Template_Examen_PM_1

1. Primul microprocesor Intel 4004 a fost folosit pentru a construi un

A fost folosit pentru a construi calculatoare de mici dimensiuni și a demonstrat pentru prima dată că logica unui CPU putea fi pusă pe un singur cip.

2. Cîate generat, ii de computere sunt recunoscute pîanăa astăzi? Care sunt acestea?

1. Prima generație (1940–1956) – Tuburi electronice

- Foloseau **tuburi vidate** pentru procesare.
- Programarea se făcea în **cod mașină** sau **asemblor**.
- Exemple: **ENIAC**, **UNIVAC**.

---

2. A doua generație (1956–1963) – Tranzistori

- Înlocuirea tuburilor cu **tranzistori** → mai mici, mai rapizi, mai fiabili.
- Introducerea **limbajelor de programare** de nivel înalt: FORTRAN, COBOL.
- Exemple: IBM 1401, IBM 7090.

---

3. A treia generație (1964–1971) – Circuite integrate (IC)

- Folosirea circuitelor integrate pentru creșterea performanței și reducerea costurilor.
- Introducerea **sistemelor de operare** și **multitasking**.
- Exemple: IBM System/360.

---

4. A patra generație (1971–prezent) – Microprocesoare

- Lansarea primului **microprocesor**: Intel 4004.
- PC-uri, laptopuri, smartphone-uri → toate aparțin acestei generații.
- Integrarea **rețelelor, graficii avansate și interfețelor GUI**.

---

5. A cincea generație (în curs de dezvoltare) – Inteligență artificială (AI)

- Se axează pe **calcul cognitiv**, **AI**, **machine learning**, **cloud computing**.
- Obiectiv: calculatoare care **înțeleg și învață** ca un om.
- Exemple: sisteme AI (chatboturi, recunoaștere vocală), supercomputere.

3. (a) Care e singura m˘asur˘a fiabil˘a de m˘asurare a performant,ei computerelor?
   (b) Descriet, i arhitectura s, i performant,ele stat, iei/laptopului pe care lucrat, i.

Nu există o singură măsură absolut fiabilă, dar **cea mai utilizată și comparabilă măsură standardizată este numărul de instrucțiuni pe secundă**, exprimat în:

- **MIPS** (Million Instructions Per Second) – pentru procesoare mai vechi.
- **FLOPS** (Floating Point Operations Per Second) – mai ales pentru aplicații științifice.
- **Benchmark-uri standardizate** (mai fiabile decât MIPS):
  - **SPECint / SPECfp** – pentru comparații între procesoare în aplicații reale.
  - **Geekbench, Cinebench, PassMark** – pentru PC-uri și laptopuri moderne.

Concluzie:
Nu MIPS-ul brut este decisiv, ci **benchmark-urile reale** sunt considerate cele mai **fiabile** metode pentru compararea performanței între computere în funcție de aplicații specifice.

4. Avem doua computere cu urm˘atoarele caracteristici, av^and acelas, i set de instructiuni:

• Computer 1: Frecvent,a ceasului = 2GHz, CPI (cicluri medii pe instruct, iune) = 1,5

• Computer 2: Frecvent,a ceasului = 1500MHz, CPI = 1

Pentru a compara **performanța celor două computere**, folosim formula timpului de execuție:

$$\text{Timp execuție} = \frac{\text{Număr instrucțiuni} \times \text{CPI}}{\text{Frecvența ceasului}}$$

Putem **compara timpul relativ de execuție** presupunând că ambele execută același număr de instrucțiuni.

Calcul timp relativ de execuție:

- **Computer 1**:

$$T_1 = \frac{1.5}{2} = 0.75 \text{ unități de timp}$$

- **Computer 2**:

$$T_2 = \frac{1}{1.5} = 0.666\ldots \approx 0.67 \text{ unități de timp}$$

Concluzie:

📌 **Computerul 2 este mai rapid**, pentru că are un timp de execuție mai mic, chiar dacă are o frecvență mai mică, datorită unui CPI mai eficient (1 vs. 1.5).

> Chiar dacă frecvența e mai mică, un CPI mai bun poate compensa și chiar depăși performanța unui procesor mai rapid dar mai ineficient pe instrucțiune.

5. USART ^ın microcontrolere bazate pe AVR funct, ioneaz˘a ^ın care dintre urm˘atoarele moduri?

R: all the above

6. Ce fel de ADC este implementat pe ATmega 328?

R: Convertor A/D de aproximare succesivă

7. Ce frecvent,a s, i ciclul de lucru va fi generat˘a dup˘a configurarea Timer1 pe ATmega 328 care rulează la fclk=16 MHz?

Configurația explicată:

1. `DDRD |= (1 << PD5);`

→ Configurează pinul **PD5 / OC1A** ca **ieșire digitală** (pentru semnal PWM).

2. `OCR1A = 7;`

→ Valoare la care Timer1 face **compare match** pentru ieșirea OC1A (definește **duty cycle**).

3. `ICR1 = 16;`

→ Definește valoarea **TOP** a timerului în modul **Fast PWM cu ICR1**. Deci, contorul merge de la 0 la 16.

4. `TCCR1A = (1 << COM1A1) | (0 << WGM10);`

→ COM1A1 = 1, COM1A0 = 0 ⇒ **PWM neinvertat** (OC1A e HIGH până la compare match).
→ WGM11 și WGM10 = 0 (temporar)

5. `TCCR1B = (1 << WGM13) | (1 << CS10);`

→ WGM13 = 1, WGM12 = 0, CS10 = 1 ⇒ **modul Fast PWM (WGM13:10 = 1000)**, prescaler = 1.

8. Cum sunt gestionate ˆıntreruperile multiple?

> Gestionarea întreruperilor multiple într-un microcontroler (cum este **ATmega328**) se face printr-un mecanism numit **sistem de priorități și mascarea întreruperilor**. Iată cum funcționează:
>
> **1. Vectori de întrerupere**
>
> Fiecare sursă de întrerupere are un **vector de întrerupere unic** (adresă în memoria de program). Când apare o întrerupere, controlul este transferat la acel vector.
>
> **2. Ordinea de prioritate (implicită)**
>
> În microcontrolerele AVR, **prioritatea este determinată de ordinea vectorilor în tabelul de întreruperi**:
>
> - Vectorii mai sus în tabel au **prioritate mai mare**.

- Exemplu: întreruperea externă INT0 are prioritate mai mare decât INT1 sau Timer0 Overflow.

3. Global Interrupt Enable (bitul I din registrul SREG)

- Când apare o întrerupere, AVR **dezactivează automat alte întreruperi** (bitul I este resetat) pentru a preveni **întreruperile recursive**.
- Se reactivează cu instrucțiunea `reti`.

4. Masca de întreruperi individuale

- Fiecare sursă de întrerupere are propriul **bit de activare** (ex: `TIMSKx`, `EIMSK`, `PCICR` etc.).
- Acestea pot fi activate/dezactivate software, pentru a permite sau bloca anumite întreruperi.

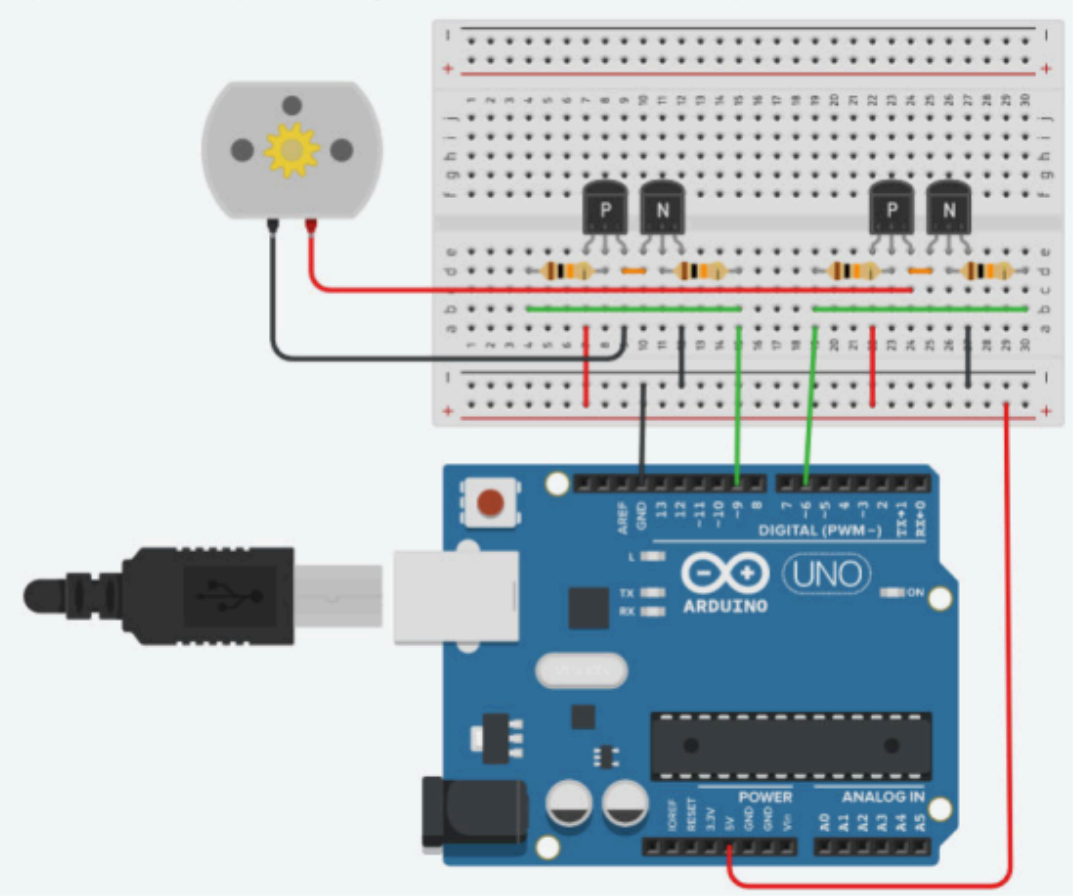
5. Cum se gestionează întreruperi multiple simultane?

- Dacă **două întreruperi apar în același timp**, este **servisată prima conform priorității vectorului**.
- După terminarea primei întreruperi (cu `reti`), bitul I este restaurat → sistemul poate prelua următoarea întrerupere.

Rezumat logic:

1. Apare o întrerupere → se sare la ISR → bitul I este resetat.
2. ISR rulează → alte întreruperi sunt **temporar blocate**.
3. Se termină cu `reti` → bitul I revine → următoarea întrerupere poate fi procesată.
4. Prioritatea decide ordinea când sunt simultane.

9. Analizați circuitul de mai jos. Motorul se rotește în sensul acelor de ceasornic când firul roșu este la +5V și firele negre este la 0V. Ce se întâmpla când:



(a) Pinul 9 este setat "1" și pinul 6 este setat "0"?

(b) Pinul 9 este setat "0" și pinul 6 este setat "1"?

(c) Pinul 9 este setat "1" și pinul 6 este setat "1"?

(a) Pinul 9 este „1", pinul 6 este „0"

- Tranzistorii controlați de pinul 9 (T1) vor conduce → se conectează un capăt al motorului la **GND**

- Tranzistorii controlați de pinul 6 (T2) sunt opriți → celălalt capăt rămâne **conectat la +5V** ➡️ Tensiunea peste motor: +5V → 0V → motorul se **rotește în sensul acelor de ceasornic**

(b) Pinul 9 este „0”, pinul 6 este „1”

- Tranzistorii controlați de pinul 6 vor conduce → un capăt al motorului ajunge la **GND**
- Pinul 9 este „0” → celălalt capăt conectat la **+5V**
  ➡️ Tensiunea peste motor: invers față de cazul anterior → motorul se **rotește în sens invers acelor de ceasornic**

(c) Pinul 9 este „1”, pinul 6 este „1”

- Toți cei 4 tranzistori sunt comandați → risc de **scurtcircuit între +5V și GND** prin tranzistori opuși
  ➡️ **Situație periculoasă**, posibil **scurtcircuit**, motorul **nu funcționează corect**

10. Mapati 32K de la adresa 0xCB000 si reprezentati EPROMSEL grafic corespunzator.

1. Determinarea dimensiunii și a intervalului de adresare

- **Dimensiunea EPROM: 32 KB = 2152^{15}215 → deci e nevoie de 15 biți de adresă**
- **Adresa de start: `0xCB000`**
- **Adresa de sfârșit: `0xCB000 + 0x7FFF = 0xD2FFF`**

Intervalul de adresare:

[0xCB000, 0xD2FFF]\textbf{[0xCB000, 0xD2FFF]}[0xCB000, 0xD2FFF

2. Convertim adresele în binar pentru a găsi care biți se modifică:

- **`0xCB000` = `1100 1011 0000 0000 0000`**
- **`0xD2FFF` = `1101 0010 1111 1111 1111`**

Se modifică ultimii 15 biți, deci:

- **Biții [14:0] → folosiți pentru adresarea internă în EPROM**
- **Biții [19:15] → identificatori de chip / semnal de selectare EPROMSEL**

3. Când activăm EPROMSEL?

EPROMSEL trebuie să fie activ (de exemplu, logic LOW dacă e activ pe 0) doar atunci când adresa este între `0xCB000` și `0xD2FFF`. Asta înseamnă:

EPROMSEL activ caˆnd A19...A15=11001 (ıˆncepe cu CB) paˆnaˇ la 11010 (D2)\text{EPROMSEL activ când A19...A15} = 11001\text{ (începe cu CB) până la 11010 (D2)}EPROMSEL activ caˆnd A19...A15=11001 (ıˆncepe cu CB) paˆnaˇ la 11010 (D2)

Pm_2_2022

1. De ce a ales IBM Intel 8088 ˆın loc de Motorola 68000?

R: Compatibilitate cu x86 – 8088 era compatibil cu 8086, pentru care exista deja software (ex: MS-DOS).

Cost mai mic – avea bus de date pe 8 biți, deci folosea memorii și periferice mai ieftine.

Disponibilitate – Intel oferea suport tehnic și garanții de livrare rapidă.

MS-DOS era deja pregătit pentru 8088, iar IBM voia lansare rapidă.

Motorola 68000 era mai puternic, dar 8088 era mai practic și ieftin.

3. (a) Numit, i ceea ce considerat, i a fi un microprocesor important.

(b) De ce?

(a) Un microprocesor important: Intel 8086

(b)A fost primul procesor x86 (1978), punând baza pentru întreaga arhitectură de PC-uri moderne.

A introdus setul de instrucțiuni x86, folosit și astăzi în majoritatea procesoarelor Intel și AMD.

A fost folosit pentru dezvoltarea primelor PC-uri IBM, definind standardul industriei.

4. Pe care dintre urmˇatorii parametri ar trebui sˇa fie de acord emit,ˇatorul s, i receptorul

ˆınainte de a ˆıncepe o comunicare pe serialˇa?

◯all of the mentioned (correct)

6. Care dintre urmˇatoarele este corectˇa? TWI este un alt nume pentru I2 (correct)

7.Care sunt unit˘at, ile de baz˘a ale unui computer cu arhitectur˘a von Neumann?

Arhitectura von Neumann, propusă în 1945, stă la baza majorității calculatoarelor moderne. În această arhitectură, un computer este compus din 5 unități de bază:

Unitățile de bază ale arhitecturii von Neumann:

1Unitatea de control (Control Unit – CU)

- Coordonează execuția instrucțiunilor.
- Extrage (fetch) instrucțiunile din memorie și le decodează.

2Unitatea aritmetică și logică (ALU – Arithmetic Logic Unit)

- Execută operații matematice și logice (ex: adunări, comparații).

3Memoria (Memory Unit)

- Conține atât datele, cât și instrucțiunile programului (caracteristică esențială în arhitectura von Neumann).
- Adresabilă secvențial.

4Unitatea de intrare (Input Unit)

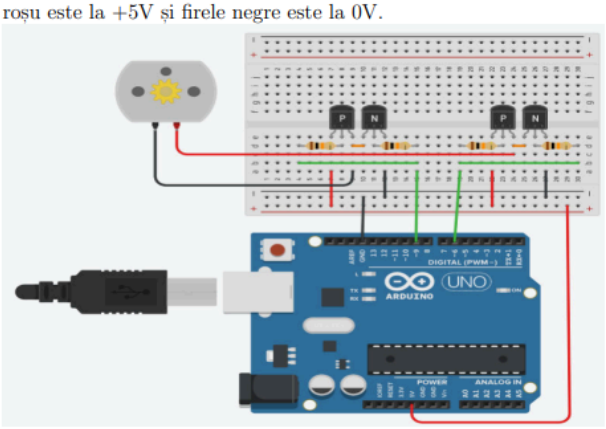
- Primește date din exterior (ex: tastatură, mouse, senzori).

5Unitatea de ieșire (Output Unit)

- Trimite rezultate către exterior (ex: ecran, imprimantă, LED-uri).

În arhitectura von Neumann, **datele și instrucțiunile folosesc aceeași magistrală (bus)**, ceea ce poate duce la **„von Neumann bottleneck”** – o limitare a vitezei din cauza accesului alternativ la memorie.

9. Analizați circuitul de mai jos. Motorul se rotește în sensul acelor de ceasornic când firul roșu este la +5V și firele negre este la 0V.



Scrieti codul Arduino pentru a controla timp de 3 secunde motorul în sens invers acelor de ceasornic cu un duty cyle de 5% și alte 3 secunde în sensul acelor de ceasornic la un duty cycle de 5%?

```
void setup() {
  pinMode(6, OUTPUT);  // PWM către unul din tranzistori
  pinMode(9, OUTPUT);  // PWM către celălalt tranzistor
}
void loop() {
  // 3 secunde sens invers acelor de ceasornic (pin 6 activ, pin 9 0)
  analogWrite(6, 13);  // 5% din 255 ≈ 13
  analogWrite(9, 0);
  delay(3000);
  // 3 secunde sensul acelor de ceasornic (pin 9 activ, pin 6 0)
  analogWrite(6, 0);
  analogWrite(9, 13);  // 5% duty cycle
  delay(3000);
}
```

PM 2024

Scrie codul necesar pentru a configura un pin GPIO astfel ˆıncˆat sˇa poatˇa comunica cu un alt microcontroler.

Transmițător (pin 8 ca ieșire):

cpp

Copy

Edit

```
void setup() {
  pinMode(8, OUTPUT);
}
void loop() {
  digitalWrite(8, HIGH);
```

```
  delay(1000);
  digitalWrite(8, LOW);
  delay(1000);
}
```

Receptor (pin 7 ca intrare):

```
cpp
Copy
Edit
void setup() {
  pinMode(7, INPUT);
  Serial.begin(9600);
}
void loop() {
  Serial.println(digitalRead(7));
  delay(100);
}
```

2.Care dintre urmˇatoarele magistrale are cea mai mare lˇat, ime de bandˇa: UART/I2C/SPI?

Explicˇa de ce este mai rapidˇa decˆat celelalte douˇa.

Răspuns corect: SPI are cea mai mare lățime de bandă dintre UART, I2C și SPI.

Explicație:

1. SPI (Serial Peripheral Interface)

Viteze tipice: până la zeci de MHz (ex: 8–20 MHz)

Transfer sincron (folosește un semnal de ceas – SCK)

Comunicare full-duplex (transmitere și recepție simultană)

Fără adresare complicată ⇒ rapid și direct

2. UART (Universal Asynchronous Receiver Transmitter)

Viteze tipice: până la 1 Mbps, uzual 9600 – 115200 bps

Comunicare asincronă (fără ceas comun)

Nevoie de start/stop/parity bits ⇒ suprasarcină mai mare

3. I2C (Inter-Integrated Circuit)

Viteze tipice: 100 kbps (standard), 400 kbps (fast), max 3.4 Mbps (high speed)

Comunicare semidulpex și adresată (master-slave)

Linia comună duce la latențe mai mari

De ce SPI este mai rapidă:

Are ceas dedicat ⇒ transfer sincron, fără așteptări

Nu are overhead de adresare sau semnale de start/stop

Comunicare dedicată între dispozitive (fiecare slave are CS propriu)

Concluzie:

SPI este cea mai rapidă dintre cele 3 deoarece oferă viteze mari, comunicare sincronă și overhead minim.

3.Avand microcontrolerul ATmega 324P, 3 senzori I2C (fiecare cu o adres˘a diferit˘a) s, i 2 senzori SPI, care este num˘arul minim de pini IO necesari pentru a-i conecta ˆımpreun˘a? Este posibil˘a aceast˘a configurat, ie?

🔢 Număr minim de pini IO necesari (adicție):

Protocol	Semnal	Număr de pini	Explicație
I2C	SDA, SCL	**2**	Comun pentru toți cei 3 senzori I2C
SPI	MOSI, MISO, SCK	**3**	Comun pentru ambii senzori SPI
SPI	CS1, CS2 (select)	**2**	Câte 1 pin dedicat pentru fiecare SPI

Microprocesorul Intel 8086 poate adresa: **Răspuns corect: 1 MB**

Microprocesorul Intel 8086 are:

magistrală de adrese pe 20 de biți

adresează memorie segmentată (cu segmente de 64 KB)

2^20=1048576bytes=1MB

2 ^20 =1048576 bytes= 1 MB

6.In ce scenarii de aplicat, ie ai alege Bluetooth ˆın loc de WiFi? Explic˘a alegerea ta ˆın termeni de vitez˘a, raz˘a de acoperire s, i consum de energie.

Bluetooth este preferat în loc de **WiFi** în scenarii unde **consumul redus de energie**, **comunicarea pe distanță scurtă** și **viteza moderată** sunt prioritare.

Comparatie și explicație:

Criteriu	Bluetooth	WiFi
	Mbps (Bluetooth Classic), ~1 Mbps (BLE)	Mbps (WiFi)
	(BLE), max 100 m (Bluetooth 5)	00 m (depinde de router)
m energie	redus (în special BLE)	ai mare (necesită sursă constantă)

Scenarii unde ai alege Bluetooth:

1. **Dispozitive portabile** sau cu baterie:
    - căști wireless
    - ceasuri inteligente (smartwatch)
    - senzori medicali purtabili

- telecomenzi

2. **IoT pe distanță scurtă**:
   - senzori BLE care trimit date periodic
   - beacon-uri pentru localizare indoor
3. **Comunicare între două dispozitive fără rețea locală**:
   - telefon → boxă portabilă
   - PC → mouse/tastatură wireless

Concluzie:

Aleg **Bluetooth** în loc de WiFi când am nevoie de **consum redus de energie**, **conectare simplă** și **raza mică este suficientă**. WiFi e potrivit pentru transfer de date mari și conexiuni la Internet, dar nu pentru senzori mici sau dispozitive portabile.

7. Un inginer a scris următorul cod pentru a utiliza întreruperea timerului pentru a accesa datele senzorului. Poți ajuta inginerul să identifice erorile în această implementare?

```
unsigned int data;
void main(){
  unsigned int localdata;
  while(1){
    ...
    localdata = data;
    ...
  }
}
```

```
void timer_isr(){
  data = sensor_buf[0];
  data = (data<<8 | sensor_buf[1]);
}
```

Dacă întreruperea apare în timpul citirii/scrierii lui data, se poate obține o valoare coruptă (de exemplu, doar jumătate actualizată).

In isr poti scrie: data = (sensor.buf[0] << 8) | sensor.buf[1];

In main: unsigned int localdata;

cli(); // dezactivează întreruperile

localdata = data;

sei(); // reactivează întreruperile

8.Un inginer dores,te sˇa scrie un program care sˇa comute LED-ul. El foloses,te un timer pentru a controla GPIO-ul care este conectat la LED. Din nefericire, dupˇa descˇarcarea programului pe ATmega 324P, LED-ul nu clipes,te. Descrie patru erori posibile care ar putea ˆımpiedica LED-ul sˇa clipeascˇa (cel put, in o propozit, ie pentru fiecare).

Desigur! Iată 4 erori posibile care pot împiedica LED-ul să clipească pe un ATmega324P:

Timerul nu este pornit corect

→ Timerul poate fi configurat greșit (ex: prescaler inexistent sau WGM incorect), deci nu generează întreruperi.

ISR-ul (rutina de întrerupere) nu este definită sau atașată corect

→ Dacă funcția de întrerupere nu are numele corect (ex: TIMER1_COMPA_vect) sau nu este legată, nu se execută codul din ea.

Pinul GPIO nu este configurat ca ieșire

→ Dacă pinul conectat la LED nu este setat cu pinMode(PIN, OUTPUT) (sau echivalent în C bare-metal), LED-ul nu va fi comutat.

LED-ul este conectat greșit pe placă

→ Polaritatea inversată sau lipsa unei rezistențe de limitare poate face ca LED-ul să nu se aprindă deloc, chiar dacă semnalul e corect.