К1 41 — 174

1. асемблер / линкер
2. организација меморије (конти, сег, страница).
3. дељење мем. (динамичко, преклоп, demand-paging, copy-on-write, ...)

malloc, free.

size → void*

picoRISC (373)

RISC-V

$4\,KB = 2^2 \cdot 2^{10}\,B = 2^{12}\,B$

1 | 00 00 | 0 000 | 000 0

$68 \rightarrow 64 + 4 \rightarrow 0x44$

$72 \rightarrow 64 + 8 \rightarrow 0x48$ (4·16)

1. (10 poena)

Dat je sadržaj jednog izvornog fajla sa C kodom.

```
int a[256], n;

int max_a (int n) {
  if (n<=1) return a[0];
  int m = max_a(n-1);
  return (a[n-1]>m) ? a[n-1] : m;
}
```

a)(6) Napisati asemblerski kod za 32-bitni procesor picoRISC, sa sintaksom direktiva pokazanom na predavanjima, kakav bi prevodilac mogao da napravi prevođenjem ovog fajla. Logički segment se na asembleru deklariše direktivom `seg` uz koju ide kvalifikator za tip segmenta (`text`, `bss` ili `data`); npr. `seg text`. Stek raste ka nižim adresama, SP ukazuje na poslednju popunjenu lokaciju, adresibilna jedinica je bajt, a instrukcija poziva potprograma na steku čuva PC i PSW tim redom.

b)(2) Za svaku labelu unutar asemblerskog koda prevoda funkcije `max_a` odrediti pomeraj (razliku) njene vrednosti (adrese u koju se preslikava) u odnosu na vrednost labele koja označava prvu instrukciju ove funkcije. Vrednosti pisati decimalno i obrazložiti rezultat.

c)(2) Ukoliko se logički segmenti nastali prevodom ovog fajla alociraju redom, jedan odmah iza drugog u virtuelnom adresnom prostoru, svaki logički segment poravnat na početak stranice veličine 4 KB, a prvi od njih počinje od virtuelne adrese 0, odrediti virtuelne adrese u koje se preslikavaju svi simboli u asemblerskom prevodu (labele za sve definisane podatke i one koje označavaju instrukcije prevedene funkcije `max_a`). Kratko obrazložiti rezultat.

Rešenje:

```
seg bss
a: dd (256 dup 0)
n: dd 0
end
seg text
max: 8 load r1, 8(sp)
     4 dec r1
     8 jlez r1, RETFST
     4 push r1
     8 call max
     4 pop r1
     8 load r7, #2
     4 shl r1, r7
     4 sub r2, r1, r0
     8 jlez r2, RETM
     4 load r0, r1
     4 ret
RETM: 4 ret
RETFST: 8 load r0, a
        4 ret
```

picoRISC
db – define byte 1B
dw – define word 2B
dd – def. double word 4B

```
int max (int n) {
  if (n<=1) return a[0];
  int m = max(n-1);
  if (a[n-1]>m)
    return a[n-1];
  else
    return m;
}
```

r0, 0.

call max

r0 := m
r1 := a[n-1]
r2 := a[n-1]-m. a + 4(n-1)

r0, ..., r15

RETM: 68B → 68

RETFST: 72B → 72

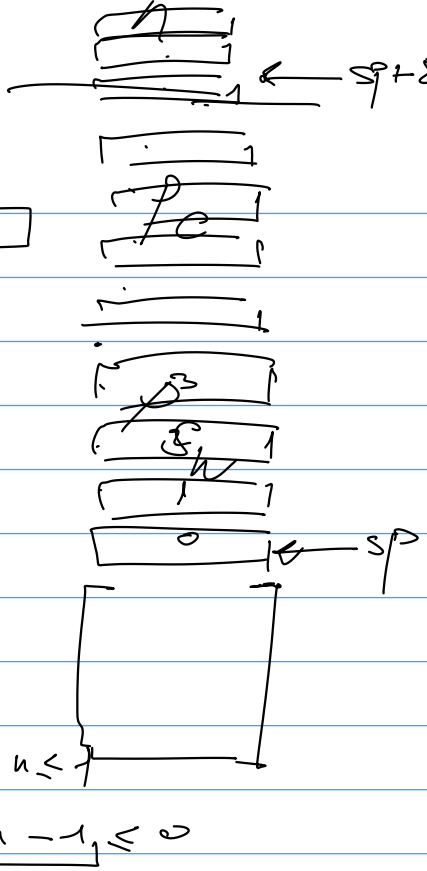
picoRISC

↝ 32b — 4B

↝ 64b — 8B

RISC-V

32b



$32 - 2^5$

JLEZ reg1, op

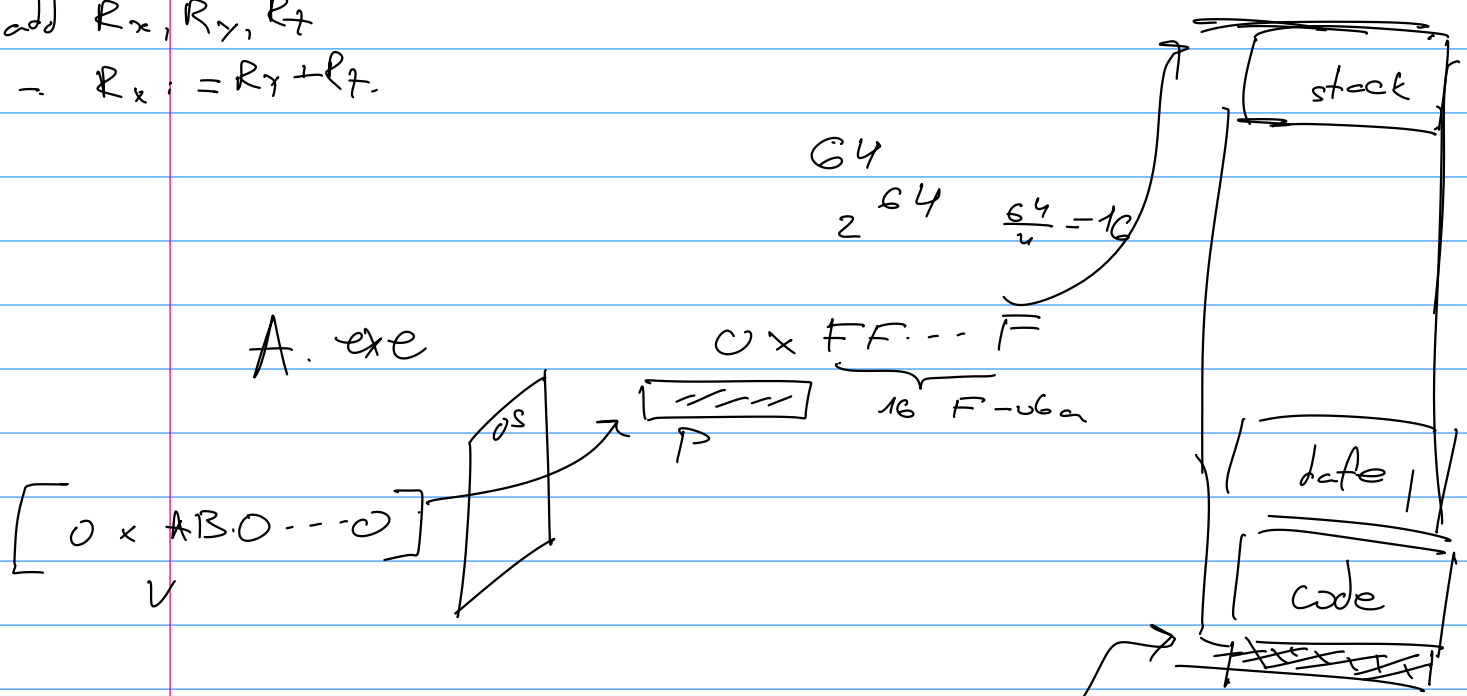
jmp if less taken or eq. to zero.

```
if (reg1 ≤ 0) {
    jmp op;
}
```

push RX ≡ load R7, #4
add sp, sp, R7
store RX, 0(sp)

add Rx, Ry, Rz

- Rx := Ry + Rz.



JZ reg, op
if (reg == 0) jmp op

1. (10 poena)

Na asembleru 32-bitnog procesora picoRISC napisati prevod dole date rekurzivne C funkcije `f`. Pretpostavke:

- vraćanje vrednosti tipa strukture iz funkcije implementira se tako što se u funkciju u registru R0 prenosi adresa lokacije u koju treba upisati povratnu vrednosti (u R0 je pokazivač na datu strukturu u koju pozivalac prihvata vraćenu vrednost);
- stek raste ka nižim adresama, a SP ukazuje na poslednju popunjenu lokaciju;
- prilikom poziva potprograma instrukcijom `call` procesor na stek najpre stavlja PSW, pa potom PC (svaki je 32-bitni); adresibilna jedinica je bajt.

```
struct S { int a, b; }
```

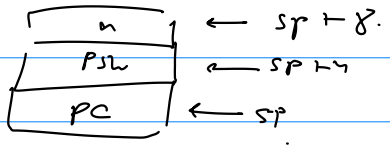
2 × 4 B = 8 B

```
struct S f (int n) {
  struct S s;
  if (n==0) s.a = s.b = 1;
  else s = f(n-1);
  return s;
}
```

r0 ← struct S*

```
f:  load r1, #8
    add  sp, sp, r1
    load r1, 16(sp)      r1 := n.
    jz   r1, SET1
    push r0
    dec  r1              r1 := n-1
    load r2, #8
    add  sp, sp, r2
    load r0, sp
    push r1
    call f.
    pop  r1
    load r2, (sp)
    store r2, 12(sp)
    load r2, 4(sp)
    store r2, 16(sp)
    load r2, #8
    sub  sp, sp, r2
    pop  r0
    jmp  EXIT:
```

r0 — return




```
f:  load r1, 8(sp)
    jz   r1, SET1
    dec  r1
    push r1
    call f.
    pop  r1
    jmp  EXIT
SET1: load r2, #1
    store r2, (r0)
    store r2, 4(r0)
EXIT: ret.
```

SET4: load r_2, #4

store r_2, (sp)

store r_2, 4(sp).

EXIT:

load r_2, (sp)

store r_2, (r_0)

load r_2, 4(sp)

store r_2, 4(r_0)

load r_2, #8

sub sp, sp, r_2

ret.

3 × 4KB = 12KB.
20B + 44B + 44B = 108B.
[(12k − 108) B]
108
12k

1. (10 poena)

Dat je sadržaj dva asemblerska fajla (`a.asm` i `b.asm`) za 32-bitni procesor picoRISC. Direktivom `export` asembleru se nalaže da navedene simbole (definisane dalje u tom fajlu) „izveze“ kao simbole dostupne za spoljno vezivanje, dok se direktivom `import` „uvoze“ navedeni simboli definisani u nekom drugom fajlu (ovom direktivom ti simboli su deklarisani u datom fajlu, ali sa nepoznatom vrednošću). Nakon prevođenja (asembliranja), linkeru se zadaje da spoji dva fajla dobijena prevođenjem fajlova `a.asm` i `b.asm`, tim redom. Linker slaže segmente u virtuelni adresni prostor redom kojim na njih nailazi počev od virtuelne adrese 0, s tim da svaki nov segment počinje od početka nove stranice veličine 4 KB (adresibilna jedinica je bajt).

a.asm

```
export main, a
import max

seg data
    a: dd 1, 2, 3, 4, 5
endseg

seg text
    main: 8 load r1,a
          4 push r1
          8 load r1,a+4
          4 push r1
          8 call max
          4 pop  r1
          4 pop  r1
          4 ret
endseg
```

b.asm

```
export max

seg text
    max:  8 load r1,[sp+8]
          8 load r2,[sp+12]
          4 cmp  r1,r2
          8 jle  L0001
          4 load r0,r1
          4 ret
    L0001:4 load r0,r2
          7 ret
endseg
```

a)(3) Koje virtuelne adrese dodeljuje linker simbolima `main`, `a` i `max` tokom povezivanja (napisati ih heksadecimalno)? Obrazložiti odgovor.

b)(3) Ukoliko instrukcija `call` na vrh steka stavlja samo PC, koje tri 32-bitne vrednosti se nalaze na vrhu steka prilikom ulaska u potprogram `max` pri izvršavanju procesa pokrenutog nad navedenim povezanim programom? Vrednosti navesti heksadecimalno u poretku kojim su stavljene na stek – poslednju onu na vrhu steka.

c)(4) Ukoliko se primenjuje stranična organizacija memorije, kolika je interna fragmentacija za sadržaj ova dva fajla (koliko je neiskorišćene memorije koja je alocirana za logičke segmente u ovim fajlovima)? Obrazložiti odgovor.

Rešenje: