Linguagens de Montagem

Controle de Fluxo de Execução

Aula 07

Edmar André Bellorini

Execução TOP → DOWN

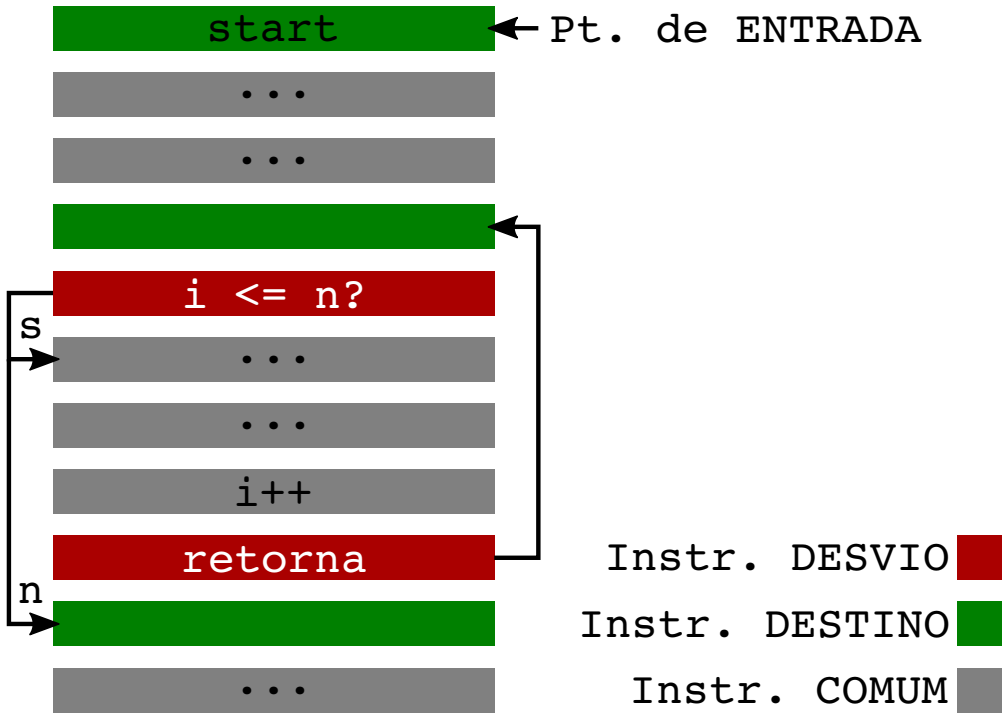
```
18      ...
19  section .data
20      fileName: db "a06e02.txt", 0
21
22  section .bss
23      texto: resb 25
24      fileHandle: resd 1
25
26  section .text
27      global _start
28
29  _start:
30      ;int open(const char *pt, int f, mode_t m);
31      mov rax, 2          ; open file
32      lea rdi, [fileName] ; *pathname
33      mov esi, openrw     ; flags
34      mov edx, userWR     ; mode
35      syscall
36
37      mov [fileHandle], eax
38  leitura:
39      ;ssize_t read(int fd, void *buf, size_t c);
40      mov rax, 0              ; leitura do arquivo
41      mov edi, [fileHandle] ; fd
42      lea rsi, [texto]        ; *buf
43      mov edx, maxChars       ; count
44      syscall
45      ...
```

- 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 $\rightarrow$ DOWN
  - Executa as instruções na ordem que estas aparecem, de forma sequêncial, do início (`_start`) até a finalização do código
  - `PC` $\leftarrow$ `PC + 1`
    Obs.: "+ 1" significa sizeof(instruçãoAtual)
- 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` $\leftarrow$ `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, 60 ; exit
   mov rdi, 0
   syscall
```

Exemplos a07e01a|b.asm

```
7  section .bss
8      destino: resq 1
9
10 section .text
11     global _start
12
13 _start:
14     ; play dead
15     jmp _start ; destino
16    ...
```

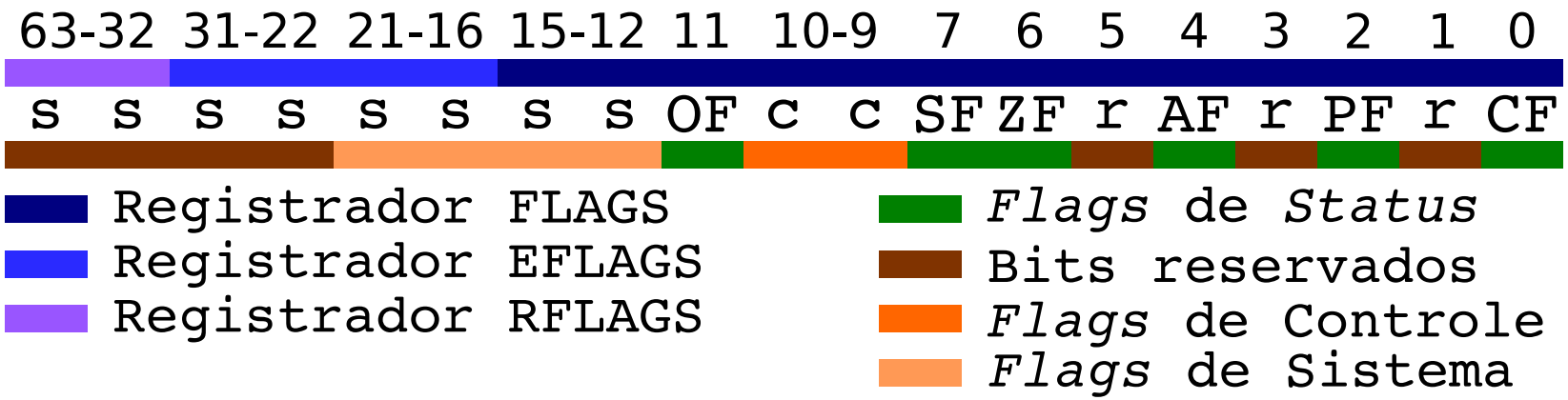
a07e01a.asm

```
7  section .data
8      msg  : db "Onde esta o JUMP?", 10, 
9      msgL : equ $-msg %$
10
11 section .bss
12     destino: resq 1
13
14 section .text
15     global _start
16
17 _start:
18     jmp fim       ; nao executa WRITE
19
20     mov rax, 1   ; WRITE
21     mov rdi, 1
22     lea rsi, [msg]
23     mov edx, msgL
24     syscall
25    ...
```

a07e01b.asm

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: OF → Overflow

Flags de Status

- CF: *Carry-out* ou *Borrow-in* em operações aritméticas
- PF: Determina se no. de "1"s no LSB 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, 5
5  repete:
6      ...
7      dec r8
8      jnz repete
9  fim: ...
```

- Instrução DEC

  `DEC r/m`

  destino = destino - 1
- JNZ

  `JNZ l/r64/m64`

  Efetua o desvio se $ZF = 0$

Exemplo a07e02.asm

```
7   _start:
8       ; var de controle e estado inicial
9       mov r8, 5 ; no. de execucoes
10  repete:
11      ; codigo para imprimir o passo
12          ; captura passo e transforma em char
13          mov [passo], r8b
14          add byte [passo], 0x30
15
16          ; chamada WRITE
17          mov rax, 1
18          mov rdi, 1
19          lea rsi, [passo]
20          mov rdx, 2 ; char + quebra de linha
21          syscall
22
23      ; decrementar contador r8
24      dec r8
25      ; se r8 nao for zero, repete
26      ; caso contrario, continua e finaliza
27      jnz repete
```

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*2 com *operando*1
    Se *operando*2 for *JCondition* do que *operando*1
    Então salta para *destino*
    Senão *continuaFluxo*

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
...
```

Instrução de Comparaçao - Exemplo a07e04.asm

C

```
1  for(ecx = 0;
2      ecx < strLidaL-1;
3      ecx++){
4     ...comandos...
5     }
```

NASM

```
66 entradaFor:
67     mov ecx, 0
68     mov r8d, [strLidaL]
69     dec r8d
70
71     preBloco:
72         cmp ecx, r8d
73         jge saidaFor
74     blocoFor:
75         ...comandos...
76         inc ecx
77         jmp preBloco
78 saidaFor:
```

Multiplicação por Somatório - Exemplo a07e05.asm

C

```c
for(int i = multiplicador;
    i>0;
    i--){
    resultado += multiplicando;
}
```

NASM

```
entradaFOR:
    ; avaliacao de saida do FOR
    cmp ecx, 0
    jle saidaFOR
corpoFOR:
    ; bloco FOR de comandos
    add eax, edx
    ; ajuste da variavel de controle
    dec ecx
    ; retorno para verificar termino de
    jmp entradaFOR

saidaFOR:
```

Atividades - slide 01

- a07at01 - *Parrot Code*:
  Considere o código a07at01.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 a07at01.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

Atividades - slide 02

- a07at02 - *Endless Stamina Parrot Code*:
  Altere o código a07at01 (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* da Atividade a07at01
  - string são vetores de caracteres terminados em 0

~~Atividade~~ Desafio - pt01 ...

- a07at03 - *Persistent Endless Stamina Parrot Code*:
  Adicione ao código a07at02 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 "*"

~~Atividade~~ Desafio - pt02 ...

- a07at03 - *Persistent Endless Stamina Parrot Code*: Exemplo de arquivo de persistência:

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

Fim do Documento

Dúvidas?

Encerramento da Etapa 01

Próximas aulas:

- Seção de tira-dúvidas
- Avaliação