## opdracht 1

|  |
| --- |
| void T0\_overflow(void)  {  if(P0 == 0)  P0 = 1;  else if ( P0 < 128)  P0 <<= 1;  else  P0 = 1;  } |

## Opdracht 2

Outline:

1. timer op 1000

2. globale cnt

2. overflow functie

2.a if(cnt%100 == 0)

2.a.a P0 = P0 ^ 1;

2.b if(cnt%200 == 0)

2.b.a P0 = P0^2;

2c t/m g herhalen voor alle leds

2.h if(cnt%20000 == 0)

2.h.a cnt = 0;

2.h.b P0 = 0;

2.g cnt += 100

|  |
| --- |
| int cnt = 0;  void main()  {  init(); // microcontroller initialisation  initTimer0 ( 2000 ); // One-Task scheduler; a TICK every 40000\*50 micro-seconds  enableTimer0(); // activate timer  for( ;; ) { // superloop  }  }  void T0\_overflow(void)  {  if(cnt%100 == 0)  P0 = P0 ^1;  if(cnt%200 == 0)  P0 = P0 ^2;  if(cnt%400 == 0)  P0 = P0 ^4;  if(cnt%800 == 0)  P0 = P0 ^8;  if(cnt%1000 == 0)  P0 = P0 ^16;  if(cnt%2000 == 0)  P0 = P0 ^32;  if(cnt%5000 == 0)  P0= P0^64;  if(cnt%10000 == 0)  P0=P0^128;  if(cnt%20000 == 0)  {  cnt = 0;  P0 = 0;  }  cnt += 100;  } |

## Opdracht 3

1. Een systeem met een microcontroller, in plaats van een complete computer en OS. Alle software wordt er met de hand op gezet, en de programmeur heeft complete toegang tot alle poorten.
2. Een embedded systeem heeft in vergelijken met een volledige computer een limiet aan resources. Op een microcontroller kan ook maar een enkel programma draaien.
3. MIPS is het aantal miljoen instructies per seconde. Dit geeft aan hoeveel regels aan assemblycode er per seconde uitgevoerd kan worden. Hoe hoger de klokfrequentie, hoe hoger het aantal MIPS is, aangezien er maar 1 instructie per cyclus kan worden uitgevoerd.
4. C als taal is niet een hoge programmeertaal, maar werkt beter als het schrijven van Assembly-code. Assembly duurt vele malen langer om te schrijven en is ingewikkelder.
5. Op een microcontroller kan er per klokcyclus maar een enkele instructie worden uitgevoerd. Multitasking kan dus niet in de meest strikte zin. Als de klokfrequentie echter hoog genoeg is kan een embeddedsysteem met een scheduler de schijn geven van multitasking.
6. Bij time-triggered wordt er iedere keer een instructie uitgevoerd. Dit gebeurt of er input voorkomt of niet. Bij event triggered architectuur worden alle events opgeslagen, en zal de scheduler deze analyseren en hiermee taken uitvoeren.
7. Dat er een bij een bepaalde tijd altijd een functie moet worden uitgevoerd, bijvoorbeeld het knipperen van een lichtje om de paar seconden.
8. ?
9. Als de deze tijd groter is zal de timer een interrupt geven. Dan wordt de functie onderbroken. Dit is niet wenselijk aangezien de variabelen dan misschien niet compleet veranderd zijn.
10. Een functie die moet worden uitgevoerd

## Opdracht 4

|  |
| --- |
| //functie aanmaken. simpelweg een getal printen  void printInt(int a)  {  printf("%D\n", a);  }  void main(void)  {  //Een functie pointer maken die foo heet, void teruggeeft en int als ingangsvariabele neemt  void (\*foo)(int);    //foo wijst naar het adress van printInt()  foo = &printInt;    //foo aanroepen met als variabele 2. dit zal 2 printen  foo(2);  } |

Hieruit blijkt dat functiepointers vergelijkbaar zijn met delegates in C#.

## Opdracht 5

|  |
| --- |
| //LED FUNCS  void led0(void)  {  P0 = P0 ^ 1;  }  void led1(void)  {  P0 = P0 ^2;  }  void led2(void)  {  P0 = P0 ^4;  }  void led3(void)  {  P0 = P0 ^8;  }  void led4(void)  {  P0 = P0 ^16;  }  void led5(void)  {  P0 = P0 ^32;  }  void led6(void)  {  P0 = P0 ^64;  }  void led7(void)  {  P0 = P0 ^128;  }  //EINDE LEDS  void main(void)  {  // setup microcontroller  PLLCON = 0x00;  P0 = 0x00;    // Set up the scheduler  SCH\_Init\_T2();  // Add the 'Flash LED' task (on for ~1000 ms, off for ~1000 ms)  // - timings are in ticks (~1 ms tick interval)  // (Max interval / delay is 65535 ticks)  SCH\_Add\_Task(led0, 0, 100);  SCH\_Add\_Task(led1, 5, 200);  SCH\_Add\_Task(led2, 10, 400);  SCH\_Add\_Task(led3, 15, 800);  SCH\_Add\_Task(led4, 20, 1000);  SCH\_Add\_Task(led5, 25, 2000);  SCH\_Add\_Task(led6, 30, 5000);  SCH\_Add\_Task(led7, 35, 10000);  // Start the scheduler  SCH\_Start();  while(1) {  SCH\_Dispatch\_Tasks();  }  } |