1. (2) Sljedeći makro za zbrajanje dva kompleksna broja ima nekoliko nedostataka:
   `#define CADD(A,B,Z) {Z->x = A->x + B->x; Z->y = A->y + B->y;}`
   Popraviti makro tako da se ti nedostaci uklone. Makro ne vraća vrijednost, ali se treba moći pozvati od bilo kuda i sa složenim parametrima (npr. i poziv `CADD(a=xz(x,y),z++,&w)` treba raditi ispravno).

```
3 stvari su potrebne:
1. do-while
2. zagrade oko A/B/Z
3. lokalne varijable da se svaki argument samo jednom javlja u kodu

#define CADD(A,B,Z)                    \
do {                                   \
    typeof(A) tmp_a = (A);             \
    typeof(B) tmp_b = (B);             \
    typeof(Z) tmp_z = (Z);             \
    tmp_z->x = tmp_a->x + tmp_b->x;    \
    tmp_z->y = tmp_a->y + tmp_b->y;    \
}                                      \
while(0)

typeof(A) daje tip argumenta
npr. za A tipa "struct z *" dio
    typeof(A) tmp_a = (A);
će se preprocesirati u:
    struct z * tmp_a = (A);
```

2. (2) Pero i Ana koriste git za rad na zajedničkom projektu. U nekom trenutku oboje su krenuli s identičnim repozitorijem, ali su tada oboje radili različite promjene nad istom datotekom `modul.c`. Ana je prva dovršila svoj posao i postavila svoje promjene u zajednički repozitorij.
   a) Kako će sustav reagirati kad Pero bude htio svoje promjene postaviti u repozitorij?
   b) Što će morati Pero napraviti da i svoje promjene stavi u repozitorij?

```
a) neće mu dati jer ne sadrži Anine izmjene
   sugerirati će mu da napravi "pull"

b) 1. napraviti pull
   2. ako ima konflikata (a vjerojatno ima) treba ih riješiti
   3. add+commit+push (ili slično)
```

3. (4) Izvorni kod nekog sustava sastoji se od datoteka *arch.c*, *kernel.c* i *programs.c* u istoimenim direktorijima *arch*, *kernel* i *programs*. Prevođenje datoteka je različito u različitim direktorijima te treba napraviti zasebne upute – *Makefile*-ove, zasebni za svaki od navedenih direktorija te jedan u početnom direktoriju koji prvo mora aktivirati navedene s `$(make)-C dir` te na kraju povezati sve objekte u sliku sustava `slika.elf`. Napišite sadržaj potrebnih *Makefile*-ova. Zastavice za prevođenje datoteka u pojedinom direktoriju označiti sa: `zast_<dir>` (ostale zastavice nisu potrebne).

`arch/Makefile:`

```
CFLAGS = zast_arch
arch.o: arch.c #+implicitna pravila

ili
arch.o: arch.c
    (CC) zast_arch -o arch.o -c arch.c
```

`slično za kernel/Makefile i programs/Makefile`

`glavni Makefile:`

```
OBJEKTI = arch/arch.o kernel/kernel.o programs/programs.o
slika.elf: $(OBJEKTI)
    $(CC) $(LDFLAGS) -o $@ $^ #ili sve navesti izravno

.PHONY: $(OBJEKTI) #nije neophodno
$(OBJEKTI): #ili tri ovakva zasebna za: arch/arch.o kern...
    $(MAKE) -C $(dir $@)
```

4. (4) Izvorni kod nekog sustava sastoji se od datoteka u direktorijima *arch*, *kernel* i *programs*. Pokretač sustava (bootloader) koji se nalazi u drugom dijelu ROM-a (koji se ne može mijenjati) pri pokretanju sustava će dio iz ROM-a s adresa 0x10000 do 0x20000 kopirati u priručni spremnik na adresi 0x60000 – u taj dio treba učitati sve što nastaje iz datoteka iz direktorija *arch* i *kernel*. Ostatak sadržaja ROM-a, od 0x20000 do 0x50000, pokretač će kopirati u radni spremnik na adresu 0x100000 – u taj dio treba učitati sve što nastaje iz direktorija *programs*. Napisati skriptu za povezivanje koja će pripremiti sliku sustava koju treba upisati u ROM (posebnim programom koji će znati interpretirati sliku), ali da se pripreme za izvođenje iz priručnog spremnika, odnosno radnog spremnika, prema opisanoj specifikaciji.

`ldscript.ld:`

```
SECTIONS {
    .cache 0x60000 : AT ( 0x10000 ) {
        arch* (*)
        kernel* (*)
    }
    .ram 0x100000 : AT ( 0x20000 ) {
        programs* (*)
    }
}
```

5. (4) Neki sklop za prihvat prekida ima 16 ulaza. Na svaki ulaz može biti spojena samo jedna naprava. Za svaki ulaz postoji upravljački/statusni registar (`PP[i]`). Postavljanjem jedinice u bit 0 tog registra omogućava se proslijeđivanje prekida prema procesoru (nulom se to onemogućava). Bit 1 će postaviti naprava spojena na taj ulaz kada zahtijeva prekid. Ostali bitovi od `PP[i]` se ne koriste. Pretpostaviti da svi ulazi imaju jednak prioritet te da je vjerojatnost preklapanja dva različita zahtjeva zanemariva (uključujući obradu). Ostvariti prekidni podsustav, tj. funkcije: `void irq_init(),void *irq_register(int irq, void (*handler)(int)), int irq_enable(int irq),int irq_disable(int irq),void irq_handler()` (ova zadnja se prva poziva pri prihvatu svakog prekida). Funkcija `irq_register` vraća prethodno registriranu funkciju, dok `irq_enable` i `irq_disable` vraćaju prethodno stanje prihvata prekida za taj broj.

```
#define IRQS 16
static void (*ih[IRQS])(int);

void irq_init () {
    int i;
    for ( i = 0; i < IRQS; i++ ) {
        ih[i] =  NULL;
        PP[i] = 0; //zabrani prekide
    }
}
void *irq_register ( int irq, void (*handler)(int) ) {
    if ( irq < 0 || irq >= IRQS )
        return NULL;
    void *old_hanlder = ih[irq];
    ih[irq] = handler;
    PP[irq] |= 1; //možda zahtjev za prekid već čeka
}
int irq_enable ( int irq ) {
    if ( irq < 0 || irq >= IRQS )
        return -1;
    int prev = PP[irq] & 1;
    PP[irq] |= 1; //možda zahtjev za prekid već čeka
    return prev;
}
int irq_disable ( int irq ) {
    if ( irq < 0 || irq >= IRQS )
        return -1;
    int prev = PP[irq] & 1;
    PP[irq] &= ~1;
    return prev;
}
void irq_handler () {
    int i;
    for ( i = 0; i < IRQS; i++ ) {
        if ( PP[i] & 3 != 0 ) { //omogućeni prekidi i postoji zahtjev
            ih[i](i);
            PP[i] &= ~2;
        }
    }
}
```

6. (4) Neko brojilo odbrojava frekvencijom od 10 MHz i kada dođe do nule izaziva prekid. Trenutna vrijednost brojila se može očitati i postaviti preko registra `BR`. Ostvariti podsustav za upravljanje vremenom koje ostvaruje sat i jedan alarm s internom preciznošću od 0,1 μs sa sučeljem:
`void clock_init();void clock_set(u64 clock);u64 clock_get();`
`void clock_alarm(u64 delay, void (*handler)());void clock_irqhandler()`
Pretpostaviti da su brojilo i procesorska riječ 64 bitovni podaci `u64` (sve vrijednosti povezane s vremenom stanu u njih u traženoj preciznosti), a sat se prema van iskazuje u mikrosekundama (parametri `clock`, `delay`). Prekide izazivati samo kada je to neophodno – kada je postavljen alarm (inače se neće dogoditi uz maksimalno veliku vrijednosti u BR).

```
#define MAXCOUNT 0xFFFFFFFFFFFFFFFF

static u64 time;         //vrijeme u desetinkama mikrosekundi
static u64 last_load;    //zadnja učitana vrijednost u brojilo
static void (*alarm)(); //funkcija za aktivaciju alarma

void clock_init () {
    time = 0;
    BR = last_load = MAXCOUNT;
}

u64 clock_get () {
    return ( time + last_load - BR ) / 10;
}

void clock_set ( u64 clock ) {
    time = clock * 10;
    BR = last_load = MAXCOUNT; //mičem alarm
}

void clock_alarm ( u64 delay, void (*handler)() ) {
    time += last_load - BR;
    BR = last_load = delay * 10;
    alarm = handler;
}

void clock_irqhandler() {
    //ovdje će doći samo ako je alarm bio postavljen
    time += last_load;
    BR = last_load = MAXCOUNT;
    alarm();
}
```