Dakle ovako,

Prvo procitas tekst projekta dva puta, onda procitas opis asemblera Zadatak je da od ulaznog fajla u kome je niz instrukcija napravis izlazni koji je rezultat asembliranja. A rezultat asembliranja je **tabela simbola**, **relokacioni zapisi** za svaku sekciju i **masinski kod** prevedenih instrukcija svake sekcije.

Mi pravimo **dvoprolazni asembler**, sto znaci da program prolazi kroz ulazni kod dva puta. U prvom prolazu popunjava se tabela simbola, u drugom relokacione tabele I generise se masinski kod.

Primer koji je Sasa dao u tekstu zadatka:

Primer ulaznog programa:

.data

a: DD a

ORG 0x20

.text

LOAD R1, a ; ucitava sadrzaj memorijske lokacije a u registar R1

LOAD R2, #a ; ucitava adresu lokacije a u registar R2

LOAD R3, $a ; ucitava sadrzaj lokacije a u registar R3 koristeci PC relativno adresiranje

x: JZ R0, x ; apsolutni skok na lokaciju x

JZ R0, $x ; PC relativni skok na lokaciju x

.end

**Tabela simbola**

Prvo procitaj u tekstu zadatka sta zapravo ona predstavlja i koji sve tacno podaci treba da budu u njoj. Evo ukratko:

Popunjava se u **prvom prolazu** kroz kod. Sluzi da se popisu sve sekcije i simboli i trazeni podaci o njima. Tako izdvojeni podaci o simbolima olaksavaju generisanje koda instrukcije koja referise neki simbol.

(npr. u Sasinom primeru, u prvom prolazu u tabelu simbola stavljas a i x u prvom prolazu, pa posle kad naidjes na njih u instrukciji kao sto je LOAD R1, #a samo potrazis simbol sa tim nazivom u tabeli simbola i u bajt propisan formatom instrukcije upises vrednost od a).

Dakle, prvo radi prvi prolaz. Dobro razmisli o tome kakvu ces strukturu da napravis I kako ces da sto lakse organizujes tabelu simbola. Ja o tome nisam razmisljala pa mi je dosta glomazan kod. (ja imam klasu za kao ulaz u tabelu simbola isti je i za simbol i za sekciju i onda ulancavam te ulaze).

Kad si to smislila ides kroz kod liniju po liniju I gledas da li nesto iz te linije treba ubaciti u TS. Za sekcije je lako, sve imaju format da pocinju sa tackom (.text, .data, .rodata), a sto se simbola tice, mogu da se definisu u labeli ili preko DEF-a cini mi se. Dakle, prvi deo je manje vise samo obrada teksta. Neki su koristili regex biblioteku za proveru poklapanja forme stringa sa nekom formom koju ti ocekujes, mada ja sam sve rucno samo sa operacijama za string. Ako proucis regex mozda ce ti dosta olaksati, mada sam cula da je nekima zbog verzije virtuelne masine bagovalo.

Kad si pronasla simbol moraces i da sracunas vrednost, adresu itd. samog simbola.   
  
U slucaju labele vrednost simbola ce biti pozicija na kojoj se simbol nasao u sekciji. Zbog toga ces morati da imas neki location counter koji ce brojati koliko mesta koja instrukcija zauzima i kako prolazis kroz kod tako ce se uvezavati za odgovarajuci pomeraj.

U slucaju da je simbol definisan sa DEF iza def moze da stoji neki konstantan izraz npr 2+4\*(7-3) itd. I on odredjuje vrednost simbola. To je jos jedna potencijalno problematicna stvar. Moraces nekako da sredis da se iz stringa izraza dolazi do int resenja, sto je malo tehnicki zez jer je glupost a oduzme dosta vremena, u principu treba prebacivari u onaj postfix oblik sto smo radili na asp-u itd... Vidi vec u mom kodu kako otprilike izgleda.

\*simbol moze I da se uveze/izveze sa direktivom .global, sto znaci da kad naidjes na .global ili moras da updatujes flag vec postojeceg simbola u tabeli na G ili da uneses novi simbol sa flag-om G.

U slucaju sekcija, svakoj ces davati pocetnu adresu nula, osim onima kod kojih naidjes iznad definicije sekcije na ORG deklaraciju. Org ti postavlja sekciju obavezno na adresu koju navedes posle org-a. Npr. ORG 0x20 .data Znaci da ce data sekcija umesto da pocne na adresi nula poceti na adresi 0x20. I onda moras da pratis velicine tih orgovanih sekcija i da prijavis gresku ako se neka od njih poklopila. Ako se preklope orgovane, nece moci da se smeste u memoriju. Opet velicinu sekcije gledas kao zbir svega sto zauzima memoriju u toj sekciji, a to mogu da budu instrukcije i one DD definicije, u principu dokle dodje location counter to je velicina sekcije.

Okej kad zgotovis prvi prolaz na konju si, jer je ostalo manje vise ceprkanje po tome.

**Drugi prolaz**

E sad, u drugom prolazu popunjavas **relokacione zapise** i generises **masinski kod** za svaku od sekcija.

Opet, ides kroz kod liniju po liniju i prvenstveno popunjavas memoriju za svaku od sekcija tj. generises masinski kod.

Kad naidjes na DD DW DB popunjavas cetiri, dva ili jedan bajt vrednoscu koja je navedena posle DD..

Kad naidjes na instrukciju, ukoliko nema referenci na neke simbole samo je dekodujes u njenu vrednost po formatu koji je Sasa dao u prilogu. Ukoliko postoji referenca na simbol, mora malo mozganja oko toga koji je nacin adresiranja u pitanju, da li je simbol iz org-ovane sekcije ili ne, da li je globalan ili lokalan, I na osnovu toga donosis zakljucak da li je potrebno ostaviti relokacioni zapis. U to ime, par reci o relokcionim zapisima:

**Relokacioni zapisi**

Dakle, kod popunjavanja TS pretpostavljamo da sve sekcije koje nisu ORG-ovane krecu od pocetne adrese nula. To je u realnosti nemoguce, jer bi se preklapale, i svakoj sekciji se nakon loadovanja u pravu memoriju, adresa menja sa nule na neku realnu prvu slobodnu pronadjenu adresu gde moze da se smesti celo parce te sekcije. Za ORG-ovanu sekciju vazi posebno pravilo, a to je da ona MORA da se smesti tacno tamo gde je org rekao da ce ona da ide. E sad posledica ove situacije jeste da labele koje su bile definisane u neorgovanim sekcijama zapravo nece imati tu vrednost koju sada imaju u tabeli simbola, vec neku drugu u zavisnosti od toga gde se smesti sekcija u memoriju. Posledica toga jeste da ti moras da ostavis negde zapis o tome koje memorijske lokacije ce morati da se azuriraju nakon punjenja memorije, a to su sve one lokacije u kojima si se pozivala na simbole iz neorgovanih sekcija. Takodje, za Pcrel moraces da ostavljas reloc zapis I za sam PC ako se nalazis u neorgovanoj sekciji, jer PC nece citati instrukciju sa te adrese na kojoj asembler misli da ce se instrukcija nalaziti, vec neku drugu vrednost.

Ovu celu zavrzlamu mi je aleksa sjajno objasnio tacno po slucajevima, pa se prokonsultuj s njim kad dodjes do toga. Poenta price je da ides liniju po liniju napises kod koji prepoznaje I desifruje instrukciju u odgovarajuci masinski kod I ako je potrebno, ubaci odredjenu adresu u relokacioni zapis I ostavi podatak sta treba cime da se zameni u tom relokacionom zapisu. Opet naprvis neku strukturu za relokacioni zapis.

Okej, ne znam malo sam ti smuljala ovaj drugi prolaz, ali kad dodjes do toga, zovi.

Emulator ti saljem tacno sta mi je Vurdelja rekao da radim jer je on to mnogo bolje napisao nego sto ja znam.

**EMULATOR**

znaci emulator radis po sledecim koracima

prvo iz izlaznog fajla asemblera, a ulaznog emulatora, na osnovu njega napravis opet tabele simbola, relokacija, sekcija

onda radis linkovanje

ja sam radio linkovanje i pre kreiranja same memorije

onda posle linkovanja treba da smestis sve to u memoriju

e sad memorija ne bi trebalo da ti bude niz recimo, jer ako treba da emulras 4GB jasno je da to puca

znaci da bi trebala da radis otprilike kao stranicenje u virtuelnoj memoriji, znaci imas ucitane neke stranice, a ostatak memorije cuvas na disku(u fajlu recimo)

e posto ja nisam hteo da se cimam sa stranicenjem, ja sam lepo cuvao celu memoriju u fajlu i kad mi treba da citam memoriju ja citam iz fajla, sto je sporo, al on ne gleda kod, niti se to primecuje

sto se tice prekida

imas dva izbora, da radis u vise niti ili u jednoj niti

ja sam radio u jednoj niti sve

i to radis tako sto prvo uzmes emuliras instrukciju, kad zavrsis emuliranje instrukcije proveris da li je doslo do prekida, ako jeste, ucitas adresu prekidne rutine u PC

registrovanje prekida radis iz C/C++-a, a sam sadrzaj prekidnih rutina pises na asembleru

to je tvoj test primer

uglavnom trazio je na odbrani da mu pokazes kako bi iskoristila te prekidne rutine da ucitas nesto sa ulaza i da ispises na izlaz

msm meni najzajebanije su bili prekidi, zato sto nisam shvatio sta treba da radim

ali odmah da ti olaksam, radi sve u jednoj niti i koristi na windowsu conio.h biblioteku

koja ima kbhit

koja registruje pritisak tastera

ali to kad budes dosla do prekida, radi to tek na kraju

**good luck ;)**