

Trabalho 2 – Simulador MIPS – Relatório

Ana Luisa Salvador Alvarez
Pedro Lucas Andrade
Riheldo Mello

160048036
160038316
110138899

Código Implementado

Bibliotecas incluídas

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <time.h>
```

(clear para windows e linux)

```
#ifdef _WIN32
    #define CLEAR "cls"
#else
    #define CLEAR "clear"
#endif
```

Enums dos opcodes

```
// ENUMS
enum OPCODES{
    EXT=0x00,    LW=0x23,    LB=0x20,    LBU=0x24,
    LH=0x21,    LHU=0x25,    LUI=0x0F,    SW=0x2B,
    SB=0x28,    SH=0x29,    BEQ=0x04,    BNE=0x05,
    BLEZ=0x06,    BGTZ=0x07,    ADDI=0x08,    SLTI=0x0A,
    SLTIU=0x0B,    ANDI=0x0C,    ORI=0x0D,    XORI=0x0E,
    J=0x02,    JAL=0x03,    ADDIU=0x09
} opcode;
```

Enums das funcoes

```
enum FUNCT{
    ADD=0x20,    SUB=0x22,    MULT=0x18,    DIV=0x1A,
    AND=0x24,    OR=0x25,    XOR=0x26,    NOR=0x27,
    SLT=0x2A,    JR=0x08,    SLL=0x00,    SRL=0x02,
    SRA=0x03,    SYSCALL=0x0c,    MFHI=0x10,    MFL0=0x12
} funct;
```

Memoria e registradores

```
// Definicao de memoria e registradores
#define MEM_SIZE 4096
int32_t mem[MEM_SIZE];
int32_t reg[32], lo, hi;
uint32_t pc = 0, ri;

int op, rs, rt, rd, sa, fct, k16, k26;

typedef enum {
    false, true
} bool;
```

```
bool sync = false;
```

Descrição das funções

```
// ESCOP0
```

```
//
```

```
// Le um inteiro alinhado. endereço mult 4
```

```
int32_t lw(uint32_t address, int16_t kte);
```

```
// Le meia palavra - 16 bits, retorna inteiro com sinal. endereço mult 2
```

```
int32_t lh(uint32_t address, int16_t kte);
```

```
// Le meia palavra - 16 bits. endereço mult 4
```

```
int32_t lhu(uint32_t address, int16_t kte);
```

```
// Le um byte, retorna int com sinal
```

```
int32_t lb(uint32_t address, int16_t kte);
```

```
// Le byte
```

```
int32_t lbu(uint32_t address, int16_t kte);
```

```
// Escreve uma palavra na memória. endereço mult 4
```

```
void sw(uint32_t address, int16_t kte, int32_t dado);
```

```
// Escreve meia palavra - 16 bits. endereço mult 2
```

```
void sh(uint32_t address, int16_t kte, int16_t dado);
```

```
// Escreve um byte na memória
```

```
void sb(uint32_t address, int16_t kte, int8_t dado);
```

```
// Executa syscall
```

```
void fsyscall();
```

```
// Função para inicializar os vetores de memória e registradores
```

```
void init();
```

```
// Aponta a instrução atual e guarda qual será a próxima
```

```
void fetch();
```

```
// Associa o binário com a função correspondente
```

```
void decode();
```

```
// Execução de cada instrução
```

```
void execute();
```

```
// fetch, decode, execute
```

```
void step();
```

```
// Executa todas as instruções fornecidas pelo binário - fetch, decode, execute  
até o término
```

```
void run();
```

```
// Gera um arquivo binário com o resultado da memória de dados
```

```
void dump_mem(int start, int end, char format);
```

```
// Gera um arquivo binário com o resultado dos registradores
```

```
void dump_reg(char format);
```

```
//
```

```
//
```

```
// FUNÇÕES
```

```
//
```

Retorna uma word armazenada no endereço/posição passados como parâmetro.

```
int32_t lw(uint32_t address, int16_t kte){
```

```
    int32_t temp;
```

```
    temp = (int32_t) mem[(address + kte)/4];
```

```
    return temp;
```

```
}
```

Retorna uma half word armazenada no endereço/posição passados como parâmetro.

```
int32_t lh(uint32_t address, int16_t kte){
```

```
    int32_t temp;
```

```
    if( (address+kte)%4 == 0){ (se mod 4 for 0)
```

```

        temp = (int32_t) (mem[(address + kte)/4] & 0x0000ffff);
        if(temp >= 0x00008000) temp = (int32_t) temp | 0xffff0000;
    } else {
        temp = (int32_t) (mem[(address + kte-2)/4] & 0xffff0000);
        if(temp >= 0x80000000) temp = (int32_t) temp | 0x0000ffff;
        temp = temp >> 16;
    }
    return temp;
}

```

Retorna uma half word unsigned armazenada no endereco/posicao passados como parametro.

```

int32_t lhu(uint32_t address, int16_t kte){
    uint32_t temp;
    if((address+kte)%4 == 0) { (se mod 4 for 0)
        temp = (int32_t) (mem[(address + kte)/4] & 0x0000ffff);
    } else {
        temp = (int32_t) (mem[(address + kte-2)/4] & 0xffff0000);
        temp = temp >> 16;
    }
    return temp;
}

```

Retorna um byte armazenado no endereco/posicao passados como parametro.

```

int32_t lb(uint32_t address, int16_t kte){
    int32_t temp;

    if((address+kte)%4 == 0){ (se mod 4 for 0)
        temp = (int32_t) (mem[(address + kte)/4] & 0x000000ff);
        if(temp >= 0x00000080) temp = (int32_t) temp | 0xffffffff00;
    } else if ( (address+kte)%4 == 3){ (se mod 4 for 3)
        temp = (int32_t) (mem[(address + kte-3)/4] & 0xff000000);
        if(temp >= 0x80000000) temp = (int32_t) temp | 0x00ffffff;
        temp = temp >> 24;
    } else if ( (address+kte)%2 == 0){ (se mod 2 for 0)
        temp = (int32_t) (mem[(address + kte-2)/4] & 0x00ff0000);
        if(temp >= 0x00800000) temp = (int32_t) temp | 0xff00ffff;
        temp = temp >> 16;
    } else {
        temp = (int32_t) (mem[(address + kte-1)/4] & 0x0000ff00);
        if(temp >= 0x00008000) temp = (int32_t) temp | 0xffff00ff;
        temp = temp >> 8;
    }

    return temp;
}

```

Retorna um byte unsigned armazenado no endereco/posicao passados como parametro

```

int32_t lbu(uint32_t address, int16_t kte){
    uint32_t temp;

    if((address+kte)%4 == 0) (se mod 4 for 0)
        temp = (int32_t) (mem[(address + kte)/4] & 0x000000ff);
    else if((address+kte)%4 == 3){ (se mod 4 for 3)
        temp = (int32_t) (mem[(address + kte-3)/4] & 0xff000000);
        temp = temp >> 24;
    } else if((address+kte)%2 == 0){ (se mod 2 for 0)
        temp = (int32_t) (mem[(address + kte-2)/4] & 0x00ff0000);
        temp = temp >> 16;
    } else{
        temp = (int32_t) (mem[(address + kte-1)/4] & 0x0000ff00);
        temp = temp >> 8;
    }
}

```

```

    return temp;
}

```

Salva na memoria (vetor), na posicao e endereco indicados, uma word. Todos são passados como parametro.

```

void sw(uint32_t address, int16_t kte, int32_t dado){
    mem[(address + kte)/4] = (int32_t) dado;
}

```

Salva na memoria (vetor), na posicao e endereco indicados, uma half word. Todos são passados como parametro.

```

void sh(uint32_t address, int16_t kte, int16_t dado){
    int32_t t1, t2;

    if((address+kte)%4 == 0){ (se mod 4 for 0)
        t1 = (int32_t) mem[(address + kte)/4] & 0xffff0000;
        t2 = (int32_t) dado & 0x0000ffff;
        mem[(address + kte)/4] = (int32_t) t1 + t2;
    } else {
        t1 = (int32_t) mem[(address + kte-2)/4] & 0x0000ffff;
        t2 = (int32_t) dado & 0x0000ffff;
        t2 = t2 << 16;
        mem[(address + kte-2)/4] = (int32_t) t1 + t2;
    }
}

```

Salva na memoria (vetor), na posicao e endereco indicados, um byte. Todos são passados como parametro.

```

void sb(uint32_t address, int16_t kte, int8_t dado){
    int32_t t1, t2;

    if( (address+kte)%4 == 0){ (se mod 4 for 0)
        t1 = (int32_t) mem[(address + kte)/4] & 0xffffffff00;
        t2 = (int32_t) dado & 0x000000ff;
        mem[(address + kte)/4] = (int32_t) t1 + t2;
    } else if((address+kte)%4 == 3) { (se mod 4 for 3)
        t1 = (int32_t) mem[(address + kte-3)/4] & 0x00ffffff;
        t2 = (int32_t) dado & 0x000000ff;
        t2 = t2 << 24;
        mem[(address + kte-3)/4] = (int32_t) t1 + t2;
    } else if((address+kte)%2 == 0) { (se mod 2 for 0)
        t1 = (int32_t) mem[(address + kte-2)/4] & 0xff00ffff;
        t2 = (int32_t) dado & 0x000000ff;
        t2 = t2 << 16;
        mem[(address + kte-2)/4] = (int32_t) t1 + t2;
    } else {
        t1 = (int32_t) mem[(address + kte-1)/4] & 0xffff00ff;
        t2 = (int32_t) dado & 0x000000ff;
        t2 = t2 << 8;
        mem[(address + kte-1)/4] = (int32_t) t1 + t2;
    }
}

```

Funcao para executar um syscall.

```

void fsyscall(){
    int temp, c, i;

    if (reg[2] == 1) { (se for um inteiro)
        printf("%d\n", reg[4]); (printa o inteiro)
    } else if(reg[2] == 4) { (se for um char)
        temp = reg[4];
        c = lb(temp,0); (c recebe o primeiro char)
        for(i = 1; c != '\0'; i++) { (até chegar no fim da string)
            printf("%c",c); (printa o char)
        }
    }
}

```

```

        c = lb(temp,i); (carrega o char do i, começando no 1 e seguindo i++)
        if(i == 3){
            i = -1;
            temp += 4;
        }
    }
    printf("\n");
} else if(reg[2] == 10){ (se for booleano)
    sync = true; (recebe verdade)
}
}

```

Inicializa memoria e registradores em 0

```

void init(){
    int i;
    for(i = 0; i < 32; i++) {
        reg[i] = 0; (todos os registradores vao receber 0)
    }
    for(i = 0; i < MEM_SIZE; i++) {
        mem[i] = 0; (memoria toda recebe 0)
    }
    hi = 0; (high recebe 0)
    lo = 0; (low recebe 0)
}

```

Aponta a instrucao atual e guarda a proxima

```

void fetch(){
    ri = mem[pc/4];
    pc += 4;
}

```

Decodifica a instrucao correspondente ao binario

```

void decode(){
    op  = ri >> 26 & 0x3f;
    rs  = ri >> 21 & 0x1f;
    rt  = ri >> 16 & 0x1f;
    rd  = ri >> 11 & 0x1f;
    sa  = ri >> 6 & 0x1f;
    fct = ri & 0x3f;
    k16 = ri & 0xffff;
    k26 = ri & 0x3fffffff;
}

```

Execucao da instrucao

```

void execute(){
    int ender;

    opcode = op; (verifica qual o opcode)
    if(opcode == BEQ || opcode == BNE || opcode == BLEZ ||
        opcode == BGTZ || opcode == ADDI || opcode == ADDIU ||
        opcode == SLTI || opcode == SLTIU ||
        opcode == ANDI || opcode == ORI || opcode == XORI) {

        if(k16 >= 0x8000) {
            k16 = k16 | 0xffff0000;
        } else {
            k16 = k16 & 0x0000ffff;
        }
    }

    switch(opcode) { (chama a funcao da instrucao desejada)
        case LW:

```

```

        reg[rt] = lw(reg[rs], k16);
        break;

case LH:
    reg[rt] = lh(reg[rs], k16);
    break;

case LHU:
    reg[rt] = lhu(reg[rs], k16);
    break;

case LB:
    reg[rt] = lb(reg[rs], k16);
    break;

case LBU:
    reg[rt] = lbu(reg[rs], k16);
    break;

case LUI:
    reg[rt] = k16 << 16;
    break;

case SW:
    sw(reg[rs], k16, reg[rt]);
    ender = (reg[rs] + k16)/4;
    break;

case SH:
    sh(reg[rs], k16, reg[rt]);
    ender = (reg[rs] + k16)/4;
    break;

case SB:
    sb(reg[rs], k16, reg[rt]);
    ender = (reg[rs] + k16)/4;
    break;

case BEQ:
    if(reg[rs] == reg[rt]) pc += (k16 << 2);
    break;

case BNE:
    if(reg[rs] != reg[rt]) pc += (k16 << 2);
    break;

case BLEZ:
    if(reg[rs] <= 0) pc += (k16 << 2);
    break;

case BGTZ:
    if(reg[rs] > 0) pc += (k16 << 2);
    break;

case ADDI:
    reg[rt] = reg[rs] + k16;
    break;

case ADDIU:
    reg[rt] = ((uint32_t) reg[rs] + k16);
    break;

case SLTI:
    reg[rt] = reg[rs] < k16;

```

```

        break;

case SLTIU:
    reg[rt] = ((uint32_t) reg[rs]) < ((uint32_t) k16);
    break;

case ANDI:
    reg[rt] = reg[rs] & k16;
    break;

case ORI:
    reg[rt] = reg[rs] | k16;
    break;

case XORI:
    reg[rt] = reg[rs] ^ k16;
    break;

case J:
    pc = (pc & 0xf0000000) | (k26 << 2);
    break;

case JAL:
    reg[31] = pc;
    pc = (pc & 0xf0000000) | (k26 << 2);
    break;

default:
    break;
}

if(op == EXT){
    switch(fct){
        case ADD:
            reg[rd] = reg[rs] + reg[rt];
            break;

        case SUB:
            reg[rd] = reg[rs] - reg[rt];
            break;

        case MULT:
            lo = reg[rs] * reg[rt];
            break;

        case DIV:
            lo = reg[rs] / reg[rt];
            hi = reg[rs] % reg[rt];
            break;

        case AND:
            reg[rd] = reg[rs] & reg[rt];
            break;

        case OR:
            reg[rd] = reg[rs] | reg[rt];
            break;

        case XOR:
            reg[rd] = reg[rs] ^ reg[rt];
            break;

        case NOR:
            reg[rd] = ~(reg[rs] | reg[rt]);

```

```

        break;

    case SLT:
        reg[rd] = reg[rs] < reg[rt];
        break;

    case JR:
        pc = reg[rs];
        break;

    case SLL:
        reg[rd] = (uint32_t) reg[rt] << sa;
        break;

    case SRL:
        reg[rd] = (uint32_t) reg[rt] >> sa;
        break;

    case SRA:
        reg[rd] = reg[rt] >> sa;
        break;

    case MFHI:
        reg[rd] = hi;
        break;

    case MFLO:
        reg[rd] = lo;
        break;

    case SYSCALL:
        fsyscall();
        break;

    default:
        break;
    }
}
}

```

Chama as funcoes fetch, decode e execute

```

void step(){
    if(syc == false && pc < MEM_SIZE * 2) {
        fetch();
        decode();
        execute();
    }
}

```

Executa todas as instrucoes fornecidas pelo binario - fetch, decode, execute ate o termino

```

void run(){
    while(syc == false && pc < MEM_SIZE * 2) {
        fetch();
        decode();
        execute();
    }
}

```

Gera um arquivo binario com o resultado da memoria de dados.

```

void dump_mem(int start, int end, char format){
    int i, cont;
    FILE *dataS;

```



```

dataS = fopen("mem.txt", "w");

fprintf(dataS, "MEMORY:\n");

if((format=='h')||(format=='H')){
    for (i=start;i<=end;i++){
        fprintf(dataS, "0x%08x = 0x%08x\n", i*4, mem[i]);
    }
}
else{
    for (i=start;i<=end;i++){
        fprintf(dataS, "0x%08x = %d\n", i*4, mem[i]);
    }
}

fclose(dataS);
}

```

Gera um arquivo binario com o conteudo dos registradores

```

void dump_reg(char format){
    int i;
    FILE *regS;

    regS = fopen("reg.txt", "w");

    fprintf(regS, "REGISTERS:\n");
    if(format == 'h' || format == 'H') {
        for(i = 0; i < 32; i++){
            fprintf(regS, "$%d = 0x%08x\n", i, reg[i]);
        }
        fprintf(regS, "pc = 0x%08x\nhi = 0x%08x\nlo = 0x%08x\n", pc, hi, lo);
    } else{
        for(i = 0; i < 32; i++){
            fprintf(regS, "$%d = %d\n", i, reg[i]);
        }
        fprintf(regS, "pc = %d\nhi = %d\nlo = %d\n", pc, hi, lo);
    }
    fclose(regS);
}

```

```

void delay(int mseconds){
    clock_t limit = mseconds + clock();
    while (limit>clock());
}

```

Cardapio de funcoes, com guia de ajuda de como utilizar cada funcao.

```

void menu(){
    int choice, quit=0, dump_mem_inicio=-1, dump_mem_end=-1, erro_flag, i;
    char formato='s', anim;

    while(quit==0){ (enquanto não for pra sair)
        system(CLEAR); (limpa a tela e printa as opcoes)
        printf("Insira o numero da funcao desejada:\n\n");
        printf("(1) - Step\n      +Essa funcao executa uma instrucao do
MIPS. Nao eh necessario inserir parametros na mesma.\n\n");
        printf("(2) - Run\n      +Essa funcao executa o programa
completo, ou seja, ateh encontrar uma chamada de sistema para encerramento\n
ou ateh atingir o limite do segmento de codigo. Nao eh necessario inserir
parametros na mesma.\n\n");
    }
}

```

```

printf("(3) - Dump_mem\n      +Essa funcao imprime e exporta o
conteudo de uma secao da memoria, em hexadecimal(h) ou decimal(d),\n
delimitada por enderecos de inicio e fim. Eh necessario inserir os parametros:\n
-Inicio(0-4095)\n      -Fim(0-4095)\n      -Formato(d/h)\n\n");
printf("(4) - Dump_reg\n      +Essa funcao imprime e exporta o
conteudo dos registradores, em hexadecimal(h) ou decimal(d). \n      Eh
necessario inserir os parametros:\n      -Formato(d/h)\n\n");
printf("(5) - Exit\n      +Termina a execucao do programa.\n\n");
scanf("%d", &choice); (le a opção escolhida)
getchar();
system(CLEAR);

erro_flag=0;
switch(choice){ (o que cada opção faz)
    case 1:
        printf("Funcao escolhida - Step\n\n");
        step();
        printf("\nExecucao terminada. Pressione qualquer
tecla para continuar\n");
        getchar();
        break;
    case 2:
        printf("Funcao escolhida - Run\n\n");
        run();
        printf("\nExecucao terminada. Pressione qualquer
tecla para continuar\n");
        getchar();
        break;
    case 3:
        printf("Funcao escolhida - Dump_mem\n");
        while((dump_mem_inicio<0)||
(dump_mem_inicio>4095)){
            if(erro_flag==1){ (erro de parametro
digitado fora da faixa)
                printf("0 parametro inserido
esta errado\n");
            }
            printf("Insira os parametros:\n
-Endereco de inicio (0-4095): ");
            scanf("%d", &dump_mem_inicio);
            erro_flag=1;
        }
        erro_flag=0;
        while((dump_mem_end<0)|| (dump_mem_end>4095)){
            if(erro_flag==1){ (erro de parametro
digitado fora da faixa)
                printf("0 parametro inserido
esta errado\n");
            }
            printf("\n      -Endereco de fim (0-
4095): ");
            scanf("%d", &dump_mem_end);
            erro_flag=1;
        }
        erro_flag=0;
        while((formato!='d')&&(formato!='D')&&(formato!
='h')&&(formato!='H')){ (seleciona decimal ou hexadecimal)
            if(erro_flag==1){
                printf("0 parametro inserido
esta errado\n"); (erro caso não tenha sido digitado d, D, h, ou H)
            }
            getchar();
            printf("\n      -Formato (d/h):");
            scanf("%c", &formato);

```

```

        erro_flag=1;
    }
    system(CLEAR);
    printf("Funcao escolhida - Dump_mem\n");
    dump_mem(dump_mem_inicio, dump_mem_end,
formato);

    dump_mem_inicio=-1;
    dump_mem_end=-1;
    formato='s';
    printf("          -Arquivo mem.txt criado.\n");
    delay(2000); (delay para facilitar a
visualizacao do usuario)

    break;
case 4:
    printf("Funcao escolhida - Dump_reg\n");
    printf("Insira os parametros:");
    erro_flag=0;
    while((formato!='d')&&(formato!='D')&&(formato!
='h')&&(formato!='H')){
        if(erro_flag==1){
            printf("0 parametro inserido
esta errado\n");

        }
        printf("\n          -Formato (d/h):");
        scanf("%c", &formato);
        erro_flag=1;
    }
    system(CLEAR);
    printf("Funcao escolhida - Dump_reg\n");
    dump_reg(formato);
    formato='s';
    printf("          -Arquivo reg.txt criado.\n");
    delay(2000); (delay para facilitar a
visualizacao do usuario)

    break;
case 5:
    quit=1; (flag para encerrar)
    printf("Funcao escolhida - Exit\n
-Encerrando Simulador\n");

    for (i=0;i<16;i++){ (animacao ilustrativa)
        switch(i%4){
            case 0:
                anim='|';
                break;
            case 1:
                anim='/' ;
                break;
            case 2:
                anim='- ' ;
                break;
            case 3:
                anim='\\' ;
                break;
        }

        delay(250); (delay para facilitar a
visualizacao do usuario)

        printf("\r          %c", anim);
    }
    system(CLEAR);
    break;
default:
    printf("Opcao invalida\n");

```

```

                                delay(2000); (delay para facilitar a
visualizacao do usuario)
                                break;
                                }
                                }
}

int main(int argc, const char * argv[]) {

    FILE *prog_text, *prog_data;

    init(); (inicializa memoria e registradores com 0)
    pc = 0;

    if (argc==4){
        prog_text = fopen(argv[1], "rb"); (abre o arquivo de instrucoes)
        prog_data = fopen(argv[2], "rb"); (abre o arquivo de data)
        fread(mem, 4, 2048, prog_text); /* Le os arquivos gerados pelo
MARS */
        fread(mem + 2048, 4, 2048, prog_data);
        fclose(prog_text);
        fclose(prog_data);
        if(argv[3][0]=='f'){ (se parametro 'f', executa todo o
programa assembly (função run()) e geraa reg.txt, e mem.txt)

                                run();
                                dump_reg('h');
                                dump_mem(0, 4095, 'h');
                                }
                                else{
com cardapio de opcoes) if(argv[3][0]=='p'){ (se parametro 'p', abre a interface
                                menu();
                                }
                                else{
                                printf("Terceiro parametro incorreto ('p' ou
'f'\n)\n"); (erro)
                                }
                                }
                                }
                                else{
                                printf("Erro no numero de parametros inseridos\n");
                                delay(3000); (delay para melhor visualizacao do usuario)
                                }
                                }
                                return 0;
}

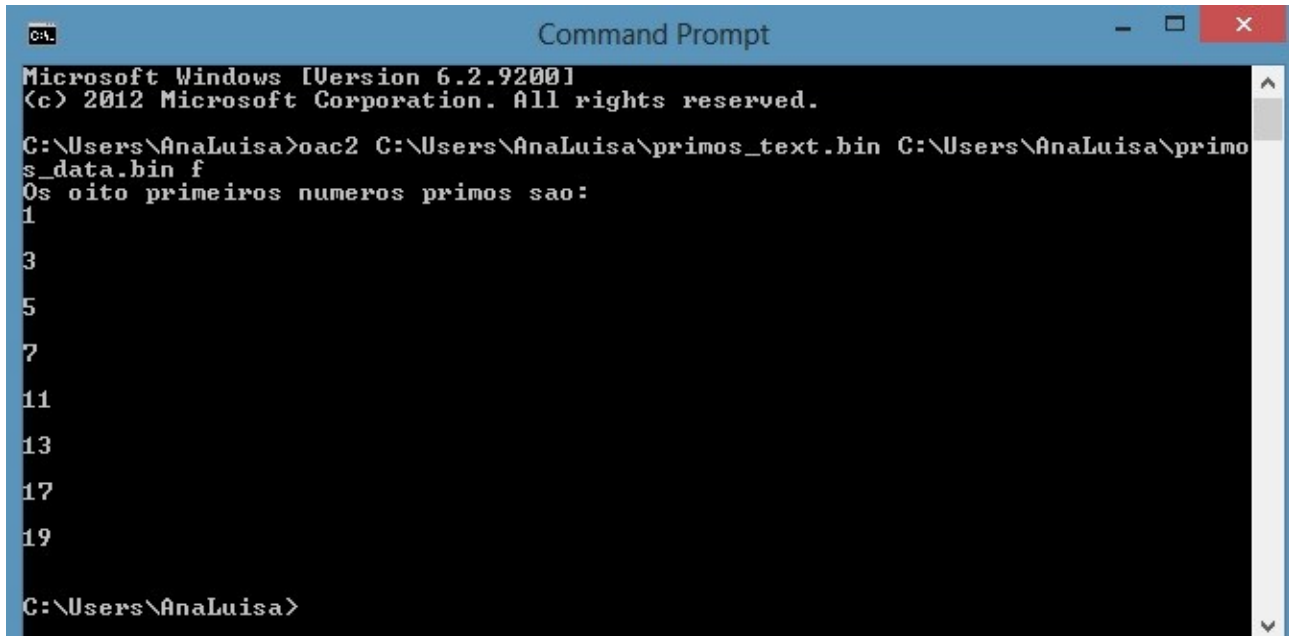
```

Compilado com GCC. Funcionando no Linux e no Windows.

Testes e Resultados

1) Executando o programa com arquivos primos_text.bin e primos_data.bin

Na opção 'f':

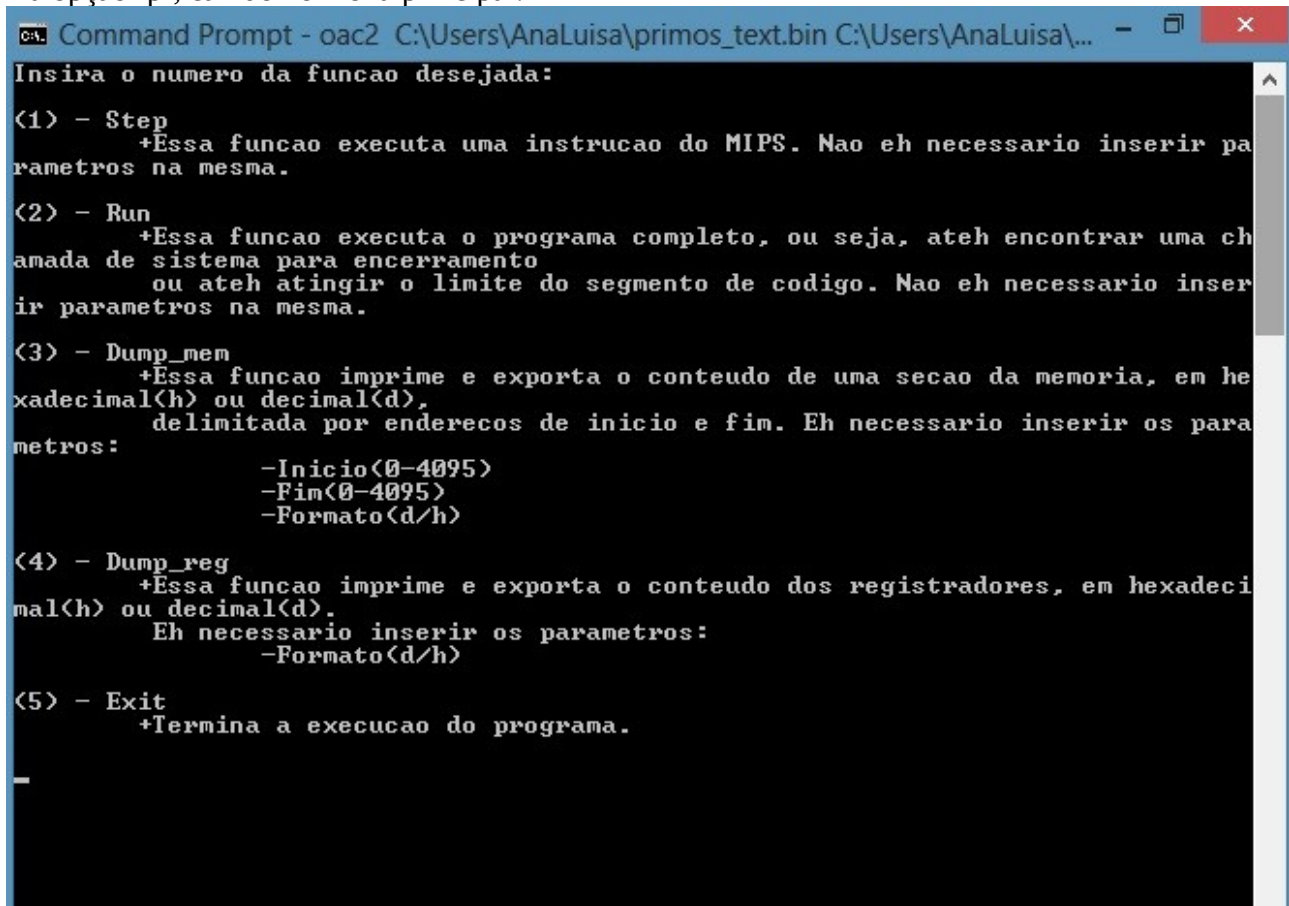


```
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

C:\Users\AnaLuisa>oac2 C:\Users\AnaLuisa\primos_text.bin C:\Users\AnaLuisa\primos_data.bin f
Os oito primeiros numeros primos sao:
1
3
5
7
11
13
17
19

C:\Users\AnaLuisa>
```

Na opção 'p', caindo no menu principal:



```
Command Prompt - oac2 C:\Users\AnaLuisa\primos_text.bin C:\Users\AnaLuisa\...
Insira o numero da funcao desejada:

(1) - Step
      +Essa funcao executa uma instrucao do MIPS. Nao eh necessario inserir pa
rametros na mesma.

(2) - Run
      +Essa funcao executa o programa completo, ou seja, ateh encontrar uma ch
amada de sistema para encerramento
      ou ateh atingir o limite do segmento de codigo. Nao eh necessario inser
ir parametros na mesma.

(3) - Dump_mem
      +Essa funcao imprime e exporta o conteudo de uma secao da memoria, em he
xadecimal(h) ou decimal(d),
      delimitada por enderecos de inicio e fim. Eh necessario inserir os para
metros:
          -Inicio(0-4095)
          -Fim(0-4095)
          -Formato(d/h)

(4) - Dump_reg
      +Essa funcao imprime e exporta o conteudo dos registradores, em hexadeci
mal(h) ou decimal(d).
      Eh necessario inserir os parametros:
          -Formato(d/h)

(5) - Exit
      +Termina a execucao do programa.

-
```

Seguindo a execução 'p', escolhendo a opção 1:

```
Command Prompt - oac2 C:\Users\AnaLuisa\primos_text.bin C:\Users\AnaLuisa\...
Funcao escolhida - Step

Execucao terminada. Pressione qualquer tecla para continuar
```

Seguindo a execução 'p', escolhendo a opção 2:

```
Command Prompt - oac2 C:\Users\AnaLuisa\primos_text.bin C:\Users\AnaLuisa\...
Funcao escolhida - Run

Os oito primeiros numeros primos sao:
1
3
5
7
11
13
17
19

Execucao terminada. Pressione qualquer tecla para continuar
```

Seguindo a execução 'p', escolhendo a opção 3:

```
Command Prompt - oac2 C:\Users\AnaLuisa\primos_text.bin C:\Users\AnaLuisa\...
Funcao escolhida - Dump_mem
Insira os parametros:
-Endereco de inicio <0-4095>: 200
-Endereco de fim <0-4095>: 100
-Formato <d/h>:d_
```

Seguindo a execução 'p', escolhendo a opção 3, após a criação do arquivo:

```
Command Prompt - oac2 C:\Users\AnaLuisa\primos_text.bin C:\Users\AnaLuisa\...
Funcao escolhida - Dump_mem
-Arquivo mem.txt criado.
```

Seguindo a execução 'p', escolhendo a opção 4:

```
C:\ Command Prompt - oac2 C:\Users\AnaLuisa\primos_text.bin C:\Users\AnaLuisa\... - [X]
Funcao escolhida - Dump_reg
Insira os parametros:
  -Formato <d/h>:d
```

Seguindo a execução 'p', escolhendo a opção 4, após criação do arquivo:

```
C:\ Command Prompt - oac2 C:\Users\AnaLuisa\primos_text.bin C:\Users\AnaLuisa\... - [X]
Funcao escolhida - Dump_reg
  -Arquivo reg.txt criado.
```

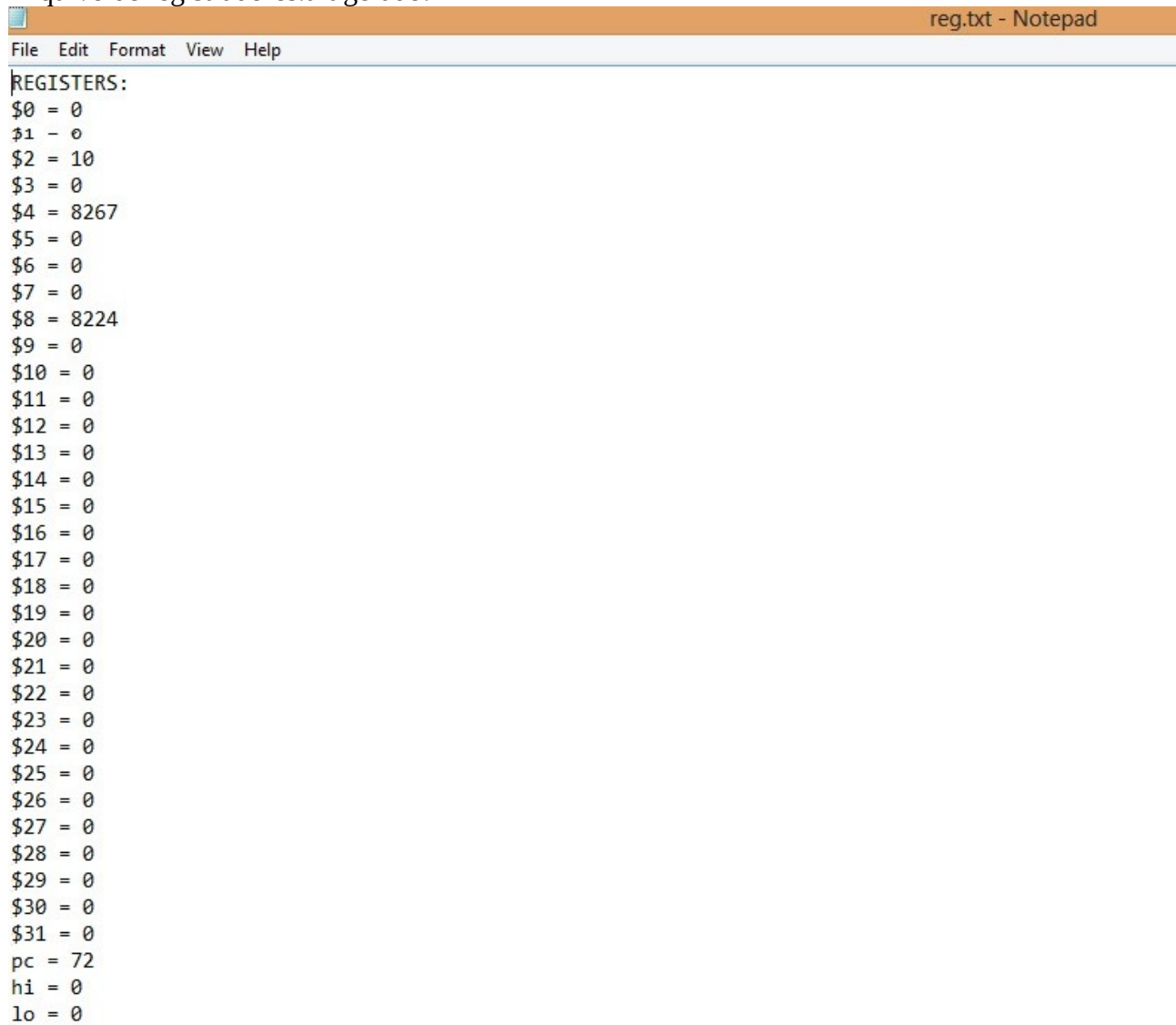
Seguindo a opção 'p', escolhendo a opção 5:

```
C:\ Command Prompt - oac2 C:\Users\AnaLuisa\primos_text.bin C:\Users\AnaLuisa\... - [X]
Funcao escolhida - Exit
  -Encerrando Simulador
/
```

Arquivo de memoria.txt gerado:

```
mem.txt - Notepad
File Edit Format View Help
MEMORY:
0x00000190 = 0
0x00000194 = 0
0x00000198 = 0
0x0000019c = 0
0x000001a0 = 0
0x000001a4 = 0
0x000001a8 = 0
0x000001ac = 0
0x000001b0 = 0
0x000001b4 = 0
0x000001b8 = 0
0x000001bc = 0
0x000001c0 = 0
0x000001c4 = 0
0x000001c8 = 0
0x000001cc = 0
0x000001d0 = 0
0x000001d4 = 0
0x000001d8 = 0
0x000001dc = 0
0x000001e0 = 0
0x000001e4 = 0
0x000001e8 = 0
0x000001ec = 0
0x000001f0 = 0
0x000001f4 = 0
0x000001f8 = 0
0x000001fc = 0
0x00000200 = 0
0x00000204 = 0
0x00000208 = 0
0x0000020c = 0
0x00000210 = 0
0x00000214 = 0
0x00000218 = 0
0x0000021c = 0
0x00000220 = 0
```

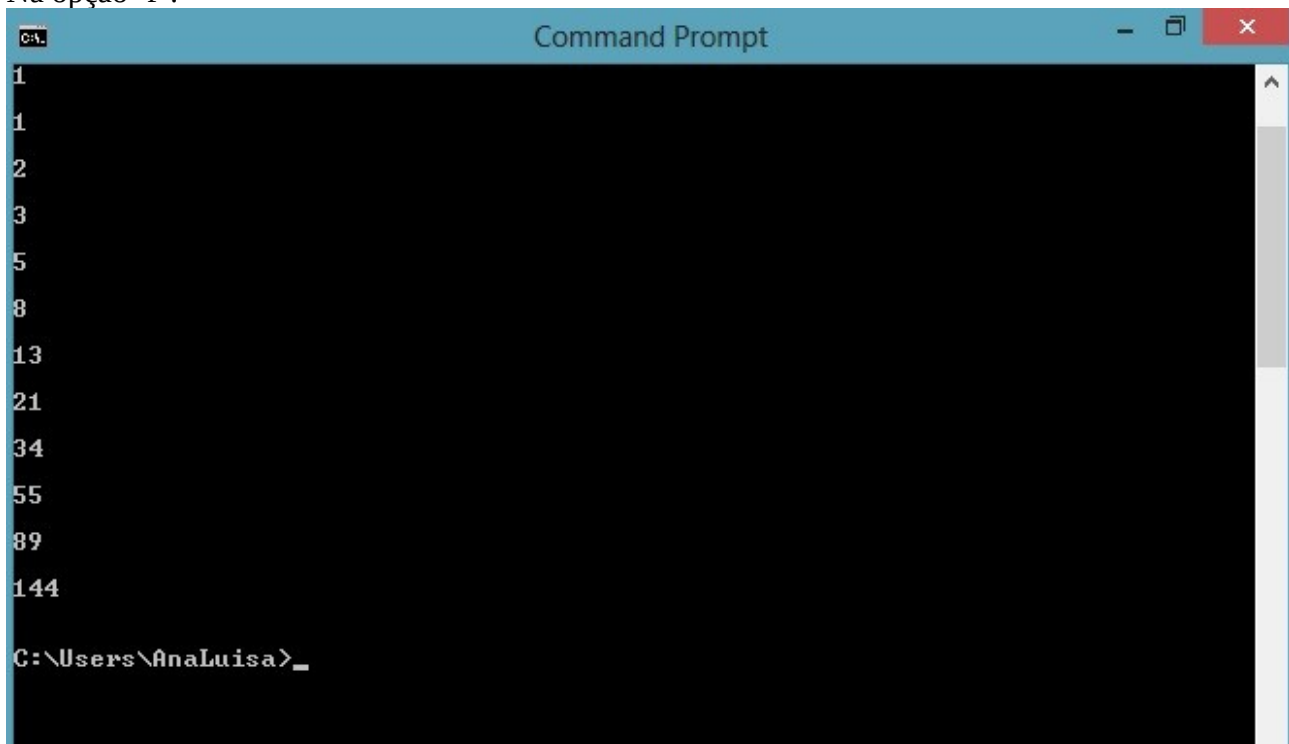
Arquivo de registradores.txt gerado:



```
REGISTERS:  
$0 = 0  
$1 = 0  
$2 = 10  
$3 = 0  
$4 = 8267  
$5 = 0  
$6 = 0  
$7 = 0  
$8 = 8224  
$9 = 0  
$10 = 0  
$11 = 0  
$12 = 0  
$13 = 0  
$14 = 0  
$15 = 0  
$16 = 0  
$17 = 0  
$18 = 0  
$19 = 0  
$20 = 0  
$21 = 0  
$22 = 0  
$23 = 0  
$24 = 0  
$25 = 0  
$26 = 0  
$27 = 0  
$28 = 0  
$29 = 0  
$30 = 0  
$31 = 0  
pc = 72  
hi = 0  
lo = 0
```

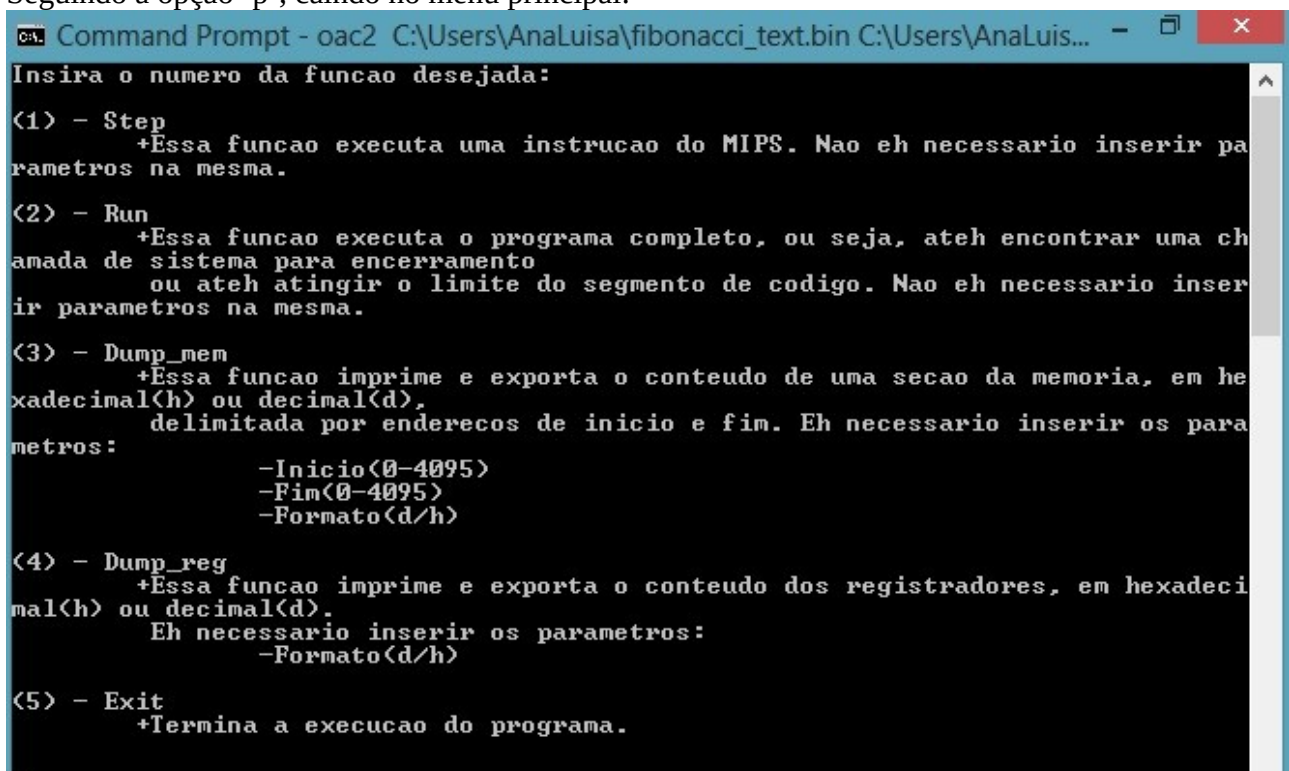

2) Executando o programa com arquivos fibonacci_text.bin e fibonacci_data.bin

Na opção 'f':



```
C:\Users\AnaLuisa>_
1
1
2
3
5
8
13
21
34
55
89
144
C:\Users\AnaLuisa>_
```

Seguindo a opção 'p', caindo no menu principal:



```
Command Prompt - oac2 C:\Users\AnaLuisa\fibonacci_text.bin C:\Users\AnaLuisa\...
Insira o numero da funcao desejada:
(1) - Step
      +Essa funcao executa uma instrucao do MIPS. Nao eh necessario inserir pa
rametros na mesma.
(2) - Run
      +Essa funcao executa o programa completo, ou seja, ateh encontrar uma ch
amada de sistema para encerramento
      ou ateh atingir o limite do segmento de codigo. Nao eh necessario inser
ir parametros na mesma.
(3) - Dump_mem
      +Essa funcao imprime e exporta o conteudo de uma secao da memoria, em he
xadecimall(h) ou decimal(d),
      delimitada por enderecos de inicio e fim. Eh necessario inserir os para
metros:
          -Inicio(0-4095)
          -Fim(0-4095)
          -Formato(d/h)
(4) - Dump_reg
      +Essa funcao imprime e exporta o conteudo dos registradores, em hexadeci
mal(h) ou decimal(d).
      Eh necessario inserir os parametros:
          -Formato(d/h)
(5) - Exit
      +Termina a execucao do programa.
```

Seguindo a opção 'p', escolhendo a opção 1:

```
CA. Command Prompt - oac2 C:\Users\AnaLuisa\fibonacci_text.bin C:\Users\AnaLuisa... - [X]
Funcao escolhida - Step

Execucao terminada. Pressione qualquer tecla para continuar
```

Seguindo a opção 'p', escolhendo a opção 2:

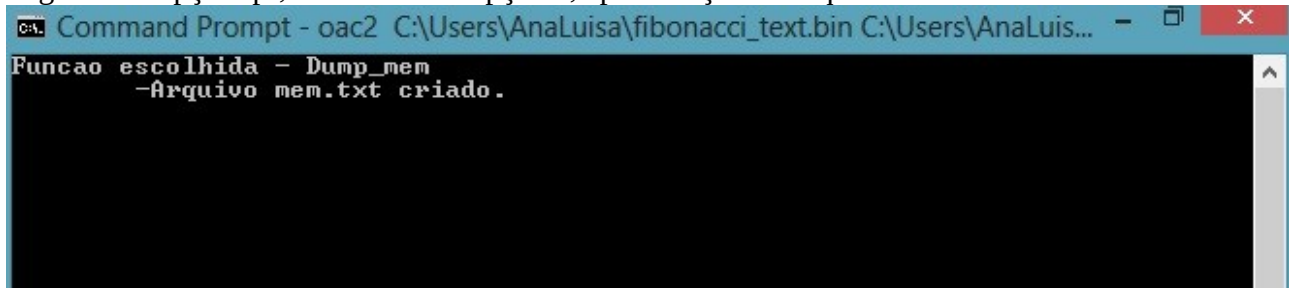
```
CA. Command Prompt - oac2 C:\Users\AnaLuisa\fibonacci_text.bin C:\Users\AnaLuisa... - [X]
Funcao escolhida - Run
The Fibonacci numbers are:
1
1
2
3
5
8
13
21
34
55
89
144

Execucao terminada. Pressione qualquer tecla para continuar
```

Seguindo a opção 'p', escolhendo a opção 3:

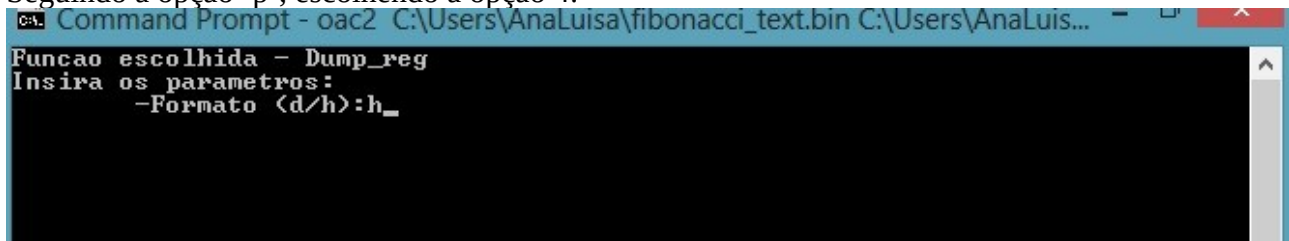
```
CA. Command Prompt - oac2 C:\Users\AnaLuisa\fibonacci_text.bin C:\Users\AnaLuisa... - [X]
Funcao escolhida - Dump_mem
Insira os parametros:
  -Endereco de inicio <0-4095>: 100
  -Endereco de fim <0-4095>: 200
  -Formato <d/h>:h_
```

Seguindo a opção 'p', escolhendo a opção 3, após criação do arquivo:



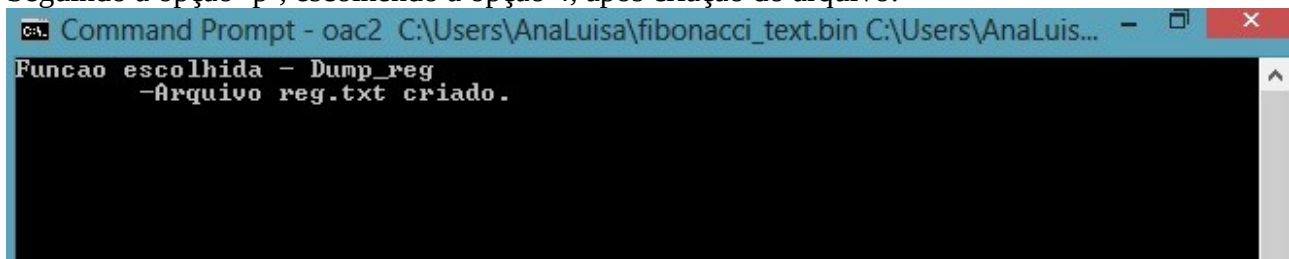
```
C:\> Command Prompt - oac2 C:\Users\AnaLuisa\fibonacci_text.bin C:\Users\AnaLuisa...
Funcao escolhida - Dump_mem
-Arquivo mem.txt criado.
```

Seguindo a opção 'p', escolhendo a opção 4:



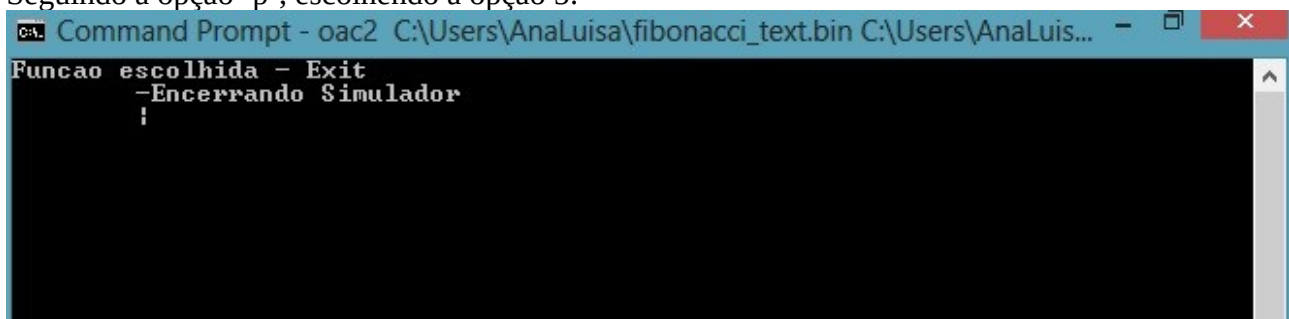
```
C:\> Command Prompt - oac2 C:\Users\AnaLuisa\fibonacci_text.bin C:\Users\AnaLuisa...
Funcao escolhida - Dump_reg
Insira os parametros:
-Formato <d/h>:h_
```

Seguindo a opção 'p', escolhendo a opção 4, após criação do arquivo:



```
C:\> Command Prompt - oac2 C:\Users\AnaLuisa\fibonacci_text.bin C:\Users\AnaLuisa...
Funcao escolhida - Dump_reg
-Arquivo reg.txt criado.
```

Seguindo a opção 'p', escolhendo a opção 5:



```
C:\> Command Prompt - oac2 C:\Users\AnaLuisa\fibonacci_text.bin C:\Users\AnaLuisa...
Funcao escolhida - Exit
-Encerrando Simulador
!
```

Arquivo de memoria.txt:

```
mem.txt - Notepad
File Edit Format View Help
MEMORY:
0x00000190 = 0x00000000
0x00000194 = 0x00000000
0x00000198 = 0x00000000
0x0000019c = 0x00000000
0x000001a0 = 0x00000000
0x000001a4 = 0x00000000
0x000001a8 = 0x00000000
0x000001ac = 0x00000000
0x000001b0 = 0x00000000
0x000001b4 = 0x00000000
0x000001b8 = 0x00000000
0x000001bc = 0x00000000
0x000001c0 = 0x00000000
0x000001c4 = 0x00000000
0x000001c8 = 0x00000000
0x000001cc = 0x00000000
0x000001d0 = 0x00000000
0x000001d4 = 0x00000000
0x000001d8 = 0x00000000
0x000001dc = 0x00000000
0x000001e0 = 0x00000000
0x000001e4 = 0x00000000
0x000001e8 = 0x00000000
0x000001ec = 0x00000000
0x000001f0 = 0x00000000
0x000001f4 = 0x00000000
0x000001f8 = 0x00000000
0x000001fc = 0x00000000
0x00000200 = 0x00000000
0x00000204 = 0x00000000
0x00000208 = 0x00000000
0x0000020c = 0x00000000
0x00000210 = 0x00000000
0x00000214 = 0x00000000
0x00000218 = 0x00000000
0x0000021c = 0x00000000
0x00000220 = 0x00000000
```

Arquivo de registradores.txt:

```
reg.txt - Notepad
File Edit Format View Help
REGISTERS:
$0 = 0x00000000
$1 = 0x00000000
$2 = 0x0000000a
$3 = 0x00000000
$4 = 0x00002034
$5 = 0x0000000c
$6 = 0x00000000
$7 = 0x00000000
$8 = 0x00002030
$9 = 0x00000000
$10 = 0x00000090
$11 = 0x00000037
$12 = 0x00000059
$13 = 0x0000000c
$14 = 0x00000000
$15 = 0x00000000
$16 = 0x00000000
$17 = 0x00000000
$18 = 0x00000000
$19 = 0x00000000
$20 = 0x00000000
$21 = 0x00000000
$22 = 0x00000000
$23 = 0x00000000
$24 = 0x00000000
$25 = 0x00000000
$26 = 0x00000000
$27 = 0x00000000
$28 = 0x00000000
$29 = 0x00000000
$30 = 0x00000000
$31 = 0x00000044
pc = 0x0000004c
hi = 0x00000000
lo = 0x00000000
```