Dvoprolazni arm-like asembler, linker I emulator

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*.

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.

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

```
.global <lista_simbola>
```

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

```
.extern <lista_simbola>
```

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

```
.section <ime_sekcije>
```

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

```
.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.

`.skip <literal>`

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

`.end`

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

Asemblerske naredbe

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

shl %*gprS*, %*gprD*	*gprD* <= *gprD* << *gprS*;
shr %*gprS*, %*gprD*	*gprD* <= *gprD* >> *gprS*;
ld *operand*, %*gpr*	*gpr* <= *operand*;
st %*gpr*, operand	*operand* <= *gpr*;
csrrd %*csr*, %*gpr*	*gpr* <= *csr*
csrwr %*gpr*, %*csr*	*csr* <= *gpr*;

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>

[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.

- <simbol> - vrednsost <simbol>

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).

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:

`-o <naziv_izlazne_datoteke>`

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

`-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.

-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.

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
```

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

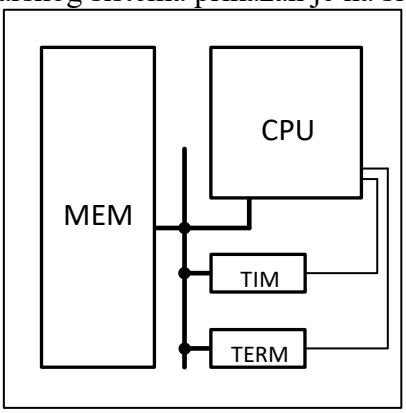
Primer komande, kojom se pokreće emulacija na osnovu zapisa za inicijalizaciju memorije u ulaznoj datoteci *mem_content.hex*, dat je u nastavku:

```
./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

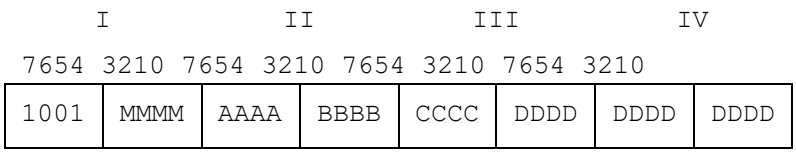
```
     I           II          III          IV
 7654 3210 7654 3210 7654 3210 7654 3210
```

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



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.