Programa em Assembly, linguagem de maquina, Hexadecimal

Rodrigo Henriky

1. Contador Crescente (0 a 9)

```
# Inicializa $r0 com 9 (valor final)
addi $r0, $r6, 9
# Incrementa $r1 em 1 a cada iteração
addi $r1, $r1, 1
# Se $r1 for igual a $r0, finaliza o loop
beq $r1, $r0, fim
# Retorna ao início do loop
j inicio
# Finaliza o programa
hlt
```

Linguagem de Máquina:

100C6009

10021001

10420001

18000001

0000000C

Hexadecimal

F809 E481 C401 4001 6000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

---

2. Contador Decrescente (9 a 0)

```
# Inicializa $r0 com 9
addi $r0, $r7, 9
# Inicializa $r6 com 1 (valor de decremento)
addi $r6, $r7, 1
# Subtrai $r6 de $r0 a cada iteração
sub $r0, $r0, $r6
# Se $r0 for igual a zero, finaliza o loop
beq $r0, $r7, fim
# Retorna ao início do loop
j inicio
# Finaliza o programa
hlt
```

Linguagem de Máquina:

100E7009

100C7001

10076022

1040E002

18000002

0000000C

Hexadecimal

FC09 FF01 0301 DC01 4002 6000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

---

3. Contador Crescente/Decrescente (0 a 9 e 9 a 0)

```
# Inicializa $r1 com 9 (limite superior)
addi $r1, $r7, 9
# Incrementa $r0 até atingir $r1
addi $r6, $r7, 1
add $r0, $r0, $r6
# Se $r0 atingir 9, inverte a contagem
beq $r1, $r0, inverter
j loop
# Decrementa $r0 até atingir 0
sub $r0, $r0, $r6
# Se $r0 for zero, inverte novamente
beq $r0, $r7, fim
j inverter
hlt
```

Linguagem de Máquina:

100E7009

1006020

10420001

18000002

1006022

1040E002

58000005

18000002

0000000C

Hexadecimal

FC89 FF01 0300 C401 4002 0301 DC01 4005 4002 6000 0000 0000 0000 0000 0000 0000

0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

---

4. Preencher toda a memória com 25

Define o tamanho da memória a ser preenchida (32 bytes)

addi $r1, $r7, 32

Define o valor 25 a ser armazenado

addi $r2, $r7, 25

Armazena o valor 25 na posição atual da memória

sw $r2, 0($r0)

Incrementa a posição da memória

addi $r0, $r0, 1

Se $r0 atingir o limite, finaliza o loop

beq $r0, $r1, fim

j inicio

hlt

Linguagem de Máquina:

220E70020

20F70019

AC420000

20000001

104210001

18000002

0000000C

Hexadecimal

FCA0 FD19 A100 E001 C401 4002 6000 0000 0000 0000 0000 0000 0000 0000 0000 0000

0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

---

5. Inversor de dados na memória

Assembly:

Carrega o valor 0xFFFF em $r2

ori $r2, $0, 0xFFFF

Inverte os bits de $r1 e armazena em $r1

nor $r1, $r1, $r2

Armazena o valor invertido na memória

sw $r1, 0($r0)

Hexadecimal

9C00 FF01 FE8C DE83 0000 1751 4003 9C81 0412 9C00 FE84 DE83 0000 1751 400B 9D02

0822 9C00 FE84 FE02 DE87 DC03 0201 ED81 4015 0F80 1751 4014 9E03 1042 0500 0200

6.Avaliador/Contador de um determinado valor na memória de dados (exemplo para o valor 15)

Assembly:

```
# Define limite de leitura (100 posições)
addi $r1, $r7, 100
# Define valor a ser contado (15)
addi $r2, $r7, 15
# Inicializa contador $r3
add $r3, $r7, $r7
# Lê o valor da memória
lw $r4, 0($r0)
# Se o valor for igual a 15, incrementa contador
beq $r4, $r2, incrementa
# Pula para a próxima posição
j proximo
# Incrementa contador
addi $r3, $r3, 1
# Avança para a próxima posição da memória
addi $r0, $r0, 4
# Continua loop se $r0 < $r1
blt $r0, $r1, loop
```

Linguagem de Máquina:

```
20E70064
20F7000F
01E7E020
8C840000
14820002
```

80000004

20630001

20000004

1501FFFC

Hexadecimal:

20E7 0064

20F7 000F

01E7 E020

8C84 0000

1482 0002

8000 0004

2063 0001

2000 0004

1501 FFFC

7. Multiplicar dois valores (9x5)

Assembly:

```
# Carrega os valores 9 e 5 nos registradores
addi $r1, $r7, 9  # $r1 = 9
addi $r2, $r7, 5  # $r2 = 5
addi $r3, $r7, 1  # Inicializa $r3 com 1

# Se $r2 for zero, termina
beq $r2, $r7, 3  # Se $r2 == 0, termina
```

```
# Loop de multiplicação por soma
add $r0, $r0, $r2  # Soma $r2 ao acumulador $r0
sub $r2, $r2, $r6  # Decrementa $r2
jump 3  # Repete até $r2 ser zero

sw $r0, 0($r7)  # Armazena o resultado na memória
hlt  # Finaliza o programa
```

Hexadecimal

FC89 FD05 FF01 DD03 0080 0B21 4003 BC00 6000 0000 0000 0000 0000 0000 0000 0000

0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

8. Dividir dois valores (45/9)

Assembly:

```
# Inicializa valores
addi $r1, $r7, 45  # $r1 = 45
addi $r2, $r7, 9   # $r2 = 9

# Se $r1 for zero, termina
beq $r1, $r7, 3   # Se $r1 == 0, termina

# Loop de subtração para divisão
sub $r1, $r1, $r2  # Subtrai $r2 de $r1
addi $r6, $r6, 9  # Conta quantas vezes subtraiu
```

jump 2 # Continua enquanto $r1 >= 0

sw $r6, 0($r7) # Armazena o resultado na memória

hlt # Finaliza o programa

Hexadecimal

FCAD FD09 DC83 0511 FB01 4002 BF00 6000 0000 0000 0000 0000 0000 0000 0000 0000

0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

9. Contar valores 5 na memória

Assembly:

Define limite da memória e o valor a ser contado

addi $r1, $r7, 32 # Define limite da memória

addi $r2, $r7, 5   # Define o valor a ser contado (5)

Lê valor da memória

lw $r3, 0($r0) # Carrega um valor da memória

addi $r0, $r0, 1 # Avança para a próxima posição de memória

Verifica se terminou a leitura

beq $r0, $r1, 4 # Se $r0 == limite, termina

```
# Verifica se o valor lido é igual a 5
beq $r2, $r3, 1  # Se $r3 == 5, incrementa contador
jump 2  # Continua verificando memória

addi $r6, $r6, 1  # Incrementa contador
jump 2  # Continua verificando memória
hlt  # Finaliza o programa
```

Hexadecimal

FCA0 FD05 8180 E001 C404 CD01 4002 FB01 4002 6000 0000 0000 0000 0000 0000 0000

0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

10. Cálculo do fatorial

Assembly:

```
# Inicializa variáveis
addi $r1, $r7, 1  # $r1 = 1
addi $r2, $r7, 1  # $r2 = 1
addi $r3, $r7, 1  # $r3 = 1
addi $r4, $r7, 0  # $r4 = 0

# Armazena valor inicial na memória
sw $r2, $r4, 0  # Armazena 1 na posição 0 da memória

# Loop do cálculo do fatorial
```

```
add $r0, $r2, $r7  # Copia $r2 para $r0
addi $r4, $r4, 1  # Incrementa contador
add $r3, $r4, $r7  # Copia contador para $r3

# Se $r3 for zero, termina
beq $r3, $r7, 3  # Se $r3 == 0, termina

# Multiplica
add $r2, $r2, $r0  # Multiplica acumulando
sub $r3, $r3, $r1  # Decrementa contador
jump 8  # Continua o loop
jump 4  # Termina o programa
```

Hexadecimal

FC81 FD01 FD81 FE00 B100 0B80 F201 13B0 DD83 0820 0CB1 4008 4004 6000 0000 0000

0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

Meu código em C

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
```

```
#define MAX_LINE 256

// Função para remover espaços extras
char *trimWhitespace(char *str) {
    while (isspace((unsigned char)*str)) str++;  // Remove espaços à esquerda
    if (*str == 0) return str;  // Se for vazio, retorna
    char *end = str + strlen(str) - 1;
    while (end > str && isspace((unsigned char)*end)) end--;  // Remove espaços à direita
    *(end + 1) = '\0';  // Finaliza corretamente
    return str;
}

// Converte um número decimal para hexadecimal
void decimalToHex(int num, int digits, char *result) {
    snprintf(result, digits + 1, "%0*X", digits, num);
}

// Converte instrução Assembly para código hexadecimal
void convertToMachineCode(const char *instruction, char *hex) {
    char inst[MAX_LINE];
    int rs, rt, rd, immediate, address;

    strcpy(inst, instruction);
    char *token = strtok(inst, " ,()");
```

```
    if (!token) { // Caso a linha seja vazia
        strcpy(hex, "ERRO");
        return;
    }

    token = trimWhitespace(token); // Remover espaços antes da comparação

    if (strcmp(token, "add") == 0) {
        if (sscanf(instruction, "add $r%d, $r%d, $r%d", &rd, &rs, &rt) != 3) {
            strcpy(hex, "ERRO");
            return;
        }
        decimalToHex((0x0 << 13) | (rs << 10) | (rt << 7) | (rd << 4) | 0x0, 4, hex);
    } else if (strcmp(token, "sub") == 0) {
        if (sscanf(instruction, "sub $r%d, $r%d, $r%d", &rd, &rs, &rt) != 3) {
            strcpy(hex, "ERRO");
            return;
        }
        decimalToHex((0x0 << 13) | (rs << 10) | (rt << 7) | (rd << 4) | 0x1, 4, hex);
    } else if (strcmp(token, "addi") == 0) {
        if (sscanf(instruction, "addi $r%d, $r%d, %d", &rt, &rs, &immediate) != 3) {
            strcpy(hex, "ERRO");
            return;
```

```
        }
        decimalToHex((0x3 << 13) | (rs << 10) | (rt << 7) | (immediate &
0x7F), 4, hex);
    } else if (strcmp(token, "lw") == 0) {
        if (sscanf(instruction, "lw $r%d, %d($r%d)", &rt, &immediate,
&rs) != 3) {
            strcpy(hex, "ERRO");
            return;
        }
        decimalToHex((0x4 << 13) | (rs << 10) | (rt << 7) | (immediate &
0x7F), 4, hex);
    } else if (strcmp(token, "sw") == 0) {
        if (sscanf(instruction, "sw $r%d, %d($r%d)", &rt, &immediate,
&rs) != 3) {
            strcpy(hex, "ERRO");
            return;
        }
        decimalToHex((0x5 << 13) | (rs << 10) | (rt << 7) | (immediate &
0x7F), 4, hex);
    } else if (strcmp(token, "beq") == 0) {
        if (sscanf(instruction, "beq $r%d, $r%d, %d", &rs, &rt,
&immediate) != 3) {
            strcpy(hex, "ERRO");
            return;
        }
        decimalToHex((0x6 << 13) | (rs << 10) | (rt << 7) | (immediate &
0x7F), 4, hex);
    } else if (strcmp(token, "jump") == 0) {
        if (sscanf(instruction, "jump %d", &address) != 1) {
            strcpy(hex, "ERRO");
```

```
            return;
        }
        decimalToHex((0x2 << 13) | (address & 0x1FFF), 4, hex);
    } else if (strcmp(token, "hlt") == 0) {
        strcpy(hex, "E000"); // Código fixo para HALT
    } else {
        strcpy(hex, "ERRO");
    }
}

// Processa entrada do arquivo e converte para hexadecimal
void processInput(FILE *inputFile, FILE *outputFile) {
    char line[MAX_LINE], hex[5];
    int lineNum = 0;

    while (fgets(line, sizeof(line), inputFile)) {
        lineNum++;
        line[strcspn(line, "\n")] = '\0'; // Remove \n no final
        convertToMachineCode(line, hex);

        if (strcmp(hex, "ERRO") == 0) {
            fprintf(stderr, "Erro na linha %d: %s\n", lineNum, line);
        } else {
            fprintf(outputFile, "%02X: %s\n", lineNum - 1, hex);
        }
    }
}
```

```
int main() {
    FILE *inputFile = fopen("input.asm", "r");
    FILE *outputFile = fopen("output.hex", "w");

    if (!inputFile) {
        fprintf(stderr, "Erro ao abrir o arquivo de entrada.\n");
        return 1;
    }
    if (!outputFile) {
        fprintf(stderr, "Erro ao abrir o arquivo de saída.\n");
        fclose(inputFile);
        return 1;
    }

    processInput(inputFile, outputFile);

    fclose(inputFile);
    fclose(outputFile);
    return 0;
}
```