1. [2 boda] Primjerima (i dodatnim opisom) demonstrirati smisao korištenja ključnih riječi `static`, `volatile` i `extern`.

2. [2 boda] Napisati makro `M1(A,B,C)` koji će vratiti vrijednost (`A`, `B` ili `C`) za koju funkcija `F1(x)` (`x` je `A`, `B` ili `C`) daje najveću vrijednost. Pretpostaviti jednostavne parametre (koji ne mijenjaju ništa, npr. neće biti `a++` ili `fun(x)` kao parametri, ali može biti `x+y`).

   Rješenje:

```
#define M1(A,B,C) \
( F1(A)>F1(B)? (F1(A)>F1(C)?(A):(C)) : ( (F1(B)>F1(C)?(B):(C)) ) )
```

3. [4 boda] Izvorni kod nekog sustava sastoji se od nekoliko datoteka raspoređenih u direktorije `jezgra`, `ui` te `programi`. U svakom se direktoriju nalaze tri datoteke s izvornim kodom čije se ime sastoji od imena direktorija te dodatkom redna broja (npr. u `ui` se nalaze `ui1.c`, `ui2.c` i `ui3.c`). Napisati datoteke `Makefile` koje treba staviti u svaki od direktorija (koje se brinu za prevođenje datoteka u tim direktorijima) uz jedan `Makefile` u početnom direktoriju. Izlazna datoteka treba se zvati `program`. Pretpostaviti da za prevođenje i povezivanje ne trebaju nikakve posebne zastavice. (Iz jednog `Makefile`-a se drugi, u nižem direktoriju `x`, može pozvati s `make -C x`.)

   Rješenje:

   jezgra/Makefile:

```
jezgra.o: jezgra1.o jezgra2.o jezgra3.o
    (LD) -r jezgra1.o jezgra2.o jezgra3.o -o jezgra.o
#ovo dalje nije neophodno, implicitna pravila su dovoljna
jezgra1.o: jezgra1.c
    (CC) -c jezgra1.c
jezgra2.o: jezgra2.c
    (CC) -c jezgra2.c
jezgra3.o: jezgra3.c
    (CC) -c jezgra3.c
```

   ui/Makefile:

```
ui.o: ui1.o ui2.o ui3.o
    (LD) -r ui1.o ui2.o ui3.o -o ui.o
```

   programi/Makefile:

```
programi.o: programi1.o programi2.o programi3.o
    (LD) -r programi1.o programi2.o programi3.o -o programi.o
```

   Makefile:

```
program: jezgra/jezgra.o ui/ui.o programi/programi.o
    (LD) jezgra/jezgra.o ui/ui.o program/iprogrami.o -o program
jezgra/jezgra.o:
    (MAKE) -C jezgra
ui/ui.o:
    (MAKE) -C ui
programi/programi.o:
    (MAKE) -C programi
```

   Može i drukčije, npr.:

   jezgra/Makefile:

```
jezgra1.o: jezgra1.c
jezgra2.o: jezgra2.c
jezgra3.o: jezgra3.c
```

ui/Makefile

```
ui1.o: ui1.c
ui2.o: ui2.c
ui3.o: ui3.c
```

programi/Makefile

```
programi1.o: programi1.c
programi2.o: programi2.c
programi3.o: programi3.c
```

Makefile:

```
JEZGRA = jezgra/jezgra1.o jezgra/jezgra2.o jezgra/jezgra3.o
UI = ui/ui1.o ui/ui2.o ui/ui3.o
PROGRAMI= programi/programi1.o programi/programi2.o programi/programi3.o

program: $(JEZGRA) $(UI) $(PROGRAMI)
    (LD) $(JEZGRA) $(UI) $(PROGRAMI) -o program
$(JEZGRA):
    (MAKE) -C jezgra
$(UI):
    (MAKE) -C ui
$(PROGRAMI):
    (MAKE) -C programi
```

4. [4 boda] Za neki ugradbeni sustav zadani su zahtjevi na pripremu programa za učitavanje u ROM. Program se sastoji od dvije datoteke `d1.c` i `d2.c`. U prvoj `d1.c` nalazi se (pored ostalog) i polje `int a[N]={/*početne vrijednosti*/}` koje treba pripremiti za učitavanje (za rad) na adresi `A1`. U drugoj datoteci nalazi se slična struktura `float b[]={/*početne vrijednosti */}` koju treba pripremiti za adresu `B1`. Kopiranje polja `a` i `b` iz ROM-a na zadane adrese (`A1`, `B1`) treba napraviti u funkciji `move_ab()`. Napisati skriptu za povezivača, deklaracije polja `a` i `b` (proširiti već navedeno) te funkciju `move_ab()`. Adresa ROM-a je `R1`, adresa RAM-a `M1`. Pretpostaviti da ostala kopiranja potrebnih dijelova iz ROM u RAM radi neka druga funkcija (nije ju potrebno ostvariti) te da sve instrukcije i konstante ostaju u ROM-u, a sve varijable se kopiraju u RAM, na početak.

Rješenje:

ldscript.ld

```
SECTIONS{
  .rom R1 : AT(R1) {
    * (.text .rodata)
  }
  m_poc = R1 + SIZEOF(.rom);
  .ram M1 : AT(m_poc) {
    * (.data .bss)
  }
  a_poc = m_poc + SIZEOF(.ram);
  .v_a A1 : AT (a_poc) {
    d1.o ( .v_a )
  }
  b_poc = a_poc + SIZEOF(.b_poc);
  .v_b B1 : AT (b_poc) {
    d2.o ( .v_b )
  }
  kraj = b_poc + SIZEOF(.v_b);
}
```

kodovi

```
/* u d1.c */
int a[N]={/* poč. vr. */}__
attribute__((section(".v_a"));

/* u d2.c */
float b[M]={/* poč. vr. */}__
attribute__((section(".v_b"));

/* "negdje" */
void move_ab(){
  extern char a_poc, b_poc, kraj;
  char *a = &a_poc, *b = &b_poc;
  char *am = A1, *bm = B1;
  while ( a < b )
    *am++ = *a++;
  while ( b < &kraj )
    *bm++ = *b++;
}
```

5. [4 boda] Sklop za prihvat prekida ima dva registra: KZ (kopija zastavica) i TP (tekući prioritet). Bitovi različiti od nule u registru KZ označavaju da dotična naprava traži obradu prekida, dok nule označavaju naprave koje ne traže prekid ili je njihov zahjev prihvaćen (u obradi). Naprava spušta svoj zahtjev kada se pozove funkcija za obradu prekida te naprave. U sustavu ima N naprava i svaka je spojena na svoj ulaz sklopa za prihvat prekida (svaka ima različiti prioritet). Prekidi većeg prioriteta trebaju prekidati obradu prekida manjeg prioriteta. U registru TP bitovi postavljeni u 1 označavaju da je dotični prekid u obradi (procesor postavlja i briše registar TP – to treba ugraditi u kod). Npr. ako je vrijednost KZ $= 00010011_2$ i TP $= 00100100_2$ tada naprave s indeksima/prioritetima 4, 1 i 0 imaju postavljen zahtjev za prekid dok se trenutno obrađuje zahtjev naprave 5, a prekinuta je obrada naprave 2 (koja se treba nastaviti po završetku prioritetnijih). Neka postoji funkcija $msb(x)$ koja vraća indeks najznačajnije jedinice (npr. $msb(001000_2) = 3$). Ostvariti prekidni podsustav (`void inicijaliziraj()`, `void registriraj_prekid (int irq, void (*obrada)())` te `void prihvat_prekida()` koja se poziva svaki puta kad se prekid prihvati, bez argumenata!) za opisani sustav uz pretpostavku da će sklop proslijediti zahtjev većeg prioriteta od tekućeg prema procesoru, dok će one manjeg prioriteta zadržati. Pri prihvatu prekida, prije poziva `prihvat_prekida` kontekst prekinuta posla spremljen je na stog, a nakon povratka iz iste funkcije sa stoga se obnavlja kontest (ne treba ga programski spremati/obnavljati).

Rješenje:

```
void (*fun[N])();
void inicijaliziraj () {
    int i;
    for ( i = 0; i < N; i++ )
        fun[i] = NULL;
    TP = 0;
}
void registriraj_prekid ( int irq, void (*obrada)() ){
    fun[irq] = obrada;
}
void prihvat_prekida () {
    int irq = msb ( KZ );
    TP = TP | (1<<irq);
    dozvoli_prekidanje();
    fun[irq]();
    zabrani_prekidanje();
    TP = TP ^ (1<<irq);
}
```

6. [4 boda] Neki sustav posjeduje 10 bitovno brojilo na adresi `CNT` koje odbrojava frekvencijom od 100 kHz. Upisom neke vrijednosti u brojilo započinje odbrojavanje prema nuli. Kada brojilo dođe do nule, izaziva prekid te se učitava zadnja upisana vrijednost pa ponovno kreće s odbrojavanjem. Izgraditi sustav upravljanja vremenom koji treba imati sučelja:
   a) inicijalizacija podsustava: `void inicijaliziraj()`
   b) obrada prekida brojila: `void prekid_sata()`
   c) dohvat trenutna sata: `long dohvati_vrijeme()` (vraća vrijeme u ms)
   d) promjena trenutna sata: `void postavi_vrijeme(long novo_vrijeme_ms)`
   e) postavljanje alarma: `void alarm(long za_koliko_ms, void (*obrada)())`

   Sučelje koristi vrijednost sata u milisekundama (pretpostaviti da je tip `long` dovoljan za prikaz sata u milisekundama). Međutim, interno, obzirom da zahtjevi za postavljanjem alarma mogu doći u bilo kojem trenutku, preciznost treba biti veća (u rezoluciji brojila). Promjena sata i postavljanje alarma briše prethodno postavljeni alarm (on se ne poziva).

Rješenje:

```
#define MAX 1000 //=> 10 ms

long sat_ms, sat_us; // sat u ms i ostatak u mikrosekundama
long odgoda; //u ms
int ucitano; //100, 200, ... 1000
void (*funkcija)();
short *br = CNT;

void inicijaliziraj () {
    sat_ms = sat_us = 0;
    odgoda = 0;
    ucitano = MAX;
    *br = ucitano;
}
long dohvati_vrijeme () {
    return sat_ms + ( sat_us + (ucitano - *br) * 10 ) / 1000;
}
void postavi_vrijeme ( long novo_vrijeme_ms ) {
    sat_ms = novo_vrijeme_ms;
    sat_us = 0;
    odgoda = 0;
    *br = ucitano = MAX;
}
void alarm ( long za_koliko_ms, void (*obrada)()) {
    odgoda = za_koliko_ms;
    funkcija = obrada;
    ucitano = odgoda * 100;
    if ( ucitano > MAX )
        ucitano = MAX;
    *br = ucitano;
}
void prekid_sata() {
    sat_us += ucitano * 10;
    sat_ms += sat_us / 1000;
    sat_us = sat_us % 1000;

    if ( odgoda > 0 ) { // nije još
        odgoda -= ucitano / 100;
        if ( odgoda > 0 ) {
            ucitano = odgoda * 100;
            if ( ucitano > MAX )
                ucitano = MAX;
            else
                *br = ucitano;
        }
        else { // alarm
            odgoda = 0;
            if ( ucitano < MAX )
                *br = ucitano = MAX;
            funkcija();
        }
    }
}
```