PROJEKAT

II termin

Dr Nevena Radović

Format programa QtSpim simulatora

- QtSpim simulator će biti korišćen za pisanje programa.
- QtSpim zahtijeva pravilno formatiran asemblerski izvorni (source) fajl.
- Pravilno formatiran asemblerski source fajl se sastoji od dva dijela:
  - Sekcija za zapis podataka (**data section**)
  - Sekcija za zapis koda (**text section**)
- Prilikom učitavanja programa, QtSpim vrši asembiliranje programa. Tada će biti prepoznate sve greške koje se tiču formata programa ili instrukcija. Te greške se moraju riješiti prije kako bi se program mogao uspješno izvršiti.

Asemblerske direktive i komentari

- **Asemblerska direktiva** je poruka asembleru, odnosno QtSpim simulatoru, kojom se govori asembleru nešto što je potrebno da zna, kako bi mogao obaviti asembliranje.
- Asemblerske direktive počinju sa ".„
- Neophodne su za definisanje početka i kraja deklarisanja podataka, kao i za definisanje početka i kraja procedurea/funkcija.
- Prilikom pisanja programa moguće je koristiti **komentare**.
- Karakter "**#**" označava programsku liniju sa komentarima. Sve što se zapiše nakon # se smatra komentarom (do kraja te programske linije).
- Prazne linije koda su prihvatljive.

Deklaracija podataka

- Podaci moraju biti deklarisani u sekciji za zapis podataka (data section), direktiva: **".data"**
- Sve varijable i konstante se smještaju u ovu sekciju.
- Ime varijable mora početi sa slovom, a u nastavku imena se mogu koristiti slova i brojevi (i specijalni karakter "_").
- Odmah nakon imena varijable se zapisuju **":"** (dvije tačke).
- Definicija varijable uključuje: **ime, tip podatka i početnu vrijednost varijable**.
- Ispred tipa podatka je neophodna **"."**
- Opšta forma deklaracije podatka je:

<Ime_varijable>: .<Tip_podatka> <Početna_vrijednost>

Deklaracija podataka

Deklaracija	
.byte	8-bitna varijabla
.half	16-bitna varijabla
.word	32-bitna varijabla
.ascii	ASCII string
.asciiz	ASCII string terminisan nulom
.float	32-bitni IEEE 754 floating point broj
.double	64-bitni IEEE 754 floating point broj
.space <n>	<n> bajta neinicijalizovane memorije

Podržani tipovi podataka

Deklaracija podataka - primjeri

- **Integeri** se definišu direktivama: **.word, .half, ili .byte**.
- Ne zaboravite – za zapis negativnih vrijednosti se koristi dvojni komplement.
- **Primjer:** Deklaracija integer varijabli "word_a" i "word_b" kao 32-bitnih word vrijednosti inicijalizovanih na 500 i -10.

```
word_a:     .word  500
word_b:     .word  -10
```

- **Primjer:** Deklaracija integer varijabli "hw_a" i "hw_b" kao 16-bitnih word vrijednosti inicijalizovanih na 50 i -3

```
hw_a:       .half 50
hw_b:       .half -3
```

Deklaracija podataka - primjeri

- **Primjer:** Deklaracija integer varijabli „byte_a" i „byte_b" kao 8-bitnih vrijednosti inicijalizovanih na 5 i -3.

```
byte_a:     .byte   5
byte_b:     .byte   -3
```

- Ako je varijabla inicijalizovana sa vrijednošću koja ne može biti smještene u alocirani prostor asembler će generisati grešku.
- Na primjer, pokušaj da se u bajt varijablu smjesti vrijednost 500 će generisati grešku.

Deklaracija podataka - primjeri

- **Stringovi** se definišu direktivama: **.ascii** ili **.asciiz**
- Novi red, **"\n"**, i tabulacija, **"\t"** su podržani unutar stringova.
- **Primjer:** Deklaracija stringa „Hello World“:

  message: .asciiz "Hello World\n"
- U ovom primjeru string je terminisan nulom (nakon novog reda).
- Terminacioni karakter NULL se ne štampa prilikom ispisivanja stringa, već se koristi da označi kraj stringa.

Deklaracija podataka - primjeri

- Ako definišemo string koji se sastoji od više linija, terminator NULL je potreban samo u posljednjoj liniji:
- **Primjer:**

```
message:    .ascii "Line 1: Goodbye World\n"
            .ascii "Line 2: So, long and thanks "
            .ascii "for everything.\n"
            .asciiz "Line 3: Game Over.\n"
```

- Floating-point vrijednosti se definišu dikretivama **.float** (32-bit) ili **.double** (64-bit).
- **Primjer:** Deklaracija floating point varijable π:

```
pi:     .float 3.14159
```

Programski kod

- Direktiva **".text"** se navodi na početku sekcije za zapis programskog koda (text section).
- Inicijalna/glavna procedura se označava requirements for naming sa **„main" .**
- Direktive **".globl <name>"** i **".ent <name>"** se koriste da definišu ime inicijalne procedure.
- Direktiva **".ent"** je opcionalna u QtSpim simulator (nije obavezna).
- Inicijalna/glavna procedura mora početi sa labelom koja nosi ime procedure.
- Inicijalna/glavna procedura (i sve ostale procedure) se završavaju direktivom **".end <name>"**.

Labele

- Labele su lokacije u kodu, i obično se koriste kao imena funkcija/procedura ili kao lokacija na koju se vrši skok.
- Prva upotreba labele je lokacija početka glavnog programa, koji se mora nazvati **„main"** (zahtjev QtSpim simulatora).
- Pravila za definisanje labela:
  - Naziv labele mora početi sa slovom
  - Ostatak imena može sadržati slova, brojeve ili podcrtu "_" (underscore)
  - Naziv labele se mora završiti sa ":" (dvije tačke).
  - Ista labela se može definisati samo jednom.
- QtSpim je ***case-sensitive***. Dakle, **Petlja:** i **petlja:** su različite labele.

Template za pisanje programa

```
# Sekcija za zapis podataka (data section)
.data
# deklaracija željenih podataka
# ...

# Sekcija za zapis koda (text section)
.text
.globl main
.ent main          # opciono, nije obavezno u QtSpim-u

main:
# zapis programskog koda
# ...
# Kraj programskog koda
        li $v0, 10
        syscall
.end main
```

Instrukcije

- U asembleru su instrukcije jednostavne i izvršavaju jednu operaciju. U višim programskim jezicima, s druge strane, jedna linija koda može biti prevedena u čitav niz instrukcija zapisanih u asembleru.
- Kao dio MIPS arhitekture, asemblerski jezik uključuje **pseudoinstrukcije**.
- Same instrukcije (bare instructions) se izvršavaju direktno od strane CPU.
- Pseudoinstrukciju će asembler, odnosno simulator, prepoznati i konvertovati u jednu ili više instrukcija.

Način zapisa instrukcija

- Instrukcija se sastoji od imena instrukcije i operanada. Operand određuje odakle dolazi podatak (nad kojim se vrši operacija), odnosno gdje će rezultat operacije biti smješten.

Oznaka	Opis
Rdest	Operand koji određuje destinaciju. Mora biti integer registar. Svaki novi upis će prebrisati prethodni sadržaj.
Rsrc	Operand koji određuje izvor podatka. Mora biti integer register. Vrijednost se ne mijenja nakon izvršenja instrukcije.
Src	Operand koji određuje izvor podatka. Mora biti integer register ili immediate vrijednost. Vrijednost se ne mijenja nakon izvršenja instrukcije.
FRdest	Operand koji određuje destinaciju. Mora biti floating point registar. Svaki novi upis će prebrisati prethodni sadržaj.
FRsrc	Operand koji određuje izvor podatka. Mora biti floating point register. Vrijednost se ne mijenja nakon izvršenja instrukcije.
Imm	Immediate vrijednost
Mem	Memorijska lokacija. Može biti ime promjenljive ili indirektna referenca (npr. memorijska adresa).

Čitanje i smještanje podataka

- CPU izračunavanja se u opštem slučaju obavljaju nad registrima.
- Stoga, prije nego što izračunavanje može da bude izvršeno, podaci se smještaju u registre (npr. iz memorije), a nakon što je izračunavanje izvršeno podaci se iz registara smještaju u druge varijable ili memoriju.
- **Load** i **store** instrukcije vrše transfer podataka između memorije i registara, i obrnuto.
- **Move** instrukcije vrše transfer podataka između registara.

Čitanje i smještanje podataka

Load i **store** instrukcije

Instrukcija	Opis
l <type> Rdest, mem	Učitaj vrijednost sa memorijske lokacije u destinacioni registar.
li Rdest, imm	Učitaj immediate vrijednost u destinacioni registar.
la Rdest, mem	Učitaj adresu memorijske lokacije u destinacioni registar.
s <type> Rsrc, mem	Smjesti sadržaj source registra na memorijsku lokaciju.

- **Primjer:** Neka su deklarisani sljedeći podaci:

```
num:    .word   0
wnum:   .word   42
hnum:   .half   73
bnum:   .byte   7
wans:   .word   0
hans:   .half   0
bans:   .byte   0
```

- Zapišimo u asembleru sljedeće operacije:

```
num = 27
wans = wnum
hans = hnum
bans = bnum
```

- **Rješenje:**

```
li $t0, 27
sw $t0, num       # num = 27
lw $t0, wnum
sw $t0, wans      # wans = wnum
lh $t1, hnum
sh $t1, hans      # hans = hnum
lb $t2, bnum
sb $t2, bans      # bans = bnum
```

Čitanje i smještanje podataka

Move instrukcije

Instrukcije	Opis
move Rdest, RSrc	Kopiraj sadržaj integer source registra u integer odredišni registar
mfhi Rdest	Kopiraj sadržaj iz $hi registra u Rdest
mflo Rdest	Kopiraj sadržaj iz $lo registra u Rdest
mthi Rdest	Kopiraj sadržaj u $hi registar iz Rdest
mtlo Rdest	Kopiraj sadržaj u $lo registar iz Rdest

Primjer: Instrukcije:

```
li      $t0, 42
move    $t1, $t0
```

će pomjeriti sadržaj registra $t0 (koji iznosi 42) u registar $t1.

Aritmetičke operacije

Instrukcija		Opis
add	**Rdest, Rsrc, Src**	Signed sabiranje, Rdest = Rsrc + Src ili Imm
addu	**Rdest, Rsrc, Src**	Unsigned sabiranje, Rdest = Rsrc + Src ili Imm
sub	**Rdest, Rsrc, Src**	Signed oduzimanje, Rdest = Rsrc – Src ili Imm
subu	**Rdest, Rsrc, Src**	Unsigned oduzimanje, Rdest = Rsrc – Src ili Imm
mul	**Rdest, Rsrc, Src**	Signed množenje bez prekoračenja, Rdest = Rsrc * Src ili Imm
mulo	**Rdest, Rsrc, Src**	Signed množenje sa prekoračenjem, Rdest = Rsrc * Src ili Imm
mulou	**Rdest, Rsrc, Src**	Unsigned množenje sa prekoračenjem, Rdest = Rsrc * Src ili Imm
mult	**Rsrc1, Rsrc2**	Signed 64-bit množenje, $hi/$lo = Rsrc1 * Rsrc2
multu	**Rsrc1, Rsrc2**	Unsigned 64-bit množenje, $hi/$lo = Rsrc1 * Rsrc2

Aritmetičke operacije

Instrukcija	Opis
div Rdest, Rsrc, Src	Signed dijeljenje, Rdest = Rsrc / Src ili Imm
divu Rdest, Rsrc, Src	Unsigned dijeljenje, Rdest = Rsrc / Src ili Imm
div Rsrc1, RSrc2	Signed dijeljenje sa ostatkom, $lo = Rsrc1 / RSrc2 $hi = Rsrc1 % RSrc2
divu Rsrc1, RSrc2	Unsigned dijeljenje sa ostatkom, $lo = Rsrc1 / RSrc2 $hi = Rsrc1 % RSrc2
rem Rdest, Rsrc, Src	Signed ostatak, Rdest = Rsrc % Src ili Imm
remu Rdest, Rsrc, Src	Unsigned ostatak, Rdest = Rsrc % Src ili Imm
abs Rdest, Rsrc	Apsolutna vrijednost, Rdest = | Rsrc |
neg Rdest, Rsrc	Negativna vrijednost, Rdest = - Rsrc

***Ove instrukcije rade sa 32-bitnim registrima

- **Primjer:** Neka su deklarisani sljedeći podaci:

```
wnum1:      .word   651
wnum2:      .word   42
wans1:      .word   0
wans2:      .word   0
wans3:      .word   0
hnum1:      .half   73
hnum2:      .half   15
hans:       .half   0
bnum1:      .byte   7
bnum2:      .byte   9
bans:       .byte   0
```

Zapišimo u asembleru sljedeće operacije:

```
wans1 = wnum1 + wnum2
wans2 = wnum1 * wnum2
wans3 = wnum1 % wnum2
hans  = hnum1 * hnum2
bans  = bnum1 / bnum2
```

- **Rješenje:**

```
lw $t0, wnum1
lw $t1, wnum2
add $t2, $t0, $t1
sw $t2, wans1     # wans1 = wnum1 + wnum2
lw $t0, wnum1
lw $t1, wnum2
mul $t2, $t0, $t1
sw $t2, wans2     # wans2 = wnum1 * wnum2
lw $t0, wnum1
lw $t1, wnum2
rem $t2, $t0, $t1
sw $t2, wans3     # wans = wnum1 % wnum2
lh $t0, hnum1
lh $t1, hnum2
mul $t2, $t0, $t1
sh $t2, hans      # hans = hnum1 * hnum2
lb $t0, bnum1
lb $t1, bnum2
div $t2, $t0, $t1
sb $t2, bans      # bans = bnum1 / bnum2
```

- **Primjer:** Napisati program u asembleru za izračunavanje površine i zapremine kvadra, čije su stranice a=73cm, b=14cm, c=16cm.
- **Rješenje:** Površina kvadra je P=2(a*b+a*c+b*c), a zapremina V=a*b*c.

```
#  Deklaracija podataka
.data
a_stranica:     .word 73
b_stranica:     .word 14
c_stranica:     .word 16
zapremina:      .word 0
povrsina:       .word 0

#  Text/kod zadatka
.text
.globl main
main:
```

```
#  Upis varijabli u registre
        lw $t0, aSide
        lw $t1, bSide
        lw $t2, cSide
# Izracunavanje zapremine, zapremina=a*b*c
        mul $t3, $t0, $t1
        mul $t4, $t3, $t2
        sw $t4, zapremina
# Izracunavanje površine, povrsina=2(a*b+a*c+b*c)
        mul $t3, $t0, $t1          # a_stranica * b_stranica
        mul $t5, $t0, $t2          # a_stranica * c_stranica
        mul $t6, $t1, $t2          # b_stranica * c_stranica
        add $t7, $t3, $t5
        add $t7, $t7, $t6
        mul $t7, $t7, 2
        sw $t7, povrsina
# Zavrsetak programa
        li $v0, 10                 # poziv za kraj
        syscall                    # sistemski poziv
.end main
```