Međuispit iz OSZUR-a, 24. 4. 2012.

_____

ime i prezime

1. a) (3 boda) Nadopuniti slijedeći kod potrebnim ključnim riječima (`include`, `static`, ...) tako da se kod može prevesti naredbom: `gcc main.c device1.c device2.c -o d2d`.

"device.h"

```
struct device_t {
        int (*init) ();
        int (*recv) ( void *data, size_t size );
        int (*send) ( void *data, size_t size );
}
```

"main.c"

```
#include "device.h"
extern device_t device1, device2;

#define M       80
int main () {
        char buffer[M];
        size_t size;
        device1.init();
        device2.init();
        while(size = device1.recv ( buffer, M ))
                device2.send ( buffer, size );
        return 0;
}
```

"device1.c"

```
#include "device.h"
static  int init () { ... }
static  int recv ( void *data, size_t size )
        { ... }
static  int send ( void *data, size_t size )
        { ... }

struct device_t device1 = (struct device_t)
{ .init = init, .recv = recv, .send = send };
```

"device2.c"

```
#include "device.h"
static  int init () { ... }
static  int recv ( void *data, size_t size )
        { ... }
static  int send ( void *data, size_t size )
        { ... }

struct device_t device2 = (struct device_t)
{ .init = init, .recv = recv, .send = send };
```

b) (2 boda) Napisati *Makefile* za prevođenje gornjih datoteka.

c) (2 boda) Navesti izlazne odjeljke koji će se pojaviti prevođenjem gornjih datoteka te navedite sadržaje tih odjeljaka (koji elementi gornjih datoteka će biti u njima).

Makefile:

```
d2d: main.o device1.o device2.o
        gcc main.o device1.o device2.o -o d2d
main.o: main.c device.h
        gcc -c main.c
device1.o: device1.c device.h
        gcc -c device1.c
device2.o: device2.c device.h
        gcc -c device2.c
```

.text => sve instrukcije
.data => device1, device2
.bss => buffer i size (na stogu)
eventualno: .rodata => 80 (ali to se najčešće ugradi u instrukciju

2. (3) Napisati makroe (sa `#define IME`) naziva `INC1(N)`, `INC2(N,X)` te `INC3(N,X)` tako da:
   * `INC1(N)` vraća vrijednost za jednu veću od `N`,
   * `INC2(N,X)` vraća vrijednost za jednu veću od `N` ako je `N<X-1` te 0 inače, te
   * `INC3(N,X)` koji povećava varijablu `N` za jedan ako je `N<X-1`, odnosno postavlja ju u 0 inače.

   Makroe napisati tako da budu uporabljivi u svim primjenama (kontekstu) koje imaju smisla (poslani parmetri N i X mogu biti i složeniji izrazi; sam makro može biti dio složenijih izraza, primjerice `INC1` može se koristiti u `INC2` a `INC2` u `INC3`). Po potrebi koristiti "uvjetno" dodjeljivanje:

   `( uvjet ? vrijednost_za_DA : vrijednost_za_NE ).`

```
#define INC1(N)      ( (N) + 1 )
#define INC2(N,X)    ( (N) < (X)-1 ? INC1(N) : 0 )
#define INC3(N,X)    do { N = INC2(N,X); } while (0)
```

3. (2) Neki zamišljeni procesor ima 4 registara opće namjene `R0-R3` te programsko brojilo `PC`, registar stanja `RS` i kazaljku stoga `SP`. Za rad sa stogom ima instrukcije `PUSH registar` i `POP registar` koje stavljaju zadani registar na stog i obnavljaju vrijednost registra sa stoga. Pri prihvatu prekida procesor sam stavlja na stog `PC` i `RS`. Ukoliko sve prekida treba obraditi funkcijom obradi_prekid (`CALL obradi_prekid`), te ukoliko se iz prekida vraćamo instrukcijom `IRET` (koja obnavlja `RS` i `PC` sa stoga i omogućuje prekide) napisati niz instrukcija koje slijede labelu `prihvati_prekid` a koje se izvode po prihvatu prekida (procesor nastavlja obradu prekida tim instrukcijama nakon što je sam na stog pohranio `PC` i `RS`).

```
prihvati_prekid:

        PUSH R0
        PUSH R1
        PUSH R2
        PUSH R3

        CALL  obradi_prekid

        POP R3
        POP R2
        POP R1
        POP R0

        IRET
```

4. (8 bodova) Ostvariti podsustav za upravljanje vremenom koji omogućuje postavljanje jednog alarma (jedina funkcionalnost). Neka sučelje koje treba ostvariti bude:

   `postavi_alarm ( vrijeme_do_aktiviranja, funkcija ).`

   Nakon isteka zadanog vremena (`vrijeme_do_aktiviranja`, u mikrosekundama, računanog od trenutka postavljanja alarma – poziva `postavi_alarm`) treba pozvati funkciju `funkcija`. Na raspolaganju stoji brojilo koje odbrojava u taktu jedne mikrosekunde, a čija se vrijednost (u mikrosekundama) postavlja sa `postavi_brojilo ( int broj )` (sa `broj=0` se brojanje isključuje) i čita sa `pročitaj_brojilo ( int *broj )` (na adresu `broj` se upisuje trenutna vrijednost brojila). Po dostizanju vrijednosti nula, brojilo izaziva prekid `PREKID_BROJILA` koji se može povezati funkcijom za obradu prekida pozivom `registriraj_prekid ( ID_PREKID, funkcija_obrade )`. Neka se podsustav, tj. sve funkcije koje ga sačinjavaju, od `postavi_alarm`, `obrada_prekida_sata` te `inicijaliziraj()`, kao i sve potrebne varijable nalaze u datoteci `alarm.c`. Napisati sadržaj te datoteke. Radi jednostavnosti vrijeme izražavati u mikrosekundama i pretpostaviti da neće doći do prekoračenja opsega brojeva tipa `int` pri njegovu korištenju za tu svrhu te da je brojilo dovoljno veliko da prihvati sve intervale.

alarm.c

```
static void (*fun) ();

static void obrada_prekida_sata ()
{
      void (*f2) ();
      postavi_brojilo (0); //nije neophodno
      f2 = fun;   //zbog mogućeg ponovnog postavljanja alarma u obradi;
      fun = NULL; //ali nije neophodno za bodove
      if ( f2 != NULL )
            f2 ();
}

void inicijaliziraj ()
{
      fun = NULL;
      postavi_brojilo (0); //nije neophodno
      registriraj_prekid ( PREKID_BROJILA, obrada_prekida_sata );
}

void postavi_alarm ( vrijeme_do_aktiviranja, funkcija )
{
      fun = funkcija;
      if (  vrijeme_do_aktiviranja > 0 )
            postavi_brojilo (  vrijeme_do_aktiviranja );
      else
            obrada_prekida_sata (); //aktiviraj odmah
}
```