Pismeni ispit iz predmeta *Operacijski sustavi za ugrađena računala*, 9.6.'10.

1. Iz navedene skripte povezivača navesti:

```
SECTIONS {
        .pocetak 0x100000 : {
                code = .;
                *(.pocetak_koda)
                . = ALIGN(4096);
        }
        .kod : {
                *(.text)
                . = ALIGN(4096);
        }
        .podaci : {
                data = .;
                *(.data)
                *(.rodata.*)
                . = ALIGN(4096);
        }
        .din_spr : {
                bss = .;
                *(.bss)
                . = ALIGN(4096);
        }
        end = .;
}
```

   a) (1 bod) imena ulaznih odjeljaka:

      `.pocetak_koda, .text, .data, .rodata, .bss`

   b) (1 bod) imena izlaznih odjeljaka:

      `.pocetak, .kod, .podaci, .din_spr`

   c) (1 bod) sve varijable definirane u skripti

      `code, data, bss, end`

   d) (2 boda) vrijednosti svih varijabli (po potrebi koristiti odgovarajuće makroe i prethodno izračunate varijable)

```
code = 0x100000,
data = code + SIZEOF(.pocetak) + SIZEOF(.kod)
bss = data + SIZEOF(.podaci)
end = bss + SIZEOF(.din_spr)
```

2. (3 boda) U nekom kodu (.c) nalazi se poziv funkcije:

```
a = zbroji ( m, n, p );
```

   Funkcija zbroji definirana je kao:

```
int zbroji ( int m, int n, int p ) {
   int suma;
   suma = m + n + p;
   return suma;
}
```

   Skicirajte instrukcije koje će prevoditelj napraviti prevođenjem zadanog koda za poziv funkcije, te za samu funkciju (uz pretpostavku da se optimizacije neće koristiti). Koristiti instrukcije procesora ARM ili x86 (ili slične).

```
// a = zbroji ( m, n, p );
push p
push n
push m
call zbroji
add $12, %esp
mov %eax, a

ili, npr.
sub $12, %esp
mov m, (%esp)
mov n, 4(%esp)
mov p, 8(%esp)
call zbroji
add $12, %esp
mov %eax, a
```

```
// int zbroji ( int m, int n, int p ) { ...
push %ebp
mov %esp, %ebp

add $4, %esp //suma

mov 12(%ebp), %eax   //eax = m
add 16(%ebp), %eax
add 20(%ebp), %eax
mov %eax, (%esp)

mov (%esp), %eax

leave
ret
```

3. (6 bodova) Jedan projekt za ugrađene sustave sastoji se od tri datoteke: `pocetak.c`, `jezgra.c` i `programi.c` (te njihova zaglavlja, ekvivalentne `.h` datoteke). Sve se datoteke sastoje od `.text`, `.rodata`, `.data` i `.bss` odjeljaka. Skicirati skriptu za povezivača (*linkera*) koji će napraviti datoteku slike sustava koja će se učitati po pokretanju sustava u spremnik na adresu 0x100000. Kôd u `pocetak.c` će odmah po početku izvođenja premjestiti kôd i podatke „jezgre“ koji nastaju iz datoteke `jezgra.c` na spremničku lokaciju 0x10000. Iza „jezgre“ kopirati će se i „programi“ (svi odjeljci iz datoteke `programi.c`), ali na prvu iduću adresu poravnatu na veličinu stranice (zadnjih 12 bitova moraju biti nule). Isto tako "poravnati" i kraj "programa".
„Jezgru“ (odjeljak `.jezgra`) treba pripremiti za fizičku adresu gdje će se ona nalaziti, tj. adresu 0x10000. „Programe“ (odjeljak `.programi`) treba pripremiti za logičku adresu 0x20000!
Dodati u skriptu varijable `programi_pocetak` i `programi_kraj` koje će sadržavati fizičku adresu početka i kraja dijela spremnika gdje se nalazi odjeljak s „programima“ (nakon premještanja), a koji su potrebni podsustavu za upravljanje spremnikom pri stvaranju novih procesa (taj se kopira za svaki novi proces). Sve se objektne datoteke nalaze u istom direktoriju (neka filter bude ime objektne datoteke). Po potrebi dodati i dodatne varijable u skriptu.

Rješenje:

```
SECTIONS {

	.pocetak {
		pocetak.o (.text)
		pocetak.o (.rodata)
		pocetak.o (.data)
		pocetak.o (.bss)
	}

	pocetak_jezgre = .;
	.jezgra 0x10000 AT ( pocetak_jezgre ) {
		jezgra.o (.text)
		jezgra.o (.rodata)
		jezgra.o (.data)
		jezgra.o (.bss)
		. = ALIGN(4096);
	}

	programi_pocetak = pocetak_jezgre + SIZEOF ( .jezgra );
	.programi 0x20000 AT ( programi_pocetak ) {
		programi.o (.text)
		programi.o (.rodata)
		programi.o (.data)
		programi.o (.bss)
		. = ALIGN(4096);
	}
	programi_kraj = programi_pocetak + SIZEOF ( .programi );

}
```

4. (3 boda) Nadopuniti navedene datoteke odgovarajućim ključnim riječima (`static`, `extern`, ...) i sl. tako da navedeno ima smisla. Sve što ne treba biti „globalno“ označiti lokalnim.
   (3 boda) Napisati odgovarajući „`Makefile`“ (pretpostaviti da nikakve dodatne zastavice i sl. nisu potrebne).

"jezgra.h"

```
int j_dohvati_znak ();
int postavi_alarm (...);
int obrisi_alarm (...);
```

"jezgra.c"

```
#include "jezgra.h"
#include "arh.h"

extern int br_zn;
int j_dohvati_znak () {
	//čekaj dok nema znakova u međuspremniku
	//arh sloja za tipkovnicu
	while ( br_zn == 0 )
		asm ( "hlt" );

	return arh_dohvati_znak ();
}

int postavi_alarm (...) { /* kod */ }
int obrisi_alarm (...) { /* kod */ }
```

"program.h"

```
void jeka ();
```

"program.c"

```
#include "program.h"
#include "jezgra.h"
static
int br_zn = 0;
static
char ms[BR_ZN] = { 0 };
static
char znak = 0;

static
void ispisi ( format, ... ) { /* kod */ }

static
void obrada_alarma () {
	ispisi ( "Stanje: ms=%s, br_zn=%d, zadnji_znak=%c\n", ms, br_zn, znak );
}
void jeka () {

	int id = postavi_alarm ( "svake 3 sekunde pozovi funkciju", obrada_alarma );

	do {
		znak = j_dohvati_znak ();
		ms[br_zn++] = znak;
	}
	while ( znak != 'q' );

	obrisi_alarm ( id );
}
```

"arh.h"

```
int arh_dohvati_znak ();
void arh_prekid_tipk ();
```

"arh.c"

```
int br_zn = 0;
static
int prvi = 0;
static
char ms[VEL_MS];


int arh_dohvati_znak () {
	if ( br_zn == 0 )
		return 0;

	char znak = ms[prvi++];

	if ( prvi > VEL_MS )
		prvi = 0;

	br_zn--;

	return znak;
}
static
int uzmi_znak_iz_tipk (){ /* kod */ }

void arh_prekid_tipk () {
	char znak = uzmi_znak_iz_tipk ();

	prvi++;
	br_zn++;

	if ( prvi > VEL_MS )
		prvi = 0;

	ms[prvi] = znak;
}
```

5. (3 boda) Pri prihvatu prekida (potpuni) kontekst prekinute dretve sprema se u dva koraka (za većinu arhitektura). Zašto su potrebna dva koraka (ili više)?

procesor „sam“ pohranjuje samo par registara, ostale treba programski (za potpuni kontekst)

6. (3 boda) Prihvaćanje prekida kojima je uzrok izvan procesora može se privremeno onemogućiti odgovarajućom zastavicom registra stanja. Zašto se prekide generirane unutar procesora ne može onemogućiti (zašto je tako sustav napravljen)?

zato jer ti prekidi ili pokazuju grešku (dijeljenje s nulom, nepostojeća instrukcija, ...) pa trenutnu dretvu treba prekinuti, ili je to programski prekid kojim dretva želi pozvati jezgrinu funkciju
i jedno i drugo je neophodno obraditi prikladnim funkcijama

7. (3 boda) Navedite osnovne funkcije/operacije podsustava za upravljanje prekidima. Koja osnovna sučelja taj podsustav treba imati?

8. (3 boda) Navedite operacije/funkcije/sučelja koja mora pružati podsustav za upravljanje vremenom.

9. (3 boda) Je li u sustavima koji nemaju višedretvenost (višezadaćnost) potreban mehanizam programskog prekida? Obrazložiti. Navesti prednosti i nedostatke takvog mehanizma za navedeni sustav.

10. (2 boda) Funkcija označene sa `inline` će se najvjerojatnije „ugraditi“ u kôd od kuda se pozivaju. Takve se funkcije vrlo često cijele navode u datotekama zaglavlja (.h). Međutim, nekad ih je ipak potrebno navesti u datotekama s kôdom. Opišite primjer kada je to neophodno.

kada se u njima koriste globalne varijable definirane u .c datoteci (mogu biti i „static“)

11. (2 boda) Navedite osnovne elemente opisnika dretvi.

12. (2 boda) Opišite princip sinkronizacije mehanizmom barijere.

13. (3 boda) Opišite mehanizme operacijskog sustava za upravljanje asinkronim događajima (generiranih izvan procesora).

14. (3 boda) Što su to *callback* funkcije (funkcije s povratnim pozivom)? U prikazanom sustavu, koji se elementi sve mogu nazvati takvim funkcijama i zašto?

15. (3 boda) U sustavu sa straničenjem koristi se sklopovska potpora te funkcije operacijskog sustava. Što radi sklopovlje a što operacijski sustav u upravljanju spremničkim prostorom?