

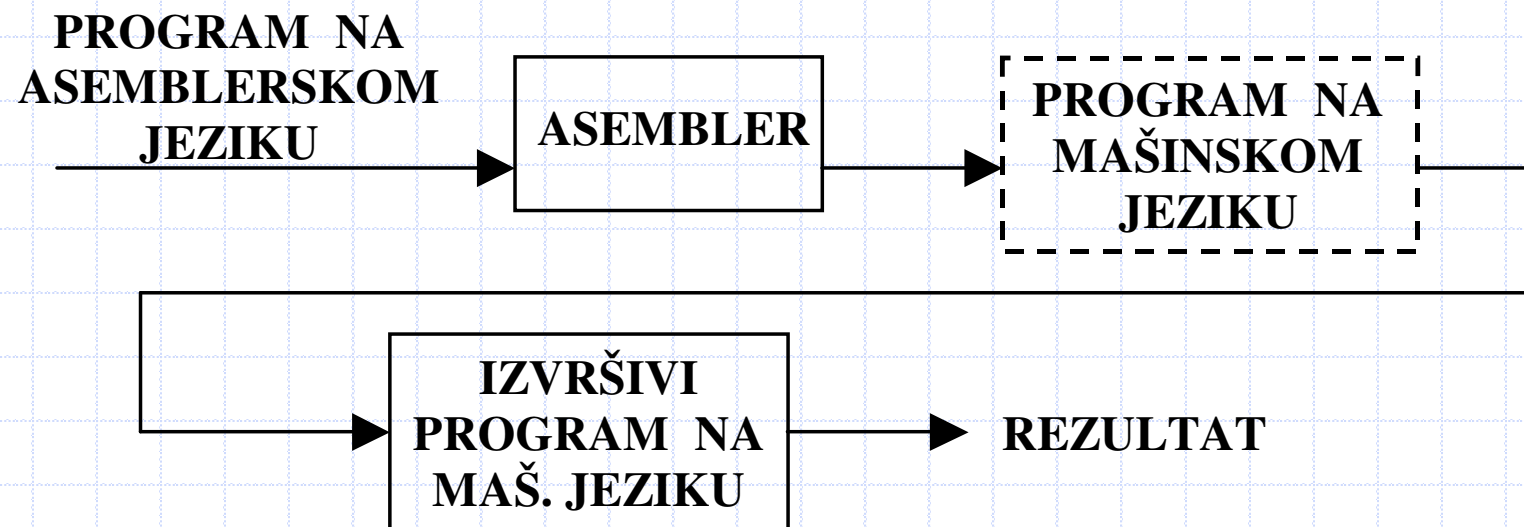
# ПРОЈЕКТОВАЊЕ АСЕМБЛЕРА

- ◆ Асемблер
- ◆ Модули асемблера

# Дефинисање новог лингвистичког нивоа превођењем

- ◆ Потребан **преводац** – алат који преводи програм написан на једном језику (на једном лингвистичком нивоу) у програм написан на неком другом језику (на другом лингвистичком нивоу)

Асемблер (асемблерски преводаца) је програм који преводи полазни (илити изворни) програм написан на асемблерском језику у програм на машинском језику



◆ Велика сличност полазног и крајњег језика чине асемблер посебном врстом преводиоца

# Операције и операнди

- ◆ Кључан део језика чине **операције** (акције) које се могу изразити и **операнди** над којима се оне могу обавити
- ◆ Главни задатак преводиоца је да **операције** из полазног језика изрази у операцијама одредишног језика и **операнде** полазног језика да замени операндима одредишног језика

# Операције и операнди

- ◆ Операције асемблерског језика (већином) тачно одговарају операцијама машинског језика (и то је основна особина асемблерског језика)
- ◆ Операције машинског језика називамо инструкцијама
- ◆ Обично исто тако називамо и асемблерске операције -> асемблерске инструкције

# Запис операције и операнда

- ◆ Разни могући начини записа, али три главне групе
- ◆ Префиксни: OPERACIJA op1 op2 ...
- ◆ Инфиксни: op1 OPERACIJA op2 ...
- ◆ Постфиксни: op1 op2 ... OPERACIJA

# Запис операције и операнда

- ◆ У C++-у све неунарне операције прате инфиксну нотацију.
- ◆ Корисник формулисањем функција суштински уводи своје операције и оне су увек у префиксној нотацији
- ◆ Преклапање оператора је посебан случај

# Формат инструкције и симболички изрази

- ◆ Асемблерски језици већином користе префиксну нотацију
- ◆ Формат асемблерске инструкције:
  - [labela] KOP [operandi] [komentar]
    - ◆ Нпр. START MOV AX,0 obriši AX
  - Операнди су симболички изрази
    - ◆ Изрази се састоје од симбола, бројева и оператора, као што су: +, -, \*
    - ◆ Нпр. 2 \* (BEGINX - ENDX)



# Примери

## ◆ MIPS32:

```
add    $t3, $t2, $0
addi   $t1, $0, 1+2*3
sub     $t2, $t2, $t1

blez    $t2, exit
nop
```

# Примери

◆ x86:

```
mov    ah, 09h
lea    dx, msg+2
int    21h
mov    ax, 4C00h
int    21h
```

# Примери

## ◆ ЛПРС:

```
inc R0, R0  
mov R2, R3  
ashl R0, R0  
dec R2, R2  
NOP
```

# Примери

◆ Али:

◆ Crystal32 DSP

```
i1 = ymem[i0]
```

```
anyreg (a0,i1)
```

```
a0&a0
```

```
if (a==0) jmp>end_premalloc:
```

```
ymem[FPT_Temp2] = x0
```

```
ymem[FPT_Temp3] = i1
```

# Примери

◆ Али:

◆ Starkey SNAP

```
AC = cmul4(WC, YC), BC+=AC ## NOP, WC = PXB0(OXB0), NOP, YC = PYB0(OYB0)
```

```
AC = cmul4(WC, YC), DC+=AC ## NOP, NOP, NOP, YC = *PYB1(OYB1)
```

```
AC = cmul4(WC, YC), BC+=AC ## NOP, NOP, NOP, NOP
```

```
B0 = B0 >> EXP ## NOP, NOP
```

• • •  
**CMP AX, BROJ+1**  
• • •

# Модови адресирања

- ◆ Регистарско: директно и индиректно
- ◆ Непосредно: непосредни операнд и апсолутно адресирање
- ◆ Имплицитно
- ◆ и разни други...

На пример:

1101 1100 0010 0011 0011 1110 0101

...  
CMP AX, BROJ+1  
...



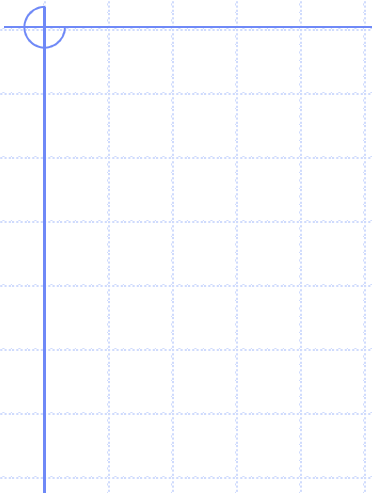
...

**BROJ: DB 5, 48**

**MOV AX, CS: BROJ**

**CMP AX, BROJ+1**

...



```
...  
MOV AX, CS: BROJ  
CMP AX, BROJ+1  
BROJ: DB 5, 48  
...
```

...

```
MOV AX, CS: BROJ
CMP AX, BROJ+1/2*3
BROJ: DB 5, 48
...
```

**BROJ: .EQU 16/4-1**

**. . .**

**MOV AX, CS: BROJ**

**CMP AX, BROJ+1/2\*3**

**. . .**

# Проблем превођења симболичких израза:

- ◆ Симболички изрази се појављују у асемблерским исказима у пољу операнда.
- ◆ Дефиниција симбола не мора претходити његовој употреби.
- ◆ Према томе асемблер мора бити организован у два пролаза.

# Први и други пролаз асемблера:

- ◆ Први пролаз: дефиниција вредности симбола који се појављују у програму.
- ◆ Други пролаз: превођење на машинске инструкције и одређивање вредности операнда израчунавањем израза који представљају вредност операнда у инструкцији.

# ТАБЕЛА СИМБОЛА:

- ◆ Дефиниција симбола представља процес придруживања симбола броју који представља његову вредност.
- ◆ Први пролаз додељује меморијску локацију свакој инструкцији.
- ◆ ТАБЕЛА СИМБОЛА служи за уписивање симбола и њима придружених вредности.

# ТАБЕЛА ЛИТЕРАЛА:

- ◆ Литерали: специјална врста симбола који се односе на константе.
- ◆ Први пролаз додељује меморијске локације и свим литералима у програму.
- ◆ ТАБЕЛА ЛИТЕРАЛА садржи адресу и бинарни запис константе која одговара датом литералу.



# Литерал?!

Пример

16-битна архитектура (инструкциона реч,  
регистар, меморија...)

Фиксна инструкциона реч.

cmp r2, 5

0111010100100101

регистар

непосредни  
операнд

cmp r0, 32000

011101010010????

шта сад?

Где сместити непосредни операнд?

# ТАБЕЛА КОДА ОПЕРАЦИЈЕ:

- ◆ Уграђена је у асемблер.
- ◆ Има по једну врсту за сваки симболички код операције.
- ◆ Дефинише разлику између машинских и псеудо операција.

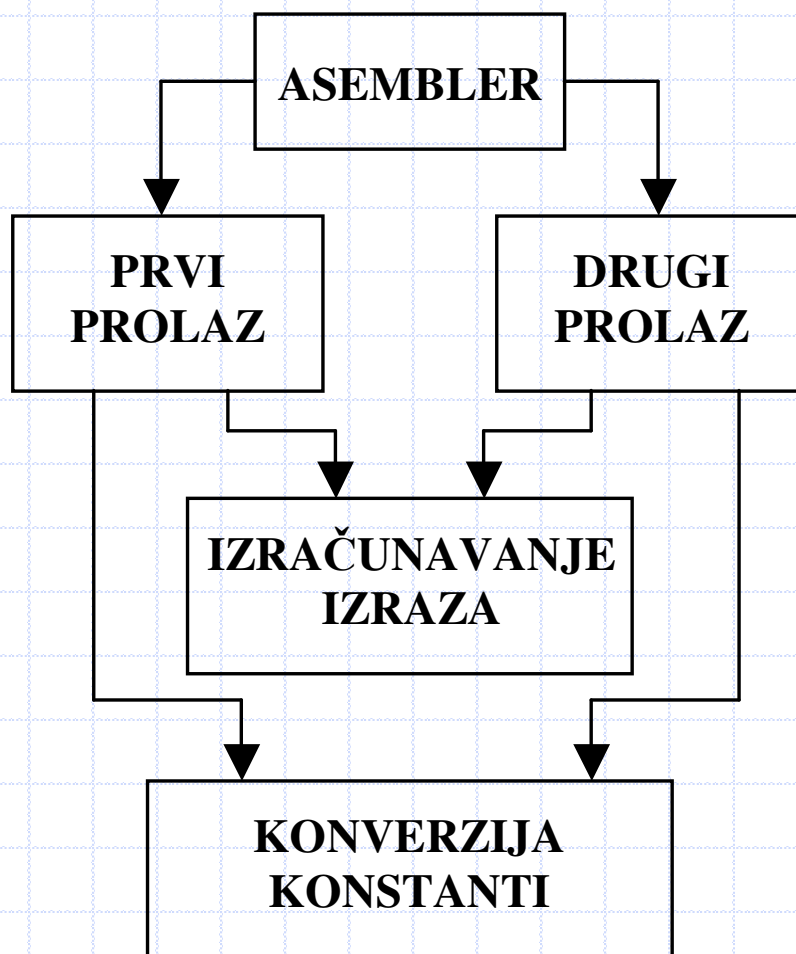
## Други пролаз завршава превођење:

- ◆ Сваки израз у пољу операнда дефинише вредност једног од поља инструкције.
- ◆ Симболички код операције се замењује његовим нумеричким кодом.
- ◆ Вредности свих поља се скупљају према формату инструкције која се преводи.

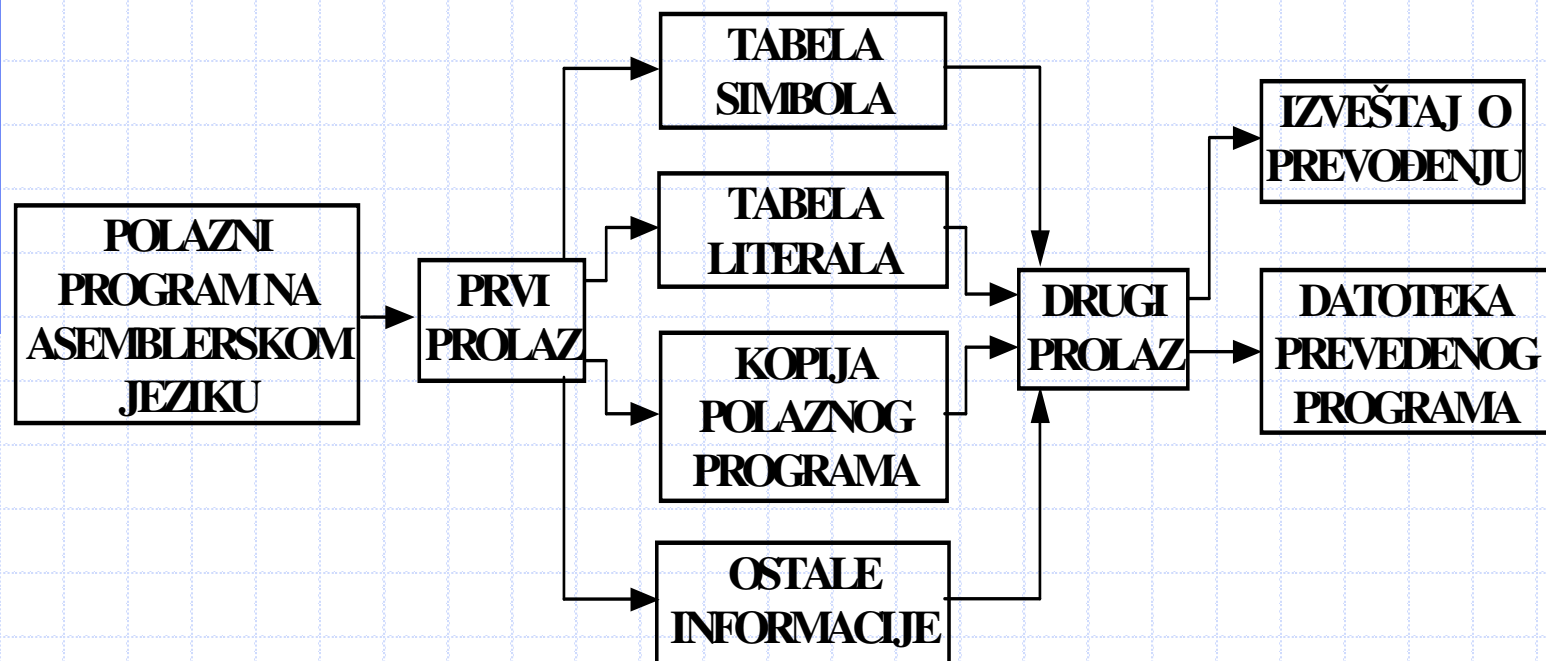
# Четири основна модула асемблера:

- ◆ Модул за први пролаз,
- ◆ Модул за други пролаз,
- ◆ Модул за израчунавање израза,
  - У I пролазу за обраду EQU, у II пролазу за одређивање вредности операнда
- ◆ Модул за конверзију константи.
  - У I пролазу за обраду литерала, у II пролазу за обраду  $\alpha$ : DATA  $\beta$  и литерала

# Односи између модула асемблера:



# Ток информација у првом и другом пролазу:



# Једноставан пример (1/2)

- ◆ Како се асемблира овај једноставан програм?
  - Табела КОП је наравно позната унапред

	TITLE	PRVI
BROJ	DB	5,48
	MOV	AX, CS: BROJ
	CMP	AX, BROJ+1
	JNE	DVA
DVA	HLT	
	END	

TKOP	
MNEMONIK	KOD
MOV ...	2E A1
CMP ...	2E 3B 06
JNE ...	75
HLT	F4

# Једноставан пример (2/2)

	TITLE	PRVI
BROJ	DB	5,48
	MOV	AX, CS: BROJ
	CMP	AX, BROJ+1
	JNE	DVA
DVA	HLT	
	END	

MAŠINSKI PROGRAM	
LOKACIJA	SADRŽAJ
0000	05
0001	30
0002	2E
0003	A1
0004	00
0005	00
0006	2E
0007	3B
0008	06
0009	01
000A	00
000B	75
000C	00
000D	F4

T S	
SIMBOL	VREDNOST
BROJ	0000
DVA	000D

TKOP	
MNEMONIK	KOD
MOV ...	2E A1
CMP ...	2E 3B 06
JNE ...	75
HLT	F4



# Пример BubleSort (1/3)

1			
2	0000		
3	0000		
4			
5	0000		
6	0000	55	
7	0001	8B	EC
8	0003	56	
9			

```
DOSEG
.MODEL    SMALL
.CODE
PUBLIC    _BubbleSort
_BubbleSort PROC
    push    bp
    mov     bp, sp

    push    si                ;sačuvaj registre
                                ; i promenljive
```

# Пример BubleSort (2/3)

10 0004 53  
11 0005 33 DB

12 0007 51  
13 0008 33 C9

14 000A 52  
15 000B 50

16 000C 33 C0  
17

18 000E 8B 76 04

19 0011 4E

push bx  
xor bx, bx

push cx  
xor cx, cx

push dx  
push ax

xor ax, ax

mov si, [bp+4] ; si pokazuje na niz

dec si

# Пример BubleSort (3/3)

20	0012	8B	56	06	mov	dx, [bp+6]	; broj znakova u dx
21	0015	8A	F2		mov	dh, dl	
22	0017	FE	CA		dec	dl	
23	0019	B1	00		mov	cl, 0	; i=0
24							
25	001B				FirstLoop:		
26	001B	FE	C1		inc	cl	; i=i+1
27	001D	8A	E9		mov	ch, cl	; j=i
28	001F				SecondLoop:		
29	001F	FE	C5		inc	ch	; j=j+1

# I пролаз асемблера (1/3)

- ◆ Треба да дефинише све симболе
- ◆ Зато додељује меморијске локације за:
  - За сваку машинску инструкцију
  - За сваку константу
    - ◆ које генерише псеудо операција  $\alpha$ : DATA  $\beta$
  - За сваки меморијски блок
    - ◆ резервисан исказом  $\alpha$ : BLOK  $\pi$
  - За сваку константу
    - ◆ Коју генерише литерал

# I пролаз асемблера (2/3)

- ◆ Сви елементи се смештају у меморију
  - Почев од адресе 0, по редоследу у програму
  - У ту сврху се уводи бројач локација
    - ◆ који указује на први недодељен бајт у меморији
    - ◆ који се увећава за вредност зависно од исказа
      - Машинска инструкција: за дужину инструкције
      - Псеудо инструкција  $\alpha$ : DATA  $\beta$ : за дужину података које генерише израз  $\beta$
      - Псеудо инструкција  $\alpha$ : BLOK  $r$ : за дужину блока који се резервише
      - Псеудо инструкција која не генерише излаз, нпр. EQU: за 0

# I пролаз асемблера (3/3)

## ◆ Дефинисање симбола

- Унос симбола и бројача локације у табелу симбола

## ◆ Обрада литерала

- Више литерала може генерисати исту константу
  - ◆ Прави се само једна инстанца
- Литерал се конвертује у бинарну вредност
  - ◆ Додатна информација: дужина и ограничења у погледу смештања

## ◆ Простор за литерале се додељује на крају I прол.

## II пролаз асемблера (1/2)

- ◆ Асемблирање машинских инструкција
- ◆ Одређивање сваког поља инструкције:
  - Код операције из табеле КОП
  - Конверзија и асемблирање константи  $\alpha$ : DATA  $\beta$ 
    - ◆ Мора се урачунати потреба за попуном
  - Заузимање простора за блок  $\alpha$ : BLOK  $\beta$ 
    - ◆ Мора се урачунати потреба за попуном
  - Псеудо операције које не генеришу излаз
    - ◆ Већ су обрађене – једноставно се игноришу

## II пролаз асемблера (2/2)

- ◆ Излази: машински код (.obj) и извештај (.lst)
- ◆ Извештај (листинг):
  - Редни број линије и указивач грешке
  - Адреса меморијске локације (релативно од 0)
  - Генерисани машински код и подаци
  - Изворни асемблерски код



# Пример програма у ЛПРС асемблеру

.data

a: 5

b: 7

.text

Begin: // program racuna a \* b

sub R0, R0, R0

ld R1, a

ld R2, b

Loop:

add R0, R0, R1

dec R2, R2

jmpnz Loop

NOP