Enter”) não devem ser gravadas
  - No caso de duas entradas iguais seguidas, a segunda dever substituída por “*”

~~Exercícios de Fixação~~ Desafio - pt02 ...

- EF0703 - *Persistent Endless Stamina Parrot Code*: Exemplo de arquivo de persistência:

```
====
Persistent
Endless
Stamina
====
Parrot
Code
*
!
====
com
*
4
seções
====
com
3
entradasCada
```

~~Exercícios de Fixação~~ Desafio - pt03 q!

- Para os alunos que entregaram todos os relatórios:
  - Este desafio não conta pontos para nota final e não é necessário entregá-lo
- Para os alunos que **não** entregaram todos os relatórios:
  - Este desafio pode substituir os relatórios pendentes
    - Porém, será avaliado em modo de defesa individual em horário préviamente marcado entre aluno e o professor.