Sažetak predavanja: Arhitektura računala 2 (Ugradbeni sustavi)

1. Ulazno/izlazni (I/O) uređaji u ugradbenim sustavima

- **Mikrokontroler = mikroprocesor + I/O uređaji** na jednom čipu.
- CPU komunicira s I/O uređajima pomoću **registara**:
  - **Podatkovni registri** (data registers) – pohranjuju podatke za prijenos.
  - **Statusni registri** – sadrže informacije o stanju uređaja.
- **Načini komunikacije CPU-a s I/O uređajima**:
  1. **Isolated I/O (izolirani I/O)** – koristi odvojene sabirnice za memoriju i I/O uređaje.
  2. **Memory-Mapped I/O (memorijski mapiran I/O)** – koristi zajedničke sabirnice za memoriju i I/O uređaje.

---

2. Memory-Mapped I/O u ARM arhitekturi

- **Adrese registara I/O uređaja** pohranjuju se u isti memorijski prostor kao i memorijski registri.
- **Dohvaćanje podataka iz I/O uređaja** radi se identično kao za memorijske registre.
- **Primjer koda u ARM asembleru**:

```
DEV1 EQU 0x1000  ; Definicija memorijske adrese I/O uređaja
LDR r1, #DEV1    ; Postavljanje adrese uređaja
LDR r0, [r1]     ; Čitanje iz registra uređaja
LDR r0, #8       ; Priprema vrijednosti za zapis
STR r0, [r1]     ; Zapisivanje u registar uređaja
```

---

3. Komunikacija s I/O uređajima u jeziku C

- **Pokazivači** omogućuju direktan pristup memorijskim adresama I/O uređaja.
- **Peek i poke funkcije**:

```
int peek (char *location) { return *location; }  // Čitanje iz memorije
void poke(char *location, char newval) { (*location) = newval; }  // Zapis u memoriju
```

- **Primjena u kodu**:

```
#define DEV1 0x1000
dev_status = peek(DEV1);  // Čitanje statusa uređaja
poke(DEV1, 8);            // Zapisivanje vrijednosti u uređaj
```

---

4. Metode komunikacije CPU-a s I/O uređajima

1. **Busy-Wait I/O (prozivanje/polling)**
    - CPU **stalno provjerava** status I/O uređaja dok operacija ne završi.
    - **Nedostatak**: CPU ne obavlja druge zadatke dok čeka.
2. **Interrupt-Driven I/O (prekidima vođena komunikacija)**
    - CPU **se ne blokira**, već nastavlja s radom dok ne primi prekid od I/O uređaja.
    - **Prednost**: Omogućava **efikasniji rad i simulaciju paralelizma**.

---

5. Busy-Wait I/O (prozivanje) – Implementacija u C-u

- **CPU neprestano provjerava status I/O uređaja** prije svake operacije.
- **Primjer ispisa niza znakova**:

```
#define OUT_CHAR 0x1000  // Registar izlaznog uređaja
#define OUT_STATUS 0x1001  // Statusni registar izlaznog uređaja
char *mystring = "Hello, world.";
char *current_char = mystring;

while (*current_char != '\0') {
    poke(OUT_CHAR, *current_char);  // Slanje znaka na izlazni uređaj
    while (peek(OUT_STATUS) != 0);  // Čekanje dok uređaj ne bude slobodan
    current_char++;  // Pomicanje pokazivača na sljedeći znak
}
```

- **Primjer kopiranja znakova iz ulaza u izlaz**:

```
#define IN_DATA 0x1000
#define IN_STATUS 0x1001
#define OUT_DATA 0x1100
#define OUT_STATUS 0x1101

while (TRUE) {
    while (peek(IN_STATUS) == 0);  // Čekanje na unos
    char achar = (char)peek(IN_DATA);
    poke(OUT_DATA, achar);
    poke(OUT_STATUS, 1);  // Početak ispisa
    while (peek(OUT_STATUS) != 0);  // Čekanje dok se ispis završi
}
```

---

6. Interrupt-Driven I/O – Prekidima vođena komunikacija

- **Prekidi omogućuju I/O uređajima da obavijeste CPU** kada su spremni.
- **CPU može raditi druge zadatke** dok ne primi signal od uređaja.
- **Sučelje između CPU-a i I/O uređaja uključuje**:
    - **Interrupt Request (IRQ)** – signal koji uređaj šalje CPU-u kada treba uslugu.
    - **Interrupt Acknowledge (IACK)** – odgovor CPU-a kada je spreman opslužiti prekid.
- **Mehanizam rada**:
    1. CPU čeka zahtjev za prekidom.
    2. Kada I/O uređaj završi operaciju, šalje prekid CPU-u.
    3. CPU prebacuje izvršavanje na **Interrupt Handler rutinu**.
    4. Po završetku, CPU nastavlja prethodno započeti zadatak.
- **Primjer Interrupt Handler rutine**:

```
void input_handler() {
    achar = peek(IN_DATA);  // Dohvaćanje znaka s ulaza
    gotchar = TRUE;  // Signalizacija da je znak primljen
    poke(IN_STATUS, 0);  // Resetiranje statusa uređaja
}

void output_handler() {
    // Nema potrebe za dodatnim operacijama – prekid označava završetak ispisa
}
```

- **Glavni program koji obrađuje znakove**:

```
main() {
    while (TRUE) {
        if (gotchar) {  // Kada novi znak stigne
            poke(OUT_DATA, achar);
            poke(OUT_STATUS, 1);  // Početak ispisa
            gotchar = FALSE;  // Resetiranje zastavice
        }
    }
}
```

---

7. Usporedba Busy-Wait i Interrupt-Driven I/O

Metoda	Prednosti	Nedostaci
Busy-Wait I/O	Jednostavna implementacija	CPU ne radi ništa korisno dok čeka
Interrupt-Driven I/O	Omogućuje CPU-u da izvršava druge zadatke	Potrebna je složenija logika (interrupt handleri)

- **Interrupt-Driven I/O je optimalno rješenje** jer omogućava CPU-u da radi druge zadatke dok čeka na I/O uređaj.

Ključni koncepti za učenje

□ Razlika između Isolated I/O i Memory-Mapped I/O

□ Kako CPU komunicira s I/O uređajima (registarska komunikacija)

□ Implementacija Memory-Mapped I/O u ARM asembleru i C-u

□ Busy-Wait I/O – kako radi i kada se koristi

□ Interrupt-Driven I/O – mehanizam prekida i njegova prednost

□ Implementacija prekida u C-u (Interrupt Handler rutine)