TPC1

* B1.sh

#!/bin/bash

#Set the name of the output file

output\_file="sha1sum.txt"

find "$1" -type f -exec sha1sum "{}" + > "$output\_file"

sort sha1sum.txt > sha1sumSorted.txt

./compara sha1sumSorted.txt $2

* Compara.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

#define MAXC 2048

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

char buffer1[MAXC];

char buffer2[MAXC];

FILE \*fd1=fopen(argv[1], "r");

FILE \*fd2=fopen(argv[2], "r");

bool isdiff=false;

int counter =1;

while(fgets(buffer1,MAXC,fd1) != NULL && fgets(buffer2,MAXC,fd2) != NULL ){

int aux = strcmp(buffer1,buffer2);

if(aux!=0){

printf("#%d: %s\n %s\n",counter,buffer1,buffer2);

isdiff=true;

}

counter++;

}

if(isdiff==false){

printf("OK\n");

}

fclose(fd1);

fclose(fd2);

return 0;

}

TPC2

* Sim.c

/\*

CPU-Mem architecture simulator

AC 2022/23 MIEI FCT/UNL

Adaptado da versão de 2020/21

\*/

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define PROMPT "cmd> " //command line prompt sign

#define MAXSZ 100 //max command line size

#define MAXMEM 4096 // 2^12 34 bit word

#define NoREGS 16

int Mem[MAXMEM]; // simulated memory ( 32 bits/addr)

int Regs[NoREGS];

extern void dorun();

char \*trim(char l[]) {

int first=0;

int last=strlen(l)-1; // last char at string 'l'

while ( isspace(l[first]) ) // ignore "spaces" at start

first++;

while ( isspace(l[last]) ) // ignore "spaces" at the end

last--;

l[last+1]='\0';

return &(l[first]);

}

char \*getarg(char l[]) {

int i=0;

while ( !isspace(l[i]) && l[i]!='\0' )

i++;

l[i]='\0';

i++;

while ( isspace(l[i]) && l[i]!='\0' )

i++;

return &(l[i]);

}

int doload(char arg[]) {

FILE \*f = fopen(arg, "r");

if ( f==NULL ) return 0; // erro

int addr = 0;

int word;

while( fscanf(f, "%x", &word) >0 ) // read next word

Mem[addr++] = word;

fclose(f);

return addr;

}

* Dorun..c

/\*

CPU-Mem architecture

AC 2022/23 MIEI FCT/UNL

32 bit instruction

8 bits instrução 31-24

12 bits para especificar registos 23-20 op1 19-16 op2 15-12 result

12 bits para endereços

\*/

#include <stdio.h>

#define NREGS 16

extern int Mem[];

extern int Regs[];

int signExtension20To32(int val) {

int value = (0x000FFFFF & val);

int mask = 0x00080000;

int sign = (mask & val) >> 19;

if (sign == 1)

value += 0xFFF00000;

return value;

}

void dorun(){

unsigned int pc; // program counter or intruction pointer

unsigned int ir; // instruction register

unsigned int opcode;

unsigned int reg1;

unsigned int reg2;

unsigned int reg3;

int val;

unsigned int address;

unsigned char zero;

unsigned char positivo;

pc = 0;

while( 1 ) {

ir = Mem[pc]; // FETCH

opcode = ir >> 24; // DECODE

reg1 = (ir & 0x00f00000) >> 20;

reg2 = (ir & 0x000f0000) >> 16;

reg3 = (ir & 0x0000f000) >> 12;

address = ir & 0x00000fff;

val = signExtension20To32(ir);

switch( opcode ){ // EXECUTE

case 0x00: /\* HALT \*/

printf("HALT instruction executed\n");

return;

case 0x01: /\* LDI \*/

Regs[reg1]=val;

pc=pc+1;

break;

case 0x02: /\* LOAD \*/

Regs[reg1] = Mem[address];

pc = pc + 1;

break;

case 0x03: /\* STORE \*/

Mem[address] = Regs[reg1];

pc=pc+1;

break;

case 0x04: /\* ADD \*/

Regs[reg3] = Regs[reg1] + Regs[reg2];

if(Regs[reg3]>0){

zero=0;

positivo=1;

}else if(Regs[reg3]<0){

zero=0;

positivo=0;

}else{

zero=1;

positivo=1;

}

pc=pc+1;

break;

case 0x05: /\* SUB \*/

Regs[reg3] = Regs[reg1] - Regs[reg2];

if(Regs[reg3]>0){

zero=0;

positivo=1;

}else if(Regs[reg3]<0){

zero=0;

positivo=0;

}else{

zero=1;

positivo=1;

}

pc=pc+1;

break;

case 0x06: /\* CLEAR \*/

Regs[reg1] = 0;

pc=pc+1;

break;

case 0x08: /\* JMP \*/

pc = address;

break;

case 0x09: /\* JZ \*/

if(zero){

pc = address;

}else{

pc=pc+1;

}

break;

case 0x0A: /\* JNZ \*/

if(!zero){

pc = address;

}

else{

pc=pc+1;

}

break;

case 0x0B: /\* JG \*/

if(!zero && positivo){

pc = address;

}

else{

pc=pc+1;

}

break;

case 0x0C: /\* JGE \*/

if(positivo){

pc = address;

}

else{

pc=pc+1;

}

break;

case 0x0D: /\* JB \*/

if(!zero && !positivo){

pc = address;

}

else{

pc=pc+1;

}

break;

case 0x0E: /\* JBE \*/

if(!positivo || zero){

pc = address;

}

else{

pc=pc+1;

}

break;

default:

printf("Invalid instruction!\n");

return;

}

}

}