# Asembler

# Prevodjenje

Koristi se operativni sistem linux, gnu alat I g++ kompajler . Potrebno je u komandnoj linija napisati sledeću komandu:

g++ Main.cpp Asembler.cpp Exception.cpp -o assembler

Napomena: U tekućem direktorijumu je potrebno da se nadju .cpp I .h fajlovi, program koji se pokreće je assembler.

# Pokretanje

Pre svega je potrebno napraviti objektne fajlove od asemblerskih fajlova. Sledeći proces je potrebno uraditi za svaki asemblerski fajl.

Kako bi se program pokrenuo potrebno je u komandnoj linija napisati sledeću komandu

assembler imeUlaznogfajla -o imeIzlaznogFajla

Izlaz asemblera je fajl imeIzlaznogFajla sa ekstenzijom o, pored ovog fajla se pravi još šjedan fajl čija je namena ucitavanje u emulator. Ime tog fajla je e + imeUlaznogFajla.

imeUlaznog fajla mora da ima .s ekstenziju, dok ime izlaznog fajla mora da ima .o ekstenziju. Moguće je da se ne navede -o imeIzlazogFajla I tada ce se generisati fajl sa defaultnim imenom.

## Opis rešenja

### Asembler

### Prvo je urađeno parsiranje teksta i odlučivanje o kojoj se instrukciji, direktivi radi. Kada se utvrdi o čemu je reč, nastavlja se dalje parsiranje kako bi se dobile dodatne informacije. Kada se prikupe sve neophodne informacie za direktivu ili instruckiju onda se prelazi na procesiranje direktive/instrukcije.

### Svaki put kada se naiđe na upotrebu simbola, dodaje se novi zapis u tabeli obraćanja unapred.

### Ukoliko se radi o upotrebi lterala, on se odmah upisuje.

### Parsiranje ulaznog fajla je uradjeno pomoću regexa.

Ukoliko se radi o instrukciji pre nego što se krene na procesiranje instrukcije, proverava se da li je kombinacija veličine operanda i načina adresiranja dozvoljena.

Ukoliko se radi o PC relativnom adresiranju, upisuje se u kod odgovarajuća dužina do kraja instrukcije kako bi se razrešio posle relokacioni zapis.

Ukoliko se radi o equ direktivi, onda se samo ceo izraz upisuje u tabelu equ izraza.

Kada se završi sa parsiranjem fajla, proveri se da li imamo end direktivu. Zatim se proverava da li postoji neku nedefinisani simbol.

Onda se kreće na računanje equ simbola, odnosno razrešavanje zavisnosti, računanje izraza i određivanje sekcije kojoj simbol pripara. Simbol pripada sekciji od kojoj zavisi. Ukoliko je samo broj onda je sekcija UNKNOWN(-1), a ukoliko zavisi od extern UNDEFINED(0).

Sekcija kojoj simbol pripada i provera da li je izraz ispravan se određuje na osnovu indeksa klasifikacije.

Ovde se takođe prave relokacioni zapisi za globalne equ simbole koji zavise od extern simbola . Ukoliko je equ simbol lokalan, onda se ne pravi equ relokacini zapis.

Kad se razreše simboli kreće se na pravljenje relokacionih zapisa. Tu imamo nekoliko slučajeva:

Ukoliko se radi o simbolu koji pripada sekciji -1, onda se pravi relokacioni samo u slucaju PC relativnog adresiranja, inače se vrednost samo upisuje u kod nezavisno da li je reč o lokalnim ili globalnim simbolima.

Ukoliko se radi o PC relativnom adresiranju a simbol pripada istoj sekciji kao i mesto korišćenja ne pravi se relokacioni zapis, bez obzira da li jerč o lokalnom ili globalnom simbolu.

Ukoliko se radi o globalnom simbolu samo se pravi relokacioni zapis ka tom simboli.

Ukoliko se radi o lokalnom simbolu potrebno je napraviti relokacioni zapis ka sekciji kojoj pripada simbol i dodati vrednost simbola u kod. Dodaje se vrednost zbog pc relativnog, umesto da se samo upisuje.

Ukoliko se radi o undefined simbolu koji je lokalni onda se pravi ka simbolu od koga zavisi simbol i stavlja se znak da li je minus ili plus taj simbol od koga zavisi.

Posle ovoga se kreće na upis u izlazni fajl. Prave se dva izlazna fajla. Jedan koji je pregledan i jedan koji se služi kao ulaz u emulator. U fajl se upisuje tabela simbola, tabela equ relokacionih zapisa, tabela relokacionih zapisa i kod sekcije.

### 

# 

# Документација

## Асемблер

//Vrsta relokacionog zapisa

enum class TypeOfUse { PC\_REL, SYMBOL, SYMBOL\_ONE\_BYTE, EQU\_SYMBOL, EQU\_SYMBOL\_SUBSTRACT, DECIDE\_LATER };

//Služi da opiše jedan relokacioni zapis

struct RealocEntry {

TypeOfUse realocType;

long location;

long symbolTableReference;

bool negativeSymbol;

RealocEntry() {

realocType = TypeOfUse::DECIDE\_LATER; negativeSymbol = false; location = 0; symbolTableReference = 0;

}

};

//Jedan ulaz tabele obračanja unapred

struct SymbolUseEntry {

long sectionOffset;

std::string sectionName;

TypeOfUse typeOfUse;

SymbolUseEntry() {

sectionOffset = 0; sectionName = ""; typeOfUse = TypeOfUse::DECIDE\_LATER;

}

};

//Jedan ulaz u tabelu simbola

struct SymbolEntry {

static long nextSymbolNumber;

std::string name;

long sectionNumber;

long value;

long size;

bool isGlobal;

bool isDefined;

std::vector<SymbolUseEntry> symbolUseTable;

long number;

SymbolEntry() {

name = ""; sectionNumber = 0; value = 0; isGlobal = false; isDefined = false; number = 0; size = -1;

}

};

//Jedan ulaz u tabelu sekcija

struct SectionEntry {

std::string name;

long startAdress;

long size;

std::vector<char> code;

std::vector<RealocEntry>realocTable;

long number;

SectionEntry() {

name = ""; startAdress = 0; size = 0; number = 0;

}

};

//Opis jednog dela izraza preko kojeg je definisan equ simbol

struct EquEntryPart {

std::string symbol;

char sign;

EquEntryPart(std::string symbol, char sign) {

this->symbol = symbol;

this->sign = sign;

}

EquEntryPart() {

this->symbol = "";

this->sign = '+';

};

};

//Jedan ulaz u tabelu equ simbola

struct EquEntry {

std::string symbol;

std::vector<EquEntryPart> symbolValue;

int value;

int realocSectionNumber;

bool isDefined;

EquEntry(std::string symbol, std::vector<EquEntryPart> symbolValue) {

this->symbol = symbol;

this->symbolValue = symbolValue; equLine = "0"; lineNumber = 0; isDefined = false; value = 0;

}

EquEntry() { symbol = ""; isDefined = false; }

std::string equLine;

long lineNumber;

};

//Nacin adresiranja

enum class AdressMode { IMD, REGDIR, REGIND, REGINDPOM, MEM };

//Informacije o jednom operandu

struct Operand {

AdressMode addressMode;

std::string symbol;

char usedRegister;

char operandSize;

char usedPart;

unsigned char operandDescriptionCode;

Operand() {

this->addressMode = AdressMode::IMD;

usedRegister = 0; operandSize = 0; usedPart = 0; usedRegister = 0;

operandDescriptionCode = 0;

}

};

//Informacije o jednoj instrukciji

struct Instruction {

std::string instructionName;

char operandNumber;

char operandSize;

bool excplicitSize;

Operand operand1, operand2;

unsigned char instructionDescriptionCode;

Instruction() {

instructionName = ""; operandNumber = 0; operandSize = 0; excplicitSize = false; instructionDescriptionCode = 0;

}

};

//Klasa koja predstavlja asembler

class Asembler{

private:

//Ime instrukcije i kod koji joj odgovara

static std::map<std::string, unsigned char> instructionsCode;

// Da li je dato ime izlaznog fajla ili ne

enum AsemblerOptions {USE\_DEFAULT\_FILE\_NAME = 2, FILE\_NAME\_GIVEN = 4};

//Specijalne oznake za definisanost simbola

enum DefaultSections {UNKNOWN = -1, UNDEFINED = 0};

//Opseg za bajt i za word

enum ByteRange {LOW\_BYTE = 0, HIGH\_BYTE = 255, LOW\_WORD = -32768, HIGH\_WORD = 32767};

enum SpecialRegister {PC = 7};

enum PC\_REL\_OFFSET\_VALUE {PC\_REL\_ONE\_OPERAND = -2};

//Velicina opernada

enum InstructionSize{ONE\_BYTE = 0, TWO\_BYTES = 1};

enum OperandSize{ONE\_BYTE\_OPERAND = 1, TWO\_BYTE\_OPERAND = 2};

//Znak dela equ izraza

enum Sign:char { PLUS = 1, MINUS = -1 };

static std::string equRealloc;

const static int expectedNumberOfArguments = 3;

const char\* outputFileName;

const char\* inputFileName;

static Asembler\* asembler;

static long locationCounter;

static bool isEnd;

static long lineCounter;

static std::string currentLine;

static long currentSectionNumber;

static std::string currentSectionName;

//Tabela simbola

static std::map<std::string,SymbolEntry> symbolTable;

//Tabela sekcija

static std::map<std::string,SectionEntry> sectionTable;

//Tabela equ simbola i izraza za date simbole

static std::map<std::string,EquEntry> equTable;

static void makeAsembler(char\* inputFile, const char\* output = "duca.out");

Asembler(const char\* inputFile, const char\* outputFile);

//Sluze za parsiranje teksta i odluku da li je rec o odgovarajucoj naredbi/instrukciji

bool checkIfLineIsDirective(std::string assemblyLine);

bool checkIfGivenDirective(std::string assemblyLine, std::string directive, void(\*obradiSimbol)(std::string param));

std::string checkifLabelAtBeginig(std::string assemblyLine);

bool checkForNothingInLine(std::string assemblyLine);

bool checkIfSection(std::string assemblyLine);

bool checkIfEqu(std::string assemblyLine);

bool checkIfEnd(std::string assemblyLine);

bool checkIfSkip(std::string assemblyLine);

//Sluzi sa upis simbola/literala u tabelu obracanja unapred i odgovarjuce informacije u kod sekcije

static void newSymbolUse(std::string symbol, TypeOfUse typeOfUse, long symbolValue = 0);

static void newByteLiteralUse(long simbol);

static void newWordLiteralUse(long simbol);

static void setSymbolInSection(long number, std::string sectionToSet, long startPosition, bool halfSymbol);

//Sluze za obradu odgovarajuce direktive.

static void processGlobalDirective(std::string symbol);

static void processExternDirective(std::string symbol);

static void processByteDirective(std::string symbol);

static void processWordDirective(std::string symbol);

static void processLabelDirective(std::string label);

static void processSectionDirective(std::string section);

static void processEndDirective();

static void processEquDirective(std::string simbol, std::vector<EquEntryPart> equParts);

static void processSkipDirective(std::string simbol);

//Sluze da li je rec o odgovarajucoj instrukciji

bool checkIfInstruction(std::string assemblyLine);

bool checkInstructionOperandNumber(std::string assemblyLine, std::regex instructionDescription, bool(\*makeInstruction)(std::smatch parsedInstruction));

//Prave instrukciju sa odgovarajucim brojem parametara

static bool makeInstructionWithoutParameter(std::smatch parsedInstruction);

static bool makeInstructionWithOneParameter(std::smatch parsedInstruction);

static bool makeInstructionWithTwoParameter(std::smatch parsedInstruction);

static bool makeInstructionWithOneParameterJump(std::smatch parsedInstuction);

//Procesuiraju napravljenu instrukciju tako sto proizvode kod u sekciji

static void processInstructionWithoutParameter(Instruction instruction);

static void processInstructionWithOneParameter(Instruction instruction);

static void processInstructionWithTwoParameter(Instruction instruction);

/Sluzi da se na osnovu operanada i njihovih nacina adresiranja odredi velicina operanada instrukcije

static void decideInstructionOperandsSize(Instruction &instruction);

static int calculateInstructionOperandSize(Operand &operand);

//Procesuira operand instrukcije, setuje mu odgovartajuce informacije, nacin adresiranja, kao i dodadatne informacije specificne za dati način adresiranja

static void processOperand(Operand& operand, int nextOperandSize = 0);

static bool checkAndSetAdressMode(Operand& operand, std::string adressModeDescription, bool isJump);

static char decideRegisterNumber(std::string registerNumberDescription);

static bool setImdAdressMode(Operand& instruction, std::smatch parsedOperand);

static bool setRegDirAdressMode(Operand& instruction, std::smatch parsedOperand);

static bool setRegIndAdressMode(Operand& instruction, std::smatch parsedOperand);

static bool setRegIndPomAdressMode(Operand& instruction, std::smatch parsedOperand);

static bool setMemDirAdressMode(Operand& instruction, std::smatch parsedOperand);

//Sluzi za razresavanje equ simbola i proveru kruzne zavisnosti

static void resolveEquSymbols();

static void checkIfHaveUndefinedSymbols();

static void ResolveSymbolValuesAndMakeRealocationEntries();

static void setBitsInChar(unsigned char& destination, unsigned char bitsToSet, unsigned char positionFromStart, unsigned char numberOfBitsToSet) {

if (positionFromStart + numberOfBitsToSet > 8) throw std::exception("Nema Mesta");

destination |= bitsToSet << (8 - positionFromStart - numberOfBitsToSet);

}

//Sluzi za formiranje izlaznog fajla, za korisnika i za emulator

void writeSectionTable(std::fstream &outFile);

void writeSymbolTable(std::fstream &outFile);

void writeSectionsRealocations(std::fstream &outFile);

void writeSectionsCode(std::fstream &outfile);

void writeFileForEmulator();

public:

//Provera na koji nacin je pozvan program

static bool checkProgramCall(int argc, char\* argv[]);

static Asembler\* getAsembler();

//Prevodjenje ulaznog asemblerskog fajla

void processInputFile();

//Generisanje izlaznog fajla

void writeToOutputFile();

static void deleteAsembler();

};

Klase koje sluze kao izuzeci

#ifndef \_exception\_h

#define \_exception\_h

#include <exception>

#include <string>

class AsemblerNoTFound : public std::exception {

public:

virtual const char\* what() const noexcept {

return "Asembler nije napravljen\n";

};

};

class FileError : public std::exception {

public:

virtual const char\* what() const noexcept {

return "Greska sa fajlom\n";

};

};

class ArgumentsFormatNotRigth : public std::exception {

std::string message;

public:

ArgumentsFormatNotRigth(std::string message) {

this->message = "Ulazni parametri nisu odgovarajuci\nFormat: asembler [-o imeIzlaznog.o] imeUlazno.s\nVas ulaz:" + message + "\n";

}

virtual const char\* what() const noexcept {

return this->message.c\_str();

};

};

class InstructionArgumentsNotRight : public std::exception {

std::string message;

public:

InstructionArgumentsNotRight(std::string line, long number) {

this->message = "Greska, operandi instrukcije:";

this->message += std::to\_string(number);

this->message +=". ";

this->message += line;

}

virtual const char\* what() const noexcept {

return this->message.c\_str();

};

};

class LineNotRecognized : public std::exception {

protected:

std::string message;

public:

LineNotRecognized(std::string line, long number, std::string aditionalMessage ="Greska, linija nije prepoznata:" ) {

this->message = aditionalMessage;

this->message += std::to\_string(number);

this->message += ". ";

this->message += line;

}

LineNotRecognized() {}

virtual const char\* what() const noexcept {

return this->message.c\_str();

};

};

class SectionCantBeGlobal : public LineNotRecognized {

public:

SectionCantBeGlobal(std::string line, long number): LineNotRecognized(line,number, "Sekcija ne moze biti globalna") {

}

};

class SectionCantBeImported : public LineNotRecognized {

public:

SectionCantBeImported(std::string line, long number) : LineNotRecognized(line, number, "Sekcija se ne moze izvoziti") {

}

};

class DefinedSymbolCantBeImported : public LineNotRecognized {

public:

DefinedSymbolCantBeImported(std::string line, long number) : LineNotRecognized(line, number, "Definisan simbol se ne moze uvoziti") {

}

};

class SymbolMarkedToExportCantBeImported : public LineNotRecognized {

public:

SymbolMarkedToExportCantBeImported(std::string line, long number) : LineNotRecognized(line, number, "Simbol koji se izvozi se ne moze uvoziti") {

}

};

class SymbolMakredToImportCantBeExported : public LineNotRecognized {

public:

SymbolMakredToImportCantBeExported(std::string line, long number) : LineNotRecognized(line, number, "Simbol koji se uvozi se ne moze izvoziti") {

}

};

class ByteOutOfRange : public LineNotRecognized {

public:

ByteOutOfRange(std::string line, long number) : LineNotRecognized(line, number, "Byte je opsega od -128 do 127") {

}

ByteOutOfRange() {

message = "Van opsega, Byte je opsega od -128 do 127";

}

};

class WordOutOfRange : public LineNotRecognized {

public:

WordOutOfRange(std::string line, long number) : LineNotRecognized(line, number, "Word je opsega od -32768 do 32767") {

}

WordOutOfRange() {

message = "Van opsega, Word je opsega od -32768 do 32767";

}

};

class SectionNotStarted : public LineNotRecognized {

public:

SectionNotStarted(std::string line, long number) : LineNotRecognized(line, number, "Sekcija nije zapoceta") {

}

};

class CantUseSectionNameAsSymbolValue : public LineNotRecognized {

public:

CantUseSectionNameAsSymbolValue(std::string line, long number) : LineNotRecognized(line, number, "Ime sekcije se ne moze koristiti kao simbol") {

}

};

class AlreadyDefinedSymbolWithThatName : public LineNotRecognized {

public:

AlreadyDefinedSymbolWithThatName(std::string line, long number) : LineNotRecognized(line, number, "Vec postoji simbol definisan sa datim imenom") {

}

};

class CantHaveSectionNameSameAsSymbolName : public LineNotRecognized {

public:

CantHaveSectionNameSameAsSymbolName(std::string line, long number) : LineNotRecognized(line, number, "Ime sekcije ne moze biti isto kao ime simbola") {

}

};

class CantHaveSymbolNameSameAsSectionName : public LineNotRecognized {

public:

CantHaveSymbolNameSameAsSectionName(std::string line, long number) : LineNotRecognized(line, number, "Ime simbola ne moze biti isto kao ime sekcije") {

}

};

class IMDCantBeDestination : public LineNotRecognized {

public:

IMDCantBeDestination(std::string line, long number) : LineNotRecognized(line, number, "Neposredna vrednost ne moze biti odrediste") {

}

};

class OperandSizeNotRight : public LineNotRecognized {

public:

OperandSizeNotRight(std::string line, long number) : LineNotRecognized(line, number, "Velicina operanda nije odgovarajuca") {

}

};

class CantDefinedImportedSymbol : public LineNotRecognized {

public:

CantDefinedImportedSymbol(std::string line, long number) : LineNotRecognized(line, number, "Ne mozete definisati simbol koji se uvozi!") {

}

};

class EndDirectiveNotFound : public std::exception {

public:

virtual const char\* what() const noexcept {

return "U ulaznom fajlu nije pronadjena .end direktiova\n";

};

};

class SymbolNotDefined : public std::exception {

std::string message;

public:

SymbolNotDefined(std::string name) {

this->message = "Simbol nije definisan ";

this->message += name;

this->message += "\n";

}

virtual const char\* what() const noexcept {

return this->message.c\_str();

};

};

class CantDefinedSimbolUsingYourself : public LineNotRecognized {

public:

CantDefinedSimbolUsingYourself(std::string line, long number) : LineNotRecognized(line, number, "Ne mozete definisati simbol koristexi sebe") {

}

};

class EquDefinitionError : public LineNotRecognized {

public:

EquDefinitionError(std::string line, long number) : LineNotRecognized(line, number, "Neispravna definicija equ-a!") {

}

};

class EquSymbolCantBeResolved : public LineNotRecognized {

public:

EquSymbolCantBeResolved(std::string line, long number) : LineNotRecognized(line, number, "Equ se ne moze razresiti!") {

}

};

class JumpAdressCanyBeLessThenZero : public LineNotRecognized {

public:

JumpAdressCanyBeLessThenZero(std::string line, long number) : LineNotRecognized(line, number, "Adresa skoka ne moze biti manja od nule!") {

}

}; #endif