1. [2 boda] Na zadanome kodu označiti koji dio (varijable, kod) će se staviti u koji odjeljak pri prevođenu (koristiti samo `.text`, `.data` i `.bss` odjeljke).

```
#include "zaglavlje.h"
struct nesto[10];                          .bss
int a = 3;                                 .data
static int b = 5;                          .data
int main () {
    int x, y;                              .bss (ili drugdje, na stogu)
    x = a * 5; y = x * a;                  .text
    b += funkcija1 ( x, y, nesto );        .text
    a += funkcija2 ( nesto );              .text
    return a + b;                          .text
}
```

2. [2 boda] Zadan je makro:

```
#define LOG(LEVEL,FORMAT,...) \
fprintf ( log, #LEVEL FORMAT "\n", ##__VA_ARGS__ )
```

Ako se on pozove sa: `LOG ( A, "%d", a );` u što će se pretvoriti makro u početnoj fazi prevođenja (engl. *preprocessing*)?

Rj.: `fprintf ( log, "A%d\n", a );`

3. [2 boda] Napisati makro `KVJ(A,B,C,X1R,X1I,X2R,X2I)` za izračunavanje rješenja kvadratne jednadžbe: `A*x^2 + B*x + C = 0`. Pretpostaviti da su parametri realni brojevi te da `A` nije nula. Ulazni parametri: `A`, `B` i `C` mogu biti i složeni izrazi pa i povratne vrijednosti funkcija (npr. `KVJ(5+f1(),3,get(a),x1r,x1i,b,c)`).

Rj.:

```
#define KVJ(A,B,C,X1R,X1I,X2R,X2I)          \
do {                                        \
    double a = (A), b = (B), c = (C);       \
    double korjen = b*b - 4*a*c;            \
    if ( korjen >= 0 ) {                    \
        korjen = sqrt(korjen);              \
        X1R = (-b - korjen) / 2 / a;        \
        X2R = (-b + korjen) / 2 / a;        \
        X1I = X2I = 0;                      \
    }                                       \
    else {                                  \
        korjen = sqrt(-korjen);             \
        X1R = X2R = -b / 2 / a;             \
        X1I = -korjen / 2 / a;              \
        X2I = -X1I;                         \
    }                                       \
}                                           \
while(0)
```

4. [2 boda] Neki sustav se sastoji od datoteka: `a.h`, `a.c`, `b.h`, `b.c` te `main.c`. Odgovarajuće `.c` datoteke koriste odgovarajuća zaglavlja tj. `.h` datoteke, dok `main.c` koristi oba zaglavlja. Pri prevođenju datoteke `a.c` treba koristiti zastavicu `-DZ1`, za `b.c` zastavicu `-DZ2` te za `main.c` zastavice `-DZ3 -DZ4`. Pri povezivanju (engl. *linking*) treba postaviti zastavicu `-lnesto`. Napisati `Makefile` kojim će se moći izgraditi zadani sustav u program naziva `test1`.

Rj.: Makefile

```
test1: a.o b.o main.o
	gcc -o test1 a.o b.o main.o -lnesto

a.o: a.c a.h
	gcc -c a.c -DZ1

b.o: b.c b.h
	gcc -c a.c -DZ2

main.o: main.c a.h b.h
	gcc -c main.c -DZ3 -DZ4
```

5. [2 boda] Čemu služe ključne riječi: **`extern`**, **`static`**, **`inline`** i **`volatile`**? Opisati njihovo korištenje na primjerima.

6. [2 boda] Zadan je algoritam dinamičkog upravljanja spremnikom kod kojeg su slobodni blokovi u LIFO listi, tj. kod kojeg se pri oslobađanju bloka i nakon njegova eventualna spajanja sa susjednim on u listu slobodnih blokova doda na početak liste. Navesti dobra i loša svojstva tog algoritma.

7. [2 boda] Čemu služi nadzorni alarm (engl. *watchdog timer*)?

8. [2 boda] Neki sustav treba pripremiti za učitavanje u ROM na adresi `0x10000`. Podaci (odjeljci `.data` i `.bss` će se pri pokretanju kopirati na adresu `0x100000` te ih (podatke) treba pripremiti za tu adresu (ali učitati u ROM). Napisati skriptu za povezivača (engl. *linker*) koja će omogućiti navedeno. U skriptu dodati potrebne varijable.

Rj.: skripta.ld

```
ROM = 0x10000;
RAM = 0x100000;
SECTIONS {
	.kod ROM:
	{
		* (.text .rodata)
	}
	podaci_pocetak = .;
	.podaci RAM : AT ( ROM + SIZEOF(.text) )
	{
		* ( .data .bss )
	}
	podaci_kraj = podaci_pocetak + SIZEOF(.podaci);
}
```

9. [3 boda] Neki ugrađeni sustav ima tri naprave. Prve dvije N1 i N2 treba poslužiti iz obrade prekida funkcijama `n1()` i `n2()` (te funkcije postoje), dok se sa N3 upravlja programski – u petlji glavnog programa, pozivom `n3()`. Naprava N1 neće ponovno generirati zahtjev za prekid dok prethodni zahtjev te naprave nije obrađen do kraja. Isto vrijedi i za napravu N2. N3 ne generira zahtjeve za prekid. Naprava N1 jest najvažnija i njene zahtjeve treba najmanje odgađati (tj. ne odgađati). Funkcije `n1()`, `n2()` i `n3()` mogu trajati proizvoljno dugo (prema potrebi u pojedinome trenutku). Sustav posjeduje prekidni podsustav sa sučeljem:

```
registriraj_prekid ( id, funkcija );
zabrani_prekidanje ();
dozvoli_prekidanje ();
```

Pokazati ostvarenje funkcija `x_n1()` i `x_n2()` te `glavni_program()` u koje će se postaviti dodatne potrebne operacije prije poziva `n1()`, `n2()` i `n3()` (prema potrebama).

Rj.:

```
glavni_program () {
    registriraj_prekid ( N1, x_n1() );
    registriraj_prekid ( N2, x_n2() );
    dozvoli_prekidanje ();
    ponavljaj {
        n3();
    }
}
x_n1() {
    n1(); //obrada prekida sa zabranjenim prekidanjem;
}
x_n2() {
    dozvoli_prekidanje ();
    n2(); //obrada prekida s dozvoljenim prekidanjem;
    zabrani_prekidanje ();
}
```

Zadaci 10. i 11. jesu opširniji, ali oni donose "dodatne" bodove – MI nosi 20% a ovim se zadacima može ostvariti 25 bodova (5 "bonus" bodova).

10. [3 boda] Ostvariti podsustav za upravljanje vremenom sa sučeljem:

```
int dohvati_sat ( int *sek, int *usek );
int postavi_sat ( int sek, int usek );
int postavi_alarm ( int sek, int usek, void (*funkcija)() );
```

Za ostvarenje na raspolaganju stoji brojilo dohvatljivo na adresi `0x8000` koje odbrojava frekvencijom od 1 Mhz. Najveća vrijednost koja stane u brojilo je 10^9. Kada brojilo dođe do nule izazove prekid i stane. U obradi tog prekida pozove se funkcija `prekid_brojila()` (koju treba napraviti, pored gornjih). Vrijeme u sekundama i mikrosekundama je relativno u odnosu na neki početni trenutak (nebitno koji).

Rj.: (jedno od)

```
#define MAXCNT 1000000000
#define TICKSPERSEC     1000000
int zadnje_ucitano = MAXCNT;
int *brojilo = (int *) 0x8000;
int sat_sec = 0, sat_usec = 0;
int alarm_sec = 0, alarm_usec = 0;
void (*alarm) () = NULL;

int postavi_sat ( int sek, int usek )
{
    //provjere preskočene ( sek >= 0 && usek >= 0 && usek < 1000000 )
    sat_sec = sek;
    sat_usec = usek;
    alarm = NULL; //poništava se alarm (napomenuto na ispitu)
    zadnje_ucitano = MAXCNT;
    *brojilo = zadnje_ucitano;
    return 0;
}

int dohvati_sat ( int *sek, int *usek )
{
    //provjere preskočene ( sek != NULL && usek != NULL )
    *sek = sat_sec + ( zadnje_ucitano - *brojilo ) / TICKSPERSEC;
    *usek = sat_usec + ( zadnje_ucitano - *brojilo ) % TICKSPERSEC;
    if ( *usek >= TICKSPERSEC ) {
        *usek = *usec - TICKSPERSEC;
        *sek = *sek + 1;
    }
    return 0;
}
//nastavak na idućoj strani
```

```
int postavi_alarm ( int sek, int usek, void (*funkcija)() )
{
    // provjere preskočene:
    // ( sek >= 0 && usek >= 0 && usek < 1000000 && funkcija != NULL )

    //relativan alarm: za {sek,usek} ga aktiviraj
    alarm_sec = sek;
    alarm_usec = usek;

    alarm = funkcija;

    prekid_brojila ();
}

void prekid_brojila ()
{
    //ažuriraj sat
    sat_sec += ( zadnje_ucitano - *brojilo ) / TICKSPERSEC;
    usec += ( zadnje_ucitano - *brojilo ) % TICKSPERSEC;
    if ( usec >= TICKSPERSEC ) {
        usec -= TICKSPERSEC;
        sec++;
    }

    zadnje_ucitano = MAXCNT;
    *brojilo = zadnje_ucitano;

    if ( alarm != NULL ) {
        if ( alarm_sec + alarm_usec == 0 ) {
            void (*tmp)() = alarm;
            alarm = NULL;
            tmp ();
        }
        else {
            if ( alarm_sec < MAXCNT / TICKSPERSEC )
                zadnje_ucitano = alarm_usec + alarm_sec * TICKSPERSEC;
            //else zadnje_ucitano = MAXCNT; -- već prije postavljeno

            //koliko još ostane za idući puta?
            alarm_sec -= zadnje_ucitano / TICKSPERSEC;
            alarm_usec -= zadnje_ucitano % TICKSPERSEC;
            if ( alarm_usec < 0 ) {
                alarm_sec--;
                alarm_usec += TICKSPERSEC;
                if ( alarm_sec < 0 ) {
                    //greškica u nepreciznosti; idući prekid je alarm
                    alarm_sec = 0;
                    alarm_usec = 0;
                }
            }
        }
    }
}
```

11. [3 boda] U nekom sustavu sučelje za rad s napravama jest:

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

Napisati upravljački program za napravu X korištenjem gornjeg sučelja. Pretpostaviti da je naprava dostupna na adresama S (za slanje), R (za čitanje) i C (za statusni registar). Čitanjem podatka na adresi C dobiva se status naprave. Ukoliko je prvi bit pročitanog broja postavljen onda se s adrese R može pročitati idući podatak (ima ga). Ukoliko je drugi bit postavljen može se napravi poslati novi podatak (ona će ga moći prihvatiti). Radi ubrzanja rada za ulaz i izlaz dodati međuspremnike kapaciteta 4096 B (rezervirati ih s `malloc()`) i koristiti ih za pohranjivanje novih podataka iz naprave, odnosno, za privremenu pohranu kada se podaci ne mogu proslijediti prema napravi. Operacije `send` i `recv` trebaju koristiti te međuspremnike (u prethodno opisanim situacijama).

Rj.: (jedno od)

```
#define MS          4096
#define int8        unsigned char

struct ms {
    int8 bi[MS], bo[MS];
    int bi_f, bi_l, bi_sz, bo_f, bo_l, bo_sz;
};

static void x_interrupt_handler ( struct naprava_t *n );

static int x_init ( struct naprava_t *n )
{
    n->param = malloc (sizeof (struct ms) );
    memset ( n->param, 0, sizeof (struct ms) );
    registriraj_prekid ( X, x_interrupt_handler, n );
}

void x_interrupt_handler ( struct naprava_t *n )
{
    struct ms *ms = n->param;
    int8 *s = (int8 *) S, *r = (int8 *) R, *c = (int8 *) C;

    while ( ms->bi_sz < MS && ( (*c) & 1 ) ) {
        ms->bi[ms->bi_l] = *r;
        ms->bi_sz++;
        ms->bi_l = ( ms->bi_l + 1 ) % MS;
    }
    while ( ms->bo_sz > 0 && ( (*c) & 2 ) ) {
        *s = ms->bo[ms->bo_f];
        ms->bo_sz--;
        ms->bo_f = ( ms->bi_f + 1 ) % MS;
    }
}
//nastavak na idućoj strani
```

```
int x_send ( struct naprava_t *n, void *data, size_t size )
{
    struct ms *ms = n->param;
    int8 *d = data, sz = size;
    int8 *s = (int8 *) S, *c = (int8 *) C;

    //prvo probaj poslat izravno na napravu
    while ( ms->bo_sz == 0 && ( (*c) & 2 ) && sz > 0 ) {
        *s = *d;
        d++;
        sz--;
    }
    //ostatak u ms
    for ( ; sz > 0 && ms->bo_sz < MS; ) {
        ms->bo[ms->bo_l] = *d;
        d++;
        sz--;
        ms->bo_sz++;
        ms->bo_l = ( ms->bo_l + 1 ) % MS;
    }

    if ( sz > 0 )
        return size - sz; //toliko je ukupno poslano i stavljeno u ms
}

int x_recv ( struct naprava_t *n, void *data, size_t size )
{
    struct ms *ms = n->param;
    int8 *d = data, sz = size;
    int8 *r = (int8 *) R, *c = (int8 *) C;

    //prvo čitaj iz ms
    for ( ; ms->bi_sz > 0 && sz > 0; ) {
        *d = ms->bi[ms->bo_f];
        d++;
        sz--;
        ms->bi_sz--;
        ms->bi_f = ( ms->bi_f + 1 ) % MS;
    }
    //sada probaj čitat izravno s naprave
    while ( ( (*c) & 1 ) && sz > 0 ) {
        *d = *r;
        d++;
        sz--;
    }

    if ( sz > 0 )
        return size - sz; //toliko je pročitano
}

/* sučelje */
struct naprava_t x = { .init = x_init, .send = x_send, .recv = x_recv };
```