A&C week 3

Remi Reuvekamp - Timo Strating - ITV2E

8 oktober 2018, Groningen

Inhoudsopgave

1. *Opdracht 1*

1.1 A

Leg uit wat in C het verschil is tussen de operatoren & en && & gaat per bit en && werkt alleen op bijvoorbeeld booleans. Booleans zijn in de standaard C libary gedefineerd als een int. Dit betekent dus dat & per bit gaat en && voor alle bit in een keer.

In andere woorden 'A & B' betekent dat er per bit een and operatie wordt gedaan. A && B geeft 1 terug als A en B meer dan 0 is.

4 & 3 = 0100 & 0011 = 0111 = 7

4 && 3 = True = 1

1.2 B

Leg uit wat in C het verschil is tussen de operatoren | en ||

| gaat per bit en || werkt op alle bits tegelijk. Zie A voor meer uitleg.

2. *Opdracht 3*

```c
#include <avr/io.h>


int main(void)
{
  static float a = 43.75;
  int addr = &a;
  int i =0;

  for (1=0; i<100; i++) {
    a = a + 0.01;
  }

    while (1){} // loop
  return 0;
}
```

2.1 A

Wat is de storage size van een float in avr-gcc ?

Avr-gcc implementeerd floats als volgens de 32-bit IEEE standaard. Wat betekent dat je 1 sign bit hebt 8 exponent en 23 bits voor een mantisse.

In ons geval heeft C deze variabele aangemaakt in het ... op adres 0x0100. C heeft hier de waarde 0x2f42 neergezet.

2.2 B

Zorg ervoor dat de compiler niet optimaliseert. Dit kun je door : Project properties > Toolchain > Compiler > Optimization level = None te zetten. Leg uit waarom we dit moeten doen.

De Compiler probeert je C code zo efficiënt en klein mogelijk te krijgen. Dit wordt gedaan omdat C je de mogelijkheid geeft om in een hogere abstractie te programmeren. Omdat je in een hogere abstractie programmeert vertaalt 1 regel C code soms naar 10-talle regels aan assembly code. Om dit niet uit de hand te laten lopen zal de compiler iedere regel aan assembly die niet nodig is niet proberen te genereren.

2.3 C

Laad het programma in de simulator, 'bouw' het programma en plaats een breakpoint op de for-statement. Start de debugger en druk 1x op continue. MetDebug>Windows>Locals kun je de (lokale)variabelen bekijken. Wat is het adres van a ?

Het adres van a is 256 oftewel 0x0100.

2.4 D

Hoeveel bytes staan er op een enkel adres in het datasegment ?

```
#include <avr/io.h>
#include <stdio.h>

int main(void)
{
  static float a = 43.75;
  static float b = 43.75;
  int addr_a = &a;
  int addr_b = &b;

  int i = 0;
  for ( i = 0; i < 100; i++) {
     a = a + 0.01;
  }
    while (1){} // loop
  return 0;
}
```

Als we de bovenstaande code uitvoeren dan zegt de debugger dat a op adres 256 te vinden is en b op adres 260. Dit is een verschil van 4. Er zitten dus 4 adressen tussen de 2 variabelen. Beide van deze waarde zijn floats en volgen de richtlijnen is dat een IEEE float met 32 bits. Als 32 bits een verschil van 4 oplevert dan staan er 32 / 4 = 16 bits = 1 byte op een enkel adres in het datasegment.

2.5 E

Met Debug>Windows>Memory kun je het datageheugen bekijken. Wat de hexadecimale representatie van de floating point waarde 43,75 ? (Zie de sheets CS week 1.) Klopt dit met wat je ziet ?

Volgens Atmel studio zou deze maarde gelijk moeten staan aan de hexedecimale representatie 0x 0000 2F42.

Dit is niet het zelfde als waar wij uitkwamen toen we het op papier met de hand gingen uitrekenen.

Wij kwamen uit op het onderstaande.

0100 0010 0010 1111 0000 0000 0000 0000

0x 422F 0000

2.6 F

Is de AVR-architectuur big of little endian ?

Zoals in opgave E te zien is gaat de AVR-architectuur anders om het de volgorde van de cellen. Tijdens het rekenen gingen wij uit van de standaard die wordt gehanteerd in de wiskunde. Dat betekend het meest linker getal heeft de hoogste waarde. Dit wordt ook wel Big endian genoemd. AVR gebruikt dus het omgekeerde hiervan. Dit wordt ook wel little endian genoemd.

2.7 G

Plaats een breakpoint op de while-lus. Wat is na afloop van de for-lus de waarde van a ? Klopt dit met wat je verwacht ?

Volgens Atmel studio heeft a nu de waarde 44,74983. We hadden wel verwacht dat er een licht afwijking in zou zitten. Maar dit is toch wel een stukje meer dan we hadden ingeschat.

3. *Opdracht 8*

Onderstaand is de code voor opdracht 8. De twee buttons zijn op Arduino pin 2 en 3. De twee leds op Arduino pin 8 en 9.

De buttons worden als een toggle voor de betreffende led.

```
#include <avr/io.h>
#include <util/delay.h>

int main( void)
{
  // NOTE: We didn't use the pins 0 and 1 on the Arduino,
  // as those are for serial and flashing messes up
  // when you connect things to those pins.

  // Init button and led pins.
  PORTD |= (1 << PD2);
  PORTD |= (1 << PD3);
  DDRB |= (1 << PB0);
  DDRB |= (1 << PB1);

  int led1State = 0;
  int led2State = 0;

  while (1) {
    // Check if led state should change.
    if (!bit_is_clear(PIND, PD2)) {
      led1State = !led1State;
      _delay_ms(200); // Easy way to ignore button 'denderen'
    }
    if (!bit_is_clear(PIND, PD3)) {
      led2State = !led2State;
      _delay_ms(200); // Easy way to ignore button 'denderen'
    }

    // Output the led state.
    if (led1State) {
      PORTB |= _BV(PORTB0);
    } else {
      PORTB &= ~_BV(PORTB0);
    }
    if (led2State) {
      PORTB |= _BV(PORTB1);
    } else {
      PORTB &= ~_BV(PORTB1);
    }
```

```
  }

  return (0);
}
```