Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Sistemski softver (13E113SS, 13S113SS)
Nastavnik: doc. dr Saša Stojanović
Školska godina: 2022/2023. (Zadatak važi počev od junskog roka 2023.)

Projekat za domaći rad
– Projektni zadatak –

Verzija dokumenta: 1.3

Važne napomene: Pre čitanja ovog teksta, **obavezno** pročitati opšta pravila predmeta i pravila vezana za izradu domaćih zadataka! Pročitati potom ovaj tekst **u celini i pažljivo**, pre započinjanja realizacije ili traženja pomoći. Ukoliko u zadatku nešto nije dovoljno precizno definisano ili su postavljeni kontradiktorni zahtevi, student treba da uvede razumne pretpostavke, da ih temeljno obrazloži i da nastavi da izgrađuje preostali deo svog rešenja na temeljima uvedenih pretpostavki. Zahtevi su namerno nedovoljno detaljni, jer se od studenata očekuje kreativnost i profesionalni pristup u rešavanju praktičnih problema!

Uvod

Cilj ovog projekta jeste realizacija alata u lancu prevođenja i emulatora za apstraktni računarski sistem. Opis apstraktnog računarskog sistema dat je u prilogu. Alati u lancu prevođenja koje treba realizovati obuhvataju asembler za navedeni apstraktni računarski sistem i linker nezavisan od ciljne arhitekture.

Rešenje projekta obuhvata izvorni kod kojim su implementirani asembler, linker i emulator. U daljem tekstu, rešenje projekta biće kratko nazivano samo *implementacija*. Izvorni asemblerski kod napisan za apstraktni računarski sistem, koji će asembler prevoditi, linker povezivati i emulator izvršavati, biće u nastavku referisan kao *korisnički program* kako bi se jasno napravila razlika u odnosu na *implementaciju*.

U cilju demonstracije i odbrane projekta, *implementacija* mora da se uspešno izvršava pod Linux operativnim sistemom kao konzolna aplikacija. Odbrana projekta pod Windows operativnim sistemom nije moguća.

Opšti zahtevi

Leksička analiza

Implementacija leksičke analize i parsiranja ulaznih tekstualnih datoteka ne predstavlja glavni zadatak projekta, ali jeste neophodna za uspešnu izradu rešenja, zbog čega je za te potrebe dozvoljeno je koristiti generatore leksera i parsera. Bez ulaska u detalje i namenu ovih alata, koji mogu biti korisni prilikom izrade rešenje, skreće se pažnja da su na virtuelnoj mašini instalirani alati *flex* i *bison* koji predstavljaju upravo generatore leksera i parsera respektivno. Pored generatora leksera i parsera dozvoljeno je i preporučeno koristiti standardne biblioteke kao što je STL (*Standard Template Library*). Takođe, dozvoljeno je koristiti i nestandardne biblioteke ukoliko one ne implementiraju stvari usko vezane sa srž projekta u koju spada generisanje mašinskog koda, pratećih metapodataka, formiranje relokacionih zapisa, relociranje i povezivanje predmetnih programa, emulacija itd.

Ocenjivanje projekta

Zahtevi u okviru zadataka navedenih u nastavku razvrstani su prema težini i obimu na tri nivoa: A (30 poena), B (35 poena) i C (40 poena). Podela zahteva po nivoima za svaki od zadataka definisana je neposredno nakon uvoda posmatranog zadatka. Prilikom odbrane projekta moguće je ostvariti:

- 30 poena, ako i samo ako su tačno urađeni svi zahtevi propisani za nivo A
- 35 poena, ako i samo ako su tačno urađeni svi zahtevi propisani za nivoe A i B
- 40 poena, ako i samo ako su tačno urađeni svi zahtevi propisani za nivoe A, B i C

Ukoliko student ne implementira ispravno sve zahteve propisane za nivo A za svaki od zadataka nije moguće uopšte pristupiti odbrani projekta. Angažovani na predmetu zadržavaju diskreciono pravo da odbranu projekta označe kao neuspešnu ukoliko ustanove da predato rešenje projekta sadrži grešku prilikom izvršavanja ili na bilo koji način odstupa od zahteva projektnog zadatka.

Zadatak 1: Asembler

Uvod

Cilj ovog zadatka jeste realizacija dvoprolaznog asemblera za procesor opisan u prilogu. Ulaz asemblera je tekstualna datoteka sa izvornim asemblerskim kodom napisanim u skladu sa sintaksom opisanom u nastavku. Izlaz asemblera je tekstualna datoteka koja predstavlja predmetni program (dozvoljeno je generisati kao izlaz, pored tekstualne datoteke, binarnu datoteku radi jednostavnijeg učitavanja izlaza asemblera u linker).

Format predmetnog programa bazirati na školskoj varijanti ELF formata čiji je referentni primer tekstualna datoteka kakva je korišćena na vežbama u delu gradiva koje se tiče konstrukcije asemblera. Dozvoljeno je praviti izmene u školskoj varijanti ELF formata (sekcije, tipovi zapisa o relokacijama, dodatna polja u postojećim tipovima zapisa, novi podaci o predmetnom programu i slično) ukoliko je to neophodno u skladu sa potrebama ciljne arhitekture. Prilikom razrešavanja svih nedefinisanih detalja za potrebe rešenja voditi se principima koje koristi GNU asembler.

Podela zahteva po nivoima

Podela zahteva po nivoima za ovaj zadatak tiče se asemblerskih direktiva i naredbi (ostale zahteve podrazumevano treba implementirati). Nivo u sklopu kojeg treba implementirati neku asemblersku direktivu naveden je u uglastim zagradama ispred tražene asemblerske direktive. Nivo u sklopu kojeg treba implementirati neku asemblersku naredbu naveden je unutar reda odgovarajuće tabele za posmatranu asemblersku naredbu.

Pokretanje iz terminala

Rezultat prevođenja *implementacije* ovog zadatka treba da ima `asembler` za naziv. Sve informacije potrebne za izvršavanje zadaju se kao argumenti komandne linije. Jednim pokretanjem `asembler` vrši asembliranje jedne ulazne datoteke. Naziv ulazne datoteke sa izvornim asemblerskim kodom zadaje se kao samostalni argument komandne linije. Način pokretanja `asembler` jeste sledeći:

```
asembler [opcije] <naziv_ulazne_datoteke>
```

Opcije komandne linije

Opis opcija komandne linije, zajedno sa opisom njihovih parametara, koje mogu biti zadate prilikom pokretanja `asembler` nalazi se u nastavku:

`-o <naziv_izlazne_datoteke>`

Opcija komandne linije `-o` postavlja svoj parametar `<naziv_izlazne_datoteke>` za naziv izlazne datoteke koja predstavlja rezultat asembliranja.

Primer pokretanja

Primer komande, kojom se inicira asembliranje izvornog koda u okviru datoteke *ulaz.s* sa ciljem dobijanja *izlaz.o* predmetnog programa, dat je u nastavku:

```
./asembler -o izlaz.o ulaz.s
```

Sintaksa izvornog asemblerskog koda

Sintaksa izvornog asemblerskog koda može se grubo podeliti na opšte detalje, asemblerske direktive i asemblerske naredbe. Opšti detalji definišu izgled jedne linije izvornog koda po pitanju zapisa labela, asemblerskih naredbi, asemblerskih direktivi, komentara itd. Asemblerska direktiva predstavlja operaciju koju asembler treba da izvrši u toku asembliranja. Asemblerska naredba predstavlja simbolički zapis mašinskih instrukcija koji asembler treba da prevede u binarnu reprezentaciju.

Opšti detalji

Opšti detalji, predstavljeni u vidu (1) funkcionalnih zahteva koje asembler treba da ispuni i (2) sintaksnih pravila za koja asembler treba da proveri da li su ispoštovana, navedeni su redom po stavkama u nastavku:

- jedna linija izvornog koda sadrži najviše jednu asemblersku naredbu ili direktivu,
- zakomentarisan sadržaj se ignoriše u potpunosti prilikom asembliranja,
- jednolinijski komentar, koji se implicitno završava na kraju linije, započinje # karakterom,
- labela, čija se definicija završava dvotačkom, mora se naći na samom početku linije izvornog koda (opciono nakon proizvoljnog broja belih znakova) i
- labela može da stoji i samostalno, bez prateće asemblerske naredbe ili asemblerske direktive u istoj liniji izvornog koda, što je ekvivalentno tome da stoji na samom početku prve naredne linije izvornog koda koja ima sadržaj.

Asemblerske direktive

`[nivo A] .global <lista_simbola>`

Izvozi simbole navedene u okviru liste parametara. Lista parametara može sadržati samo jedan simbol ili više njih razdvojenih zapetama.

`[nivo A] .extern <lista_simbola>`

Uvozi simbole navedene u okviru liste parametara. Lista parametara može sadržati samo jedan simbol ili više njih razdvojenih zapetama.

`[nivo A] .section <ime_sekcije>`

Započinje novu asemblersku sekciju, čime se prethodno započeta sekcija automatski završava, proizvoljnog imena navedenog kao parametar asemblerske direktive.

`[nivo A] .word <lista_simbola_ili_literala>`

Alocira prostor fiksne veličine po četiri bajta za svaki inicijalizator (simbol ili literal) naveden u okviru liste parametara. Lista parametara može sadržati samo jedan inicijalizator ili više njih razdvojenih zapetama. Asemblerska direktiva alocirani prostor inicijalizuje vrednošću navedenih inicijalizatora.

`[nivo A] .skip <literal>`

Alocira prostor čija je veličina jednaka broju bajtova definisanom literalom navedenim kao parametar. Asemblerska direktiva alocirani prostor inicijalizuje nulama.

[nivo B] .ascii <string>

Alocira prostor fiksne veličine po jedan bajt za svaki karakter stringa (niska karaktera između znakova navoda). Asemblerska direktiva alocirani prostor inicijalizuje vrednostima koje odgovaraju navedenim karakterima prema ASCII tabeli.

[nivo C] .equ <novi_simbol>, <izraz>

Definiše novi simbol čija je vrednost jednaka navedenom izrazu.

[nivo A] .end

Završava proces asembliranja ulazne datoteke. Ostatak ulazne datoteke se odbacuje odnosno ne vrši se njegovo asembliranje.

Asemblerske naredbe

Format	Efekat	Nivo
halt	Zaustavlja izvršavanje instrukcija	A
int	Izaziva softverski prekid	A
iret	pop pc; pop status;	A
call *operand*	push pc; pc <= *operand*;	A
ret	pop pc;	A
jmp *operand*	pc <= *operand*;	A
beq %*gpr1*, %*gpr2*, *operand*	if (*gpr1* == *gpr2*) pc <= *operand*;	A
bne %*gpr1*, %*gpr2*, *operand*	if (*gpr1* != *gpr2*) pc <= *operand*;	A
bgt %*gpr1*, %*gpr2*, *operand*	if (*gpr1* signed> gpr2) pc <= *operand*;	A
push %*gpr*	sp <= sp - 4; mem32[sp] <= *gpr*;	A
pop %*gpr*	*gpr* <= mem32[sp]; sp <= sp + 4;	A
xchg %*gprS*, %*gprD*	temp <= *gprD*; *gprD* <= *gprS*; *gprS* <= temp;	A
add %*gprS*, %*gprD*	*gprD* <= *gprD* + *gprS*;	A
sub %*gprS*, %*gprD*	*gprD* <= *gprD* - *gprS*;	A
mul %*gprS*, %*gprD*	*gprD* <= *gprD* * *gprS*;	A
div %*gprS*, %*gprD*	*gprD* <= *gprD* / *gprS*;	A
not %*gpr*	*gpr* <= ~*gpr*;	A
and %*gprS*, %*gprD*	*gprD* <= *gprD* & *gprS*;	A
or %*gprS*, %*gprD*	*gprD* <= *gprD* | *gprS*	A
xor %*gprS*, %*gprD*	*gprD* <= *gprD* ^ *gprS*;	A
shl %*gprS*, %*gprD*	*gprD* <= *gprD* << *gprS*;	A
shr %*gprS*, %*gprD*	*gprD* <= *gprD* >> *gprS*;	A
ld *operand*, %*gpr*	*gpr* <= *operand*;	A
st %*gpr*, operand	*operand* <= *gpr*;	A
csrrd %*csr*, %*gpr*	*gpr* <= *csr*	A
csrwr %*gpr*, %*csr*	*csr* <= *gpr*;	A

Oznaka *gprX* predstavlja oznaku nekog od programski dostupnih opštenamenskih registara ciljne arhitekture. Programski dostupni opštenamenski registri su `r0`, `r1`, `r2`, `r3`, `r4`, `r5`, `r6`, `r7`, `r8`, `r9`, `r10`, `r11`, `r12`, `r13`, `r14`/`sp` i `r15`/`pc`.

Oznaka *csrX* predstavlja oznaku nekog od programski dostupnih kontrolnih i statusnih registara ciljne arhitekture. Programski dostupni kontrolni i statusni registri su: `status`, `handler` i `cause`.

Oznaka *operand* obuhvata sve sintaksne notacije za navođenje operanada. Sintaksne notacije se razlikuju zavisno od toga da li se radi o asemblerskim naredbama za rad sa podacima ili asemblerskim naredbama skoka.

Asemblerske naredbe za rad sa podacima podržavaju različite sintaksne notacije za *operand*, opisane u nastavku, kojima se definiše vrednost podatka:

- `$<literal>` - vrednost `<literal>`
- `$<simbol>` - vrednost `<simbol>`
- `<literal>` - vrednost iz memorije na adresi `<literal>`
- `<simbol>` - vrednost iz memorije na adresi `<simbol>`
- `%<reg>` - vrednost u registru `<reg>`
- `[%<reg>]` - vrednost iz memorije na adresi `<reg>`
- `[%<reg> + <literal>]` - vrednost iz memorije na adresi `<reg> + <literal>`[1]
- `[%<reg> + <simbol>]` - vrednost iz memorije na adresi `<reg> + <simbol>`[2]

Asemblerske naredbe skoka i poziva potprograma podržavaju različite sintaksne notacije za *operand*, opisane u nastavku, kojima se definiše vrednost odredišne adrese skoka:

- `<literal>` - vrednost `<literal>`
- `<simbol>` - vrednost `<simbol>`

[1] Ukoliko vrednost literala nije moguće zapisati na širini od 12 bita kao označenu vrednost prijaviti grešku u procesu asembliranja.

[2] Ukoliko konačna vrednost simbola nije poznata u trenutku asembliranja ili konačnu vrednost simbola nije moguće zapisati na širini od 12 bita kao označenu vrednost prijaviti grešku u procesu asembliranja.

Zadatak 2: Linker

Uvod

Cilj ovog zadatka jeste realizacija linkera nezavisnog od ciljne arhitekture koji na osnovu metapodataka (tabela simbola, relokacioni zapisi itd.) vrši povezivanje jednog ili više predmetnih programa generisanih od strane asemblera iz prvog zadatka.

Ulaz linkera je izlaz asemblera pri čemu je moguće zadati veći broj predmetnih programa koje je potrebno povezati. Linker podrazumevano smešta sekcije, počevši od nulte adrese, jednu odmah iza druge onim redom kako su definisane unutar predmetnog programa. Veći broj predmetnih programa na svom ulazu linker obrađuje u redosledu njihovog navođenja preko komandne linije. Prilikom smeštanja sekcije istog imena kao prethodno smeštena sekcija dolazi do njenog umetanja počev od mesta gde se istoimena sekcija ranije završila, stvarajući time agregaciju istoimenih sekcija, pri čemu nema preklapanja sa sekcijama sledbenicama što se postiže njihovim guranjem ka višim adresama.

Izlaz linkera je tekstualna datoteka sa sadržajem u skladu sa opisom u nastavku (dozvoljeno je generisati kao izlaz, pored tekstualne datoteke, binarnu datoteku radi jednostavnijeg učitavanja izlaza linkera u emulator).

Podela zahteva po nivoima

Podela zahteva po nivoima za ovaj zadatak tiče se opcija komandne linije (ostale zahteve podrazumevano treba implementirati). Nivo u sklopu kojeg treba implementirati neku opciju komandne linije naveden je u uglastim zagradama ispred tražene opcije komandne linije.

Pokretanje iz terminala

Rezultat prevođenja *implementacije* ovog zadatka treba da ima `linker` za naziv. Sve informacije potrebne za izvršavanje zadaju se kao argumenti komandne linije. Jednim pokretanjem `linker` vrši povezivanje jedne ili više ulaznih datoteka. Nazivi ulaznih datoteka, koje predstavljaju predmetne programe, zadaju se kao samostalni argumenti komandne linije. Način pokretanja `linker` jeste sledeći:

```
linker [opcije] <naziv_ulazne_datoteke>...
```

Opcije komandne linije

Opis opcija komandne linije, zajedno sa opisom njihovih parametara, koje mogu biti zadate prilikom pokretanja `linker` u proizvoljnom redosledu nalazi se u nastavku:

[nivo A] `-o <naziv_izlazne_datoteke>`

Opcija komandne linije `-o` postavlja svoj parametar `<naziv_izlazne_datoteke>` za naziv izlazne datoteke koja predstavlja rezultat povezivanja.

[nivo A] `-place=<ime_sekcije>@<adresa>`

Opcija komandne linije `-place` eksplicitno definiše adresu počev od koje se smešta sekcija zadatog imena pri čemu su adresa i ime sekcije određeni `<ime_sekcije>@<adresa>` parametrom. Ovu opciju moguće je navoditi više puta za različita imena sekcija kako bi se definisala adresa za veći broj sekcija iz ulaznih datoteka. Sve sekcije za koje ova opcija nije

navedena smeštaju se na podrazumevani način, opisan u sklopu uvoda ovog zadatka, počev odmah iza kraja sekcije koja je smeštena na najvišu adresu.

[nivo A] -hex

Opcija komandne linije `-hex` predstavlja smernicu linkeru da kao rezultat povezivanja generiše zapis, na osnovu kojeg se može izvršiti inicijalizacija memorije, u vidu skupa parova (*adresa, sadržaj*). Sadržaj predstavlja mašinski kod koji treba da se nađe na zadatoj adresi. Parovi se generišu samo za one adrese na koje treba smestiti sadržaj sa definisanom početnom vrednošću. Format zapisa, na primeru u kojem je početna vrednost sadržaja jednaka njegovoj adresi, prikazan je u nastavku:

```
0000: 00 01 02 03 04 05 06 07
0008: 08 09 0A 0B 0C 0D 0E 0F
0010: 10 11 12 13 14 15 16 17
```

Povezivanje je moguće samo u slučaju da ne postoje (1) višestruke definicije simbola, (2) nerazrešeni simboli i (3) preklapanja između sekcija iz ulaznih predmetnih programa kada se uzmu u obzir `-place` opcije komandne linije. Ukoliko za zadate ulazne datoteke linkera nije ispunjen neki od prethodnih uslova linker mora da prijavi grešku uz odgovarajuću poruku. Nazivi simbola ili sekcija koji su uzrok greške treba da budu deo poruke o grešci.

Prilikom pokretanja linkera navođenje tačno jedne od `-relocatable` i `-hex` opcija komandne linije je obavezno. Linker ne treba da generiše nikakav izlaz ako nije navedena tačno jedna od dve prethodno navedene opcije komandne linije.

[nivo B] -relocatable

Opcija komandne linije `-relocatable` predstavlja smernicu linkeru da kao rezultat povezivanja generiše predmetni program, istog formata kao i izlaz asemblera, u kojem se sve sekcije smeštaju takođe od nulte adrese (potpuno se ignorišu potencijalno navedene `-place` opcije komandne linije). Predmetni program dobijen na ovakav način može kasnije biti naveden kao ulaz linkera.

Povezivanje je moguće samo u slučaju da nema višestrukih definicija simbola. Ukoliko za zadate ulazne datoteke linkera postoji višestruka definicija simbola linker mora da prijavi grešku uz odgovarajuću poruku. Naziv višestruko definisanog simbola treba da bude deo poruke o grešci.

Prilikom pokretanja linkera navođenje tačno jedne od `-relocatable` i `-hex` opcija komandne linije je obavezno. Linker ne treba da generiše nikakav izlaz ako nije navedena tačno jedna od dve prethodno navedene opcije komandne linije.

Primer pokretanja

Primer komande, kojom se pokreće povezivanje predmetnih programa *ulaz1.o* i *ulaz2.o* pri čemu se (1) definišu adrese na koje se smeštaju odgovarajuće sekcije i (2) zahteva generisanje zapisa za inicijalizaciju memorije, dat je u nastavku:

```
./linker -hex
    -place=data@0x4000F000 -place=text@0x40000000
    -o mem_content.hex
    ulaz1.o ulaz2.o
```

Zadatak 3: Emulator

Uvod

Cilj ovog zadatka jeste realizacija interpretativnog emulatora za računarski sistem opisan u prilogu. Ulaz emulatora jeste datoteka za inicijalizaciju memorije dobijena kao izlaz linkera uz navedenu `-hex` opciju komandne linije. Emulacija je moguća samo ukoliko je ulaznu datoteku moguće uspešno učitati u memorijski adresni prostor emuliranog računarskog sistema. Nakon pokretanja emulatora jedini ispis u konzoli jeste ispis direktno iz *korisničkog programa*, dok *implementacija* ne ispisuje ništa samostalno do završetka emulacije. Emulacija se završava u onom trenutku kada emulirani procesor izvrši `halt` instrukciju *korisničkog programa*. Nakon završetka emulacije *implementacija* ispisuje stanje emuliranog procesora u sledećem formatu:

```
-----------------------------------------------------------------
Emulated processor executed halt instruction
Emulated processor state:
 r0=0x00000000    r1=0x00000000    r2=0x00000000    r3=0x00000000
 r4=0x00000000    r5=0x00000000    r6=0x00000000    r7=0x00000000
 r8=0x00000000    r9=0x00000000   r10=0x00000000   r11=0x00000000
r12=0x00000000   r13=0x00000000   r14=0x00000000   r15=0x00000000
```

Podela zahteva po nivoima

Podela zahteva po nivoima za ovaj zadatak definisana je u nastavku. Za nivo A potrebno je emulirati celokupan posmatrani računarski sistem osim periferija terminal i tajmer. Za nivo B, pored svega definisanog nivoom A, potrebno je emulirati i periferiju terminal. Za nivo C, pored svega definisanog nivoom B, potrebno je emulirati i periferiju tajmer.

Pokretanje iz terminala

Rezultat prevođenja *implementacije* ovog zadatka treba da ima `emulator` za naziv. Sve informacije potrebne za izvršavanje zadaju se kao argumenti komandne linije. Jednim pokretanjem `emulator` emulira jedno izvršavanje programa iz ulazne datoteke. Naziv ulazne datoteke, koja predstavlja izlaz linkera uz navedenu `-hex` opciju komandne linije, zadaje se kao samostalni argument komandne linije. Način pokretanja `emulator` jeste sledeći:

```
emulator <naziv_ulazne_datoteke>
```

Primer pokretanja

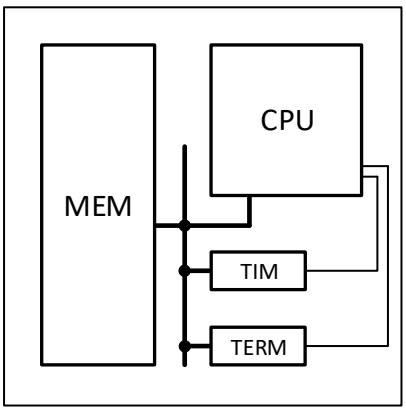
Primer komande, kojom se pokreće emulacija na osnovu zapisa za inicijalizaciju memorije u ulaznoj datoteci *mem_content.hex*, dat je u nastavku:

```
./emulator mem_content.hex
```

Prilog: Opis računarskog sistema

Uvod

Apstraktni računarski sistem se sastoji od procesora, operativne memorije, tajmera i terminala. Sve komponente računarskog sistema su međusobno povezane preko sistemske magistrale. Tajmer i terminal su povezani pored sistemske magistrale i direktno sa procesorom preko linija za slanje zahteva za prekid. Uprošćen šematski prikaz posmatranog apstraktnog računarskog sistema prikazan je na sledećoj slici:



Opis procesora

U nastavku je opisan deo 32-bitnog dvoadresnog procesora sa Von-Neuman arhitekturom. Adresibilna jedinica je jedan bajt, a raspored bajtova u reči je little-endian. Veličina memorijskog adresnog prostora je 2^{32}B. Nakon inicijalnog odnosno hladnog (engl. cold) restarta isto kao i nakon toplog (engl. warm) restarta procesor počinje da izvršava instrukcije počev od adrese `0x40000000`.

Procesorski registri

Procesor poseduje šesnaest opštenamenskih 32-bitnih registara označenih sa `r<num>` gde `<num>` može imati vrednosti od nula do petnaest. Registar `r0` je ožičen na vrednost nula. Registar `r15` se koristi kao `pc` registar. Vrednost registra `r15` sadrži adresu instrukcije koja naredna treba da se izvrši. Registar `r14` se koristi kao `sp` registar. Vrednost registra `r14` sadrži adresu zauzete lokacije na vrhu steka (stek raste ka nižim adresama).

Pored pomenutih opštenamenskih registara postoje sledeći statusni i kontrolni 32-bitni registri: `status` (statusna reč procesora), `handler` (adresa prekidne rutine) i `cause` (uzrok prekida). Statusna reč procesora odnosno `status` registar sastoji se od flegova koji pružaju mogućnost konfiguracije mehanizma prekida. Izgled nižih 16 bita statusne reči procesora jeste:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
													I	Tl	Tr

Značenje flegova u `status` registru:

- `Tr` (Timer) - maskiranje prekida od tajmera (0 - omogućen, 1 - maskiran),
- `Tl` (Terminal) - maskiranje prekida od terminala (0 - omogućen, 1 - maskiran) i
- `I` (Interrupt) - globalno maskiranje spoljašnjih prekida (0 - omogućeni, 1 - maskirani).

Memorijski mapirani registri

Memorijski mapirani registri jesu registri kojima se pristupa instrukcijama za pristup memorijskom adresnom prostoru. Počev od adrese `0xFFFFFF00` memorijskog adresnog prostora nalazi se prostor veličine 256 bajtova rezervisan za memorijski mapirane registre. Memorijski mapirani registri koriste se za rad sa periferijama u računarskom sistemu.

Mehanizam prekida

Sistem poseduje samo jednu prekidnu rutinu čija je adresa definisana vrednošću `handler` registra. Uzrok ulaska u datu prekidnu rutinu određen je vrednošću `cause` registra. Moguće vrednosti `cause` registra usled različitih uzroka ulaska u prekidnu rutinu su:

- vrednost 1 u slučaju izvršavanja nekorektne instrukcije (nepostojeći operacioni kod, neispravan način adresiranja itd.),
- vrednost 2 usled pristiglog zahteva za prekid od tajmera (opis principa rada tajmera i način njegove konfiguracije dat je u zasebnom poglavlju),
- vrednost 3 usled pristiglog zahteva za prekid od terminala (opis principa rada terminala dat je u zasebnom poglavlju) i
- vrednost 4 ukoliko je reč o softverskom prekidu.

Svaka mašinska instrukcija procesora izvršava se atomično. Zahtevi za prekid se opslužuju tek nakon što se trenutna mašinska instrukcija atomično izvrši do kraja. Procesor prilikom prihvatanja zahteva za prekid i ulaska u prekidnu rutinu postavlja na stek statusnu reč i povratnu adresu upravo tim redom i zatim globalno maskira prekide.

Format procesorskih instrukcija

Svaka instrukcija je veličine četiri bajta. Format instrukcije dat je u nastavku:

I		II		III		IV	
OC	MOD	RegA	RegB	RegC	Disp	Disp	Disp

Polje `OC` označava operacioni kod. Vrednost `OC[3:0]` definiše o kojoj mašinskoj instrukciji procesora je reč.

Polje `MOD` označava modifikator instrukcije. Vrednost `MOD[3:0]` govori šta tačno data instrukcija treba da uradi.

Polje `RegX` definiše jedan korišćeni registar instrukcije. Vrednost `RegX[3:0]` specificira korišćeni registar navodeći njegov indeks. Opštenamenskim registrima dodeljeni su indeksi koji odgovaraju njihovim imenima. Indeks registra `r0` je nula, indeks registra `r1` je jedan itd. Kontrolnim i statusnim registrima dodeljeni su indeksi navedeni u nastavku. Indeks registra `status` je nula, indeks registra `handler` je jedan i indeks registra `cause` je dva.

Polje `Disp` predstavlja označeni (engl. signed) pomeraj. Vrednost `Disp[11:0]` instrukcije koriste u izračunavanjima u skladu sa operacijom koju obavljaju.

Pregled procesorskih instrukcija

Instrukcija za zaustavljanje procesora

I		II		III		IV	
7654	3210	7654	3210	7654	3210	7654	3210
0000	0000	0000	0000	0000	0000	0000	0000

Zaustavlja procesor kao i dalje izvršavanje narednih instrukcija.

Instrukcija softverskog prekida

I		II		III		IV	
7654	3210	7654	3210	7654	3210	7654	3210
0001	0000	0000	0000	0000	0000	0000	0000

Generiše softverski zahtev za prekid.

```
push status; push pc; cause<=4; status<=status&(~0x1); pc<=handle;
```

Instrukcija poziva potprograma

I		II		III		IV	
7654	3210	7654	3210	7654	3210	7654	3210
0010	MMMM	AAAA	BBBB	0000	DDDD	DDDD	DDDD

Poziva potprogram pre čega sačuva povratnu adresu na steku. Zavisno od modifikatora instrukcija skače na sledeću adresu:

```
MMMM==0b0000: push pc; pc<=gpr[A]+gpr[B]+D;
MMMM==0b0001: push pc; pc<=mem32[gpr[A]+gpr[B]+D];
```

Instrukcija skoka

I		II		III		IV	
7654	3210	7654	3210	7654	3210	7654	3210
0011	MMMM	AAAA	BBBB	CCCC	DDDD	DDDD	DDDD

Skače bezuslovno ili uslovno, u skladu sa modifikatorom instrukcije, koristeći zadati pomeraj. Zavisno od modifikatora instrukcija vrši sledeći tip skoka:

```
MMMM==0b0000: pc<=gpr[A]+D;
MMMM==0b0001: if (gpr[B] == gpr[C]) pc<=gpr[A]+D;
MMMM==0b0010: if (gpr[B] != gpr[C]) pc<=gpr[A]+D;
MMMM==0b0011: if (gpr[B] signed> gpr[C]) pc<=gpr[A]+D;
MMMM==0b1000: pc<=mem32[gpr[A]+D];
MMMM==0b1001: if (gpr[B] == gpr[C]) pc<=mem32[gpr[A]+D];
MMMM==0b1010: if (gpr[B] != gpr[C]) pc<=mem32[gpr[A]+D];
MMMM==0b1011: if (gpr[B] signed> gpr[C]) pc<=mem32[gpr[A]+D];
```

Instrukcija atomične zamene vrednosti

I		II		III		IV	
7654	3210	7654	3210	7654	3210	7654	3210
0100	0000	0000	BBBB	CCCC	0000	0000	0000

Zamenjuje vrednost dva registra atomično bez mogućnosti da zamena bude prekinuta usled asinhronog zahteva za prekid.

```
temp<=gpr[B]; gpr[B]<=gpr[C]; gpr[C]<=temp;
```

Instrukcija aritmetičkih operacija

I		II		III		IV	
7654	3210	7654	3210	7654	3210	7654	3210
0101	MMMM	AAAA	BBBB	CCCC	0000	0000	0000

Vrši odgovarajuću aritmetičku operaciju, u skladu sa modifikatorom instrukcije, nad vrednostima u zadatim registrima. Zavisno od modifikatora instrukcija obavlja operaciju:

```
MMMM==0b0000: gpr[A]<=gpr[B] + gpr[C];
MMMM==0b0001: gpr[A]<=gpr[B] - gpr[C];
MMMM==0b0010: gpr[A]<=gpr[B] * gpr[C];
MMMM==0b0011: gpr[A]<=gpr[B] / gpr[C];
```

Instrukcija logičkih operacija

I		II		III		IV	
7654	3210	7654	3210	7654	3210	7654	3210
0110	MMMM	AAAA	BBBB	CCCC	0000	0000	0000

Vrši odgovarajuću logičku operaciju, u skladu sa modifikatorom instrukcije, nad vrednostima u zadatim registrima. Zavisno od modifikatora instrukcija obavlja operaciju:

```
MMMM==0b0000: gpr[A]<=~gpr[B];
MMMM==0b0001: gpr[A]<=gpr[B] & gpr[C];
MMMM==0b0010: gpr[A]<=gpr[B] | gpr[C];
MMMM==0b0011: gpr[A]<=gpr[B] ^ gpr[C];
```

Instrukcija pomeračkih operacija

I		II		III		IV	
7654	3210	7654	3210	7654	3210	7654	3210
0111	MMMM	AAAA	BBBB	CCCC	0000	0000	0000

Vrši odgovarajuću pomeračku operaciju, u skladu sa modifikatorom instrukcije, nad vrednostima u zadatim registrima. Zavisno od modifikatora instrukcija obavlja operaciju:

```
MMMM==0b0000: gpr[A]<=gpr[B] << gpr[C];
MMMM==0b0001: gpr[A]<=gpr[B] >> gpr[C];
```

Instrukcija smeštanja podatka

I		II		III		IV	
7654	3210	7654	3210	7654	3210	7654	3210
1000	MMMM	AAAA	BBBB	CCCC	DDDD	DDDD	DDDD

Smešta podatak u memoriju. Zavisno od modifikatora instrukcija obavlja operaciju:

```
MMMM==0b0000: mem32[gpr[A]+gpr[B]+D]<=gpr[C];
MMMM==0b0010: mem32[mem32[gpr[A]+gpr[B]+D]]<=gpr[C];
MMMM==0b0001: gpr[A]<=gpr[A]+D; mem32[gpr[A]]<=gpr[C];
```

Instrukcija učitavanja podatka

I		II		III		IV	
7654	3210	7654	3210	7654	3210	7654	3210
1001	MMMM	AAAA	BBBB	CCCC	DDDD	DDDD	DDDD

Učitava podatak u registar. Zavisno od modifikatora instrukcija obavlja operaciju:

```
MMMM==0b0000: gpr[A]<=csr[B];
MMMM==0b0001: gpr[A]<=gpr[B]+D;
MMMM==0b0010: gpr[A]<=mem32[gpr[B]+gpr[C]+D];
MMMM==0b0011: gpr[A]<=mem32[gpr[B]]; gpr[B]<=gpr[B]+D;
MMMM==0b0100: csr[A]<=gpr[B];
MMMM==0b0101: csr[A]<=csr[B]|D;
MMMM==0b0110: csr[A]<=mem32[gpr[B]+gpr[C]+D];
MMMM==0b0111: csr[A]<=mem32[gpr[B]]; gpr[B]<=gpr[B]+D;
```

Sve kombinacije instrukcija i operanada, za koje ne postoji razumno tumačenje, proglasiti greškom.

Opis terminala

Terminal predstavlja ulazno/izlaznu periferiju koja se sastoji od displeja (izlaz) i tastature (ulaz). Terminal poseduje dva programski dostupna registra `term_out` i `term_in` kojima se pristupa kroz memorijski adresni prostor (memorijski mapirani registri). Navedeni programski dostupni registri mapirani su u memorijski adresni prostor na sledeći način:

Registar	Opseg adresa
`term_out`	`[0xFFFFFF00-0xFFFFFF03]`
`term_in`	`[0xFFFFFF04-0xFFFFFF07]`

Registar `term_out` predstavlja registar izlaznih podataka. Terminal prati upise u ovaj registar i prilikom svakog upisa vrednosti u `term_out` registar ispisuje znak na displej. Ispisani znak određen je sadržajem ASCII tabele za vrednost koja je upisana u `term_out` registar.

Registar `term_in` predstavlja registar ulaznih podataka. Terminal vrši sledeće dve operacije svaki put kada se pritisne bilo koje dugme na tastaturi: (1) upisuje ASCII kod pritisnutog tastera u `term_in` registar kako bi čitanjem njegove vrednosti bilo moguće ustanoviti koje dugme je pritisnuto i (2) generiše zahtev za prekid. Dve prethodno opisane operacije terminal vrši čim se pritisne bilo koje dugme; pri čemu treba naglasiti da terminal nipošto ne čeka terminator unosa odnosno *enter* dugme. Terminal ne vrši *echo* pritisnutog dugmeta odnosno ne prikazuje ga na displeju. Za potrebe implementacije ove funkcionalnosti emulatora moguće je, na primer, koristiti `<termios.h>` zaglavlje. Ukoliko *korisnički program* ne pročita vrednost `term_in` registra na vreme, odnosno pre nego što korisnik pritisne naredno dugme, ASCII kod prethodno pritisnutog dugmeta biće bespovratno izgubljen. Ne treba vršiti baferisanje ASCII kodova u okviru *implementacije*. Smatrati da je procesor posmatranog računarskog sistema dovoljno brz da garantovano, ukoliko je *korisnički program* ispravno napisan, može da stigne da pročita vrednost `term_in` registra u prekidnoj rutini pre nego što ona bude pregažena.

Opis tajmera

Tajmer predstavlja periferiju koja periodično generiše zahtev za prekid. Tajmer poseduje jedan programski dostupan registar `tim_cfg` kojem se pristupa kroz memorijski adresni prostor (memorijski mapiran registar). Navedeni programski dostupan registar mapiran je u memorijski adresni prostor na sledeći način:

Registar	Opseg adresa
`tim_cfg`	`[0xFFFFFF10-0xFFFFFF13]`

Registar `tim_cfg` predstavlja konfiguracioni registar tajmera. Perioda generisanja zahteva za prekid od strane tajmera definisana je vrednošću `tim_cfg` registra na sledeći način: `0x0` -> 500ms, `0x1` -> 1000ms, `0x2` -> 1500ms, `0x3` -> 2000ms, `0x4` -> 5000ms, `0x5` -> 10s, `0x6` -> 30s i `0x7` -> 60s. Inicijalna vrednost `tim_cfg` registra, nakon pokretanja odnosno resetovanja emuliranog računarskog sistema, jeste `0x00000000`.

Primer korisničkog programa

Program prikazan u nastavku predstavlja primer programa napisanog na asembleru opisanom u prvom zadatku. Izvorni asemblerski kod programa je razdvojen na dve tekstualne datoteke *handler.s* i *main.s*. Program obavlja sledeće radnje:

- postavlja adresu prekidne rutine,
- usled prekida od terminala ispisuje se na displeju znak koji odgovara pritisnutom dugmetu,
- usled prekida od tajmera ispisuje se na displeju znak 'T',
- konfiguriše tajmer tako da generiše zahtev za prekid na svaku sekundu i
- čeka da korisnik pritisne pet puta dugme pre nego što zaustavi procesor.

```
# file: handler.s
.equ term_out, 0xFFFFFF00
.equ term_in, 0xFFFFFF04
.equ ascii_code, 84 # ascii('T')
.extern my_counter
.global handler
.section my_code_handler
handler:
   push %r1
   push %r2
   csrrd %cause, %r1
   ld $2, %r2
   beq %r1, %r2, my_isr_timer
   ld $3, %r2
   beq %r1, %r2, my_isr_terminal
# obrada prekida od tajmera
my_isr_timer:
   ld $ascii_code , %r1
   st %r1, term_out
   jmp finish
# obrada prekida od terminala
my_isr_terminal:
   ld term_in, %r1
   st %r1, term_out
   ld my_counter, %r1
   ld $1, %r2
   add %r2, %r1
   st %r1, my_counter
finish:
   pop %r2
   pop %r1
   iret
.end
```

```
# file: main.s
.equ tim_cfg, 0xFFFFFF10
.equ init_sp, 0xFFFFFF00
.extern handler
.section my_code_main
   ld $init_sp, %sp
   ld $handler, %r1
   csrwr %r1, %handler
   ld $0x1, %r1
   st %r1, tim_cfg
wait:
   ld my_counter, %r1
   ld $5, %r2
   bne %r1, %r2, wait
   halt
.global my_counter
.section my_data
my_counter:
   .word 0
.end
```

Komande za prevođenje, povezivanje i emuliranje ovog primera jesu:

```
./asembler -o handler.o handler.s
./asembler -o main.o main.s
./linker -hex
       -place=my_code_main@0x40000000
       -place=my_code_handler@0xC0000000
       -o program.hex handler.o main.o
./emulator program.hex
```

Predaja projekta

Potrebno je realizovati prethodno opisane alate prema datim zahtevima pod Linux operativnim sistemom na amd64 arhitekturi koristeći C/C++ programski jezik. Potrebni alati su opisani u okviru materijala za predavanja i vežbe. Preporuka je rešenje izrađivati u okviru VMware virtuelne mašine koju je moguće preuzeti sa eLearning platforme. Virtuelna mašina sadrži već instalirane sve neophodne alate a kao integrisano okruženje za razvoj programa moguće je koristiti VS Code koje treba povezati sa GNU toolchain setom.

Podešavanje okruženja potrebnog za uspešno prevođenje izvornog koda i pokretanje programa takođe spada u deo postavke projektnog zadatka. Odbrana rešenja projektnog zadatka vrši se isključivo pod prethodno opisanim okruženjem u okviru virtuelne mašine. Pristup internetu prilikom odbrane rešenja projektnog zadatka biće onemogućen što znači da treba koristiti samo alate dostupne na virtuelnoj mašini.

Pravila za predaju projekta

Projekat se predaje isključivo kao jedna .zip arhiva unutar koje se nalazi samo jedan direktorijum imena *resenje* čiji je sadržaj opisan u nastavku. Unutar direktorijuma *resenje* može se naći (1) opciono *makefile* ili neka druga skripta za prevođenje rešenja zadatka, (2) opciono poddirektorijum *misc* predviđen za dodatne stvari kao što su flex i bison ulazne datoteke (izlazne datoteke generisane ovim alatima ne treba nikako predavati jer će one biti generisane) i (3) obavezno sledeća tri poddirektorijuma: *tests*, *src* i *inc*. Sadržaj ovih poddirektorijuma treba da bude sledeći:

- *tests*: ulazne datoteke za prikaz funkcionalnosti rešenja zadatka,
- *src*: sve .c i .cpp datoteke sa izvornim kodom i
- *inc*: sve .h i .hpp datoteke sa izvornim kodom.

Opisani sadržaj .zip arhive ujedno treba da bude i jedini njen sadržaj. Nije dozvoljeno predavati izvršne datoteke, datoteke generisane od strane alata flex i bison niti bilo šta drugo što iznad nije eksplicitno navedeno. Nepoštovanje pravila za predaju projekta po pitanju sadržaja, strukture i naziva direktorijuma, povlači negativne poene ili zabranu izlaska na odbranu u datom ispitnom roku.

Prikaz primitivnog primera (rešenje svakako treba da sadrži mnogo veći broj datoteka sa izvornim kodom u odnosu na prikazano) sadržaja .zip arhive nalazi se u nastavku:

```
.
└── resenje
    ├── makefile
    ├── misc
    │   ├── flex
    │   └── bison
    ├── inc
    │   ├── assembler.hpp
    │   ├── linker.hpp
    │   └── emulator.hpp
    ├── src
    │   ├── assembler.cpp
    │   ├── linker.cpp
    │   └── emulator.cpp
    └── tests
        ├── test1.s
        ├── test2.s
        └── test3.s
```

Zapisnik revizija

Ovaj zapisnik sadrži spisak izmena i dopuna ovog dokumenta po verzijama.

Verzija 1.1

Strana	Izmena
8	Dodate dve fusnote vezane za vrednost podatka operanda.

Verzija 1.2

Strana	Izmena
14	Dodati modifikatori za instrukciju skoka i poziva potprograma.

Verzija 1.3

Strana	Izmena
16	Dodati modifikatori za instrukciju smeštanja podatka.

Verzija 1.4

Strana	Izmena