OSEUP 18.6.2013.

(1.) struktura

```
struct vaprava {
    int *ctr_reg;
    void (*int_hndl)();
};
vap[N] = {NULL, NULL};
```

ϕ

```
int register(void *control_register, void (*interrupt_handler)()) {
    int i;
    for(i=0; i<N; i++) {
        if(vap[i].ctr_reg == NULL) {
            vap[i].ctr_reg = control_register;
            vap[i].int_hndl = interrupt_handler;
            return 0;
        }
    }
    return -1;
}
int unregister(void *control_register) {
    int i;
    for(i=0; i<N; i++) {
        if(vap[i].ctr_reg == control_register) {
            vap[i].ctr_reg = NULL;
            vap[i].int_hndl = NULL;
            return 0;
        }
    }
    return -1;
}
```

```
void int_handler () {
  int i;
  for(i=0; i<N; i++) {
     if(uap[i].cntreg != NULL) {
        if (*uap[i].cntr_reg == 1) {
              uap[i].int_hndl();
              *uap[i].ctr_reg = 0;
        }
  }
}
```

2.

```
int load = 0;
int *adr = 0x1000;
int ms = 250 000 / 1000;

int wdt_start (int wiliseconds) {
    load = wiliseconds
    *adr= load * ms;
    return 0;
}
cut wdt_signal () {
    *adr= load * ms;
    return 0;
}
```

```
int wdt_stop () {
    *adr = 0;
    return 0;
}
```

3. (E)

```
char buf_in [BUF_IN_SZ];
char buf_out [BUF_OUT_SZ];
int in_start, in_sz;
int out_start, out_sz;
#define IRQ 50

int devx_init (){
    in_start = out_start = in_sz = out_sz = 0;
    register_interrupt_handler (IRQ, send_and_receive);
    init (IRQ);
}

int devx_send (void *buffer, size_t size) {
    int i;
    if ( out_sz + size > BUF_OUT_SZ )
        return -1;

    for( i=0; i<size; i++) {
        buf_out [out_start++] = ((char *) buffer) [i];
        if (out_start >= BUF_OUT_SZ)
            out_start = 0;
        out_sz ++;
    }
    if ( get_status () == 0 ) {
        send_and_receive (); // ili sadji ();
    return 0;
}

int devx_recv ....

⇒ vrlo slično!
```

```
void send_and_receive () {
    if ( get_status () != 0)
        return;

    static int zadnje = 0; // zadnje ili primaj, da alternira (ujc neoph.)

    if ( zadnje == 0) {
        salji ();
        primaj ();
    }
    else {
        primaj ();
        salji ();
    }
}

void salji () {
    if (out_sz == 0)
        return;
    if (out_start + out_sz < BUF_OUT_SZ) {
        send (&buf_out[out_start], out_sz);
        out_start += out_sz;
        out_sz = 0;
    }
    else {
        send (&buf_out[out_start], BUF_OUT_SZ - out_start);
        out_sz -= BUF_OUT_SZ - out_start;
        out_start = 0;
    }
}

void primaj () ... slično
```

4. <u>strukture podataka</u>

- opruga zaglavlja, npr:

```
struct header {
    int size;
    struct header *next;
};
```

- lista za slobodne blokove

```
void *malloc(int size) {
    prodi listu slob. blokova i nadi blok odg. veličine
    ako ga nema
        vrati NULL
    inače {
        uzmi blok iz liste
        ako je preveliki podijeli ga na dva dijela
        drugi vrati u listu
        dio dodijeli korisniku => return adresa na zaglavlje
    }
}
void free(void *adr) {
    blok na adr - vel. zaglavlja
    - probaj spojiti sa susjednim sl. blokovima
    - stavi ga u listu slobodnih
}
```

---

5. `#define MAX(A,B,C) \`

```
do {
    int _a = A;
    int _b = B;
    if (_a > _b)
        C = _a;
    else
        C = _b;
} while(0)
```

6. zamijeni_dretve (stara, nova) {

spremi_sve_registre_opće_namjene na stog (ili dretvu)
spremi ~~kontekst stoga~~ registar stanja;
na stog stavi „tu_nastavi";
adresu stoga pohrani u opisnik dretve stara;
odnovi ~~kontekst~~ kontekst stoga iz opisnika dretve nova;
odnovi registar stanja sa stoga
odnovi PC sa stoga (upr. rutr. RET);

tu_nastavi:
vrati se iz funkcije (zamijeni_dretve);

}

---

12. SECTIONS {

. jezgra 0x10000 {

jezgra* (.text);
jezgra* (.rodata);
jezgra* (.data);
jezgra* (.bss);

}

. prog1_ROM . ;

. prog1 0 : AT (. prog1_ROM) {

prog1* (.text .rodata .data .bss)

}

. prog1_size = SIZEOF (. prog1);

. prog2_ROM = . prog1_ROM + . prog1_size;

. prog2 0 : AT (. prog2_ROM)

... slično kao iza prog1 ...