

Rekenaarstelsels 245 - Prakties 1

31 Julie 2015

1 Doel

Leer Renesas se e2 Studio ontwikkelingsomgewing (IDE) ken.

- Stel 'n projek in e2 Studio op.
- Programmeer 'n mikroverwerker
- Integreer saamsteltaal en C-kode met mekaar en skryf 'n funksie in saamsteltaal

Alle kode-brokke wat in hierdie opdrag gelys word is ook elektronies op die module se webblad beskikbaar. Dit behoort die kopiëring daarvan na die ontwikkelingsomgewing te vergemaklik.
PS: Alle teks in blokke soos hierdie is wenke.

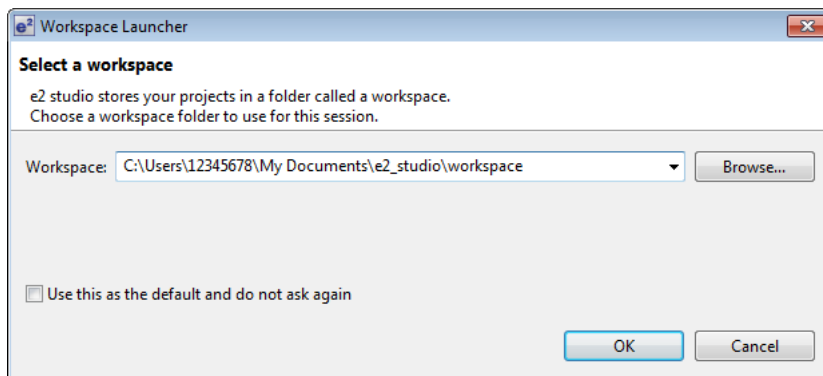
2 Opstelling van 'n projek in e2 studio

Begin deur e2 Studio oop te maak. Dit behoort te wees onder
Start ⇒ All Programs ⇒ Renesas Electronics e2 studio ⇒ Renesas e2 studio.

Wanneer die IDE vir die eerste keer oopgemaak word sal dit vra waar jou *Workspace* gestoor word. Verander dit om onder jou gebruiker se My Documents te lê.

Indien jy gevra word vir “**Toolchain Registry**”, kies GNURL78, en klik Register.

Jy word deur 'n mooi volblad verwelkomingsbladsy gegroet. **Maak dit toe.**



Wanneer daar op die laboratorium se rekenaars geprogrammeer word is dit beter om die kode op die plaaslike C:-skyf te stoor. Die H:-skyf is op 'n *server* by IT, en as almal dit gelyk gebruik raak dit vreeslik stadig. Aan die einde van die prakties kan jy jou kode vanaf die rekenaar se plaaslike C:-skyf na jou H:-skyf, of na jou *flash drive*, kopiëer. Dit sal handig wees om 'n nota te maak van e2 se standaard stoorplek.

Gaan na **File ⇒ New C Project**. 'n Nuwe venster sal oopmaak waarmee jy 'n nuwe projek skep. Let op dat ons C gebruik en nie C-plus-plus nie.

Project name RSPrak1 (of enige ander naam wat jy kies)

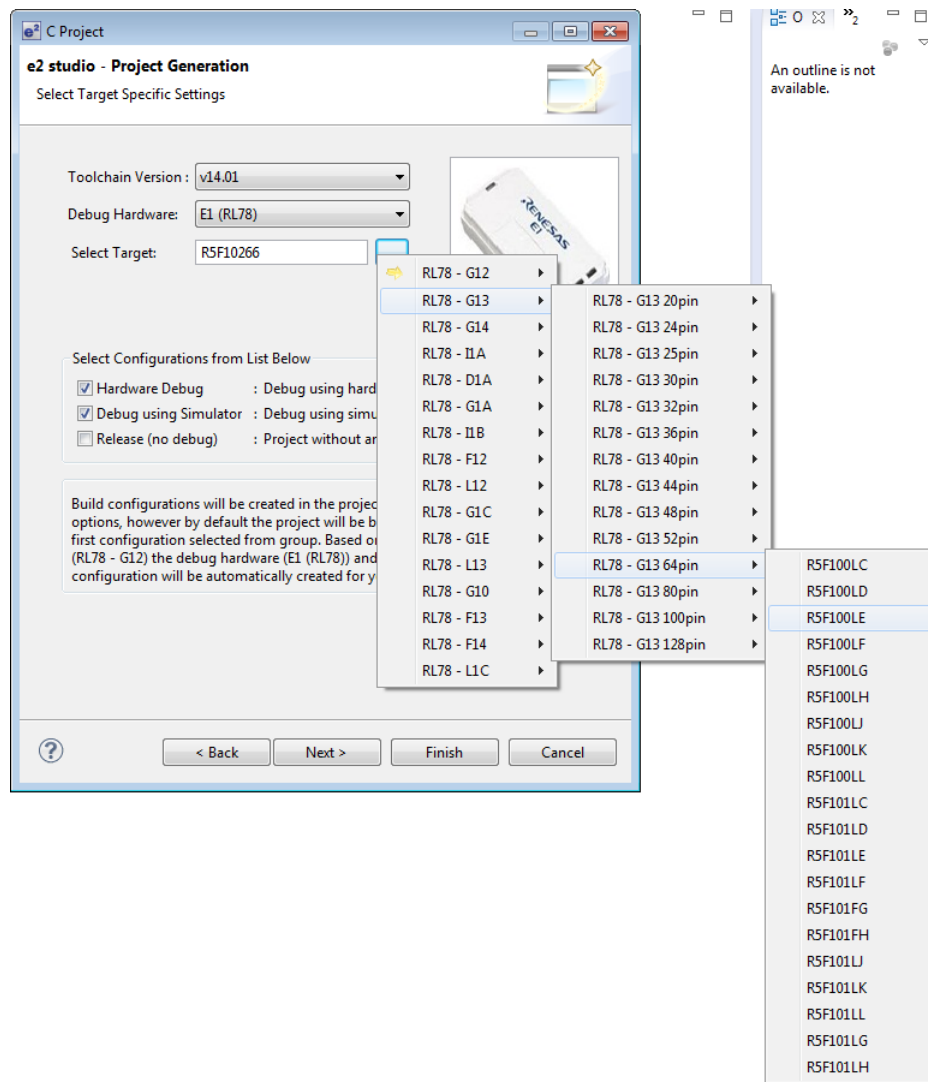
Project type Sample Project

Toolchain KPIT GNURL78-ELF Toolchain

Toolchain version v14.01 (of die nuutste een wat beskikbaar is)

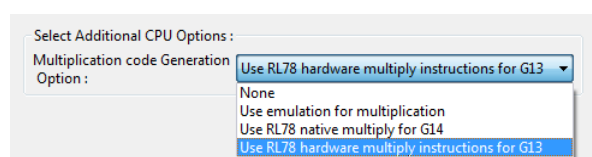
Debug Hardware E1 (RL78)

Select Target R5F100LE (RL78-G13⇒RL78-G13 64pin⇒ RF5100LE)



Code Generator Settings Ons gebruik nie die code generator nie. Dit sal wel volgende jaar in Ontwerp gebruik word.

Select Additional CPU Options Selekteer **Use RL78 hardware multiply instructions for G13**. Next.

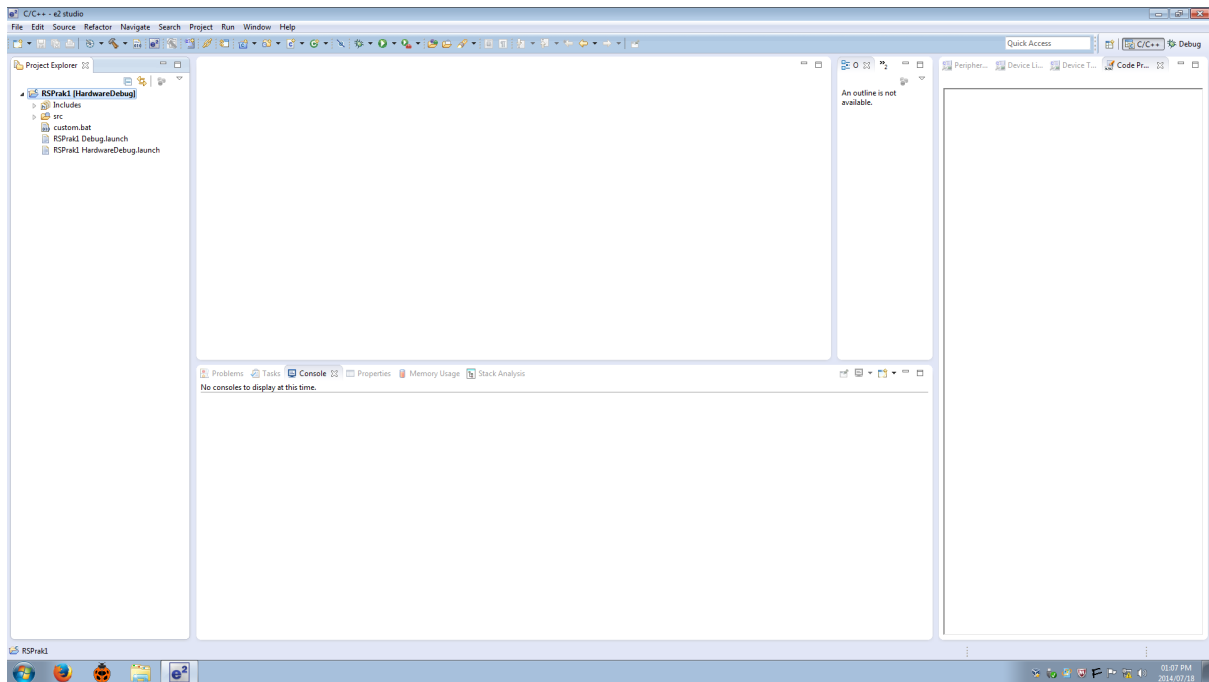


Kies die geoptimiseerde- (**Optimized**) en die vooraf saamgestelde (**Pre-Built**) biblioteke. Finish.

By die opsomming van wat jy alles gekies het, kies OK.

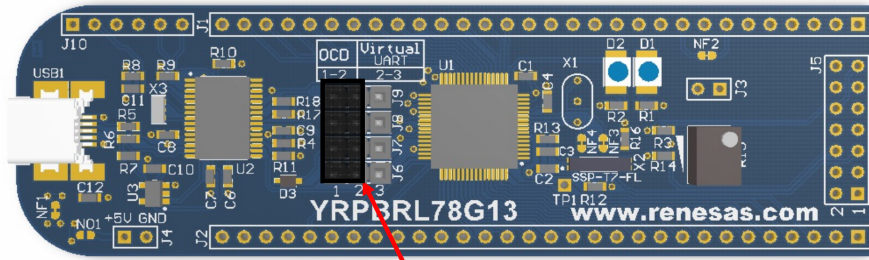
Die *code generator* genereer kode vir konfigurasies om sekere funksies van die mikroverwerker te gebruik. Dit is handig om te gebruik as jy nie alles per hand wil skryf nie, maar vir nou gebruik ons dit nie omdat ons self met die registers wil werk.

Die nuwe projek venster sal toemaak en jy sal die e2 IDE voor jou sien. As dit nie soos die lyk nie, gaan na **Window** ⇒ **Reset Perspective** ⇒ **Yes**. Nou is jou IDE op die standaard uitleg en behoort dit te lyk soos getoon word in die volgende figuur.



3 Programmeer 'n verwerker

Aan die linkerkant, onder **Project Explorer** gaan na jou projek en onder **src** maak die RSPrak1.c lêer oop. Dit mag dalk 'n ander naam hê as jy jou projek anders genoem het, maar dit behoort te wees **ProjekNaam.c**. Prop die RL78-G13 ontwikkelingsbordjie by die USB-poort van die rekenaar in. Die brugskakelaars(*jumpers*) moet op die 1-2 posisie (OCD) wees om programmering en ontfouting toe te laat. Verwys na die volgende tabel uit die ontwikkelingsbordjie se handleiding (bladsy 16).



Jumpers	Configuration
J6	1-2
J7	1-2
J8	1-2
J9	1-2

Table 2: J6 to J9, On-Board Debug / Flash Programming Mode

Wanneer die RL78 krag kry begin dit 'n program uitvoer. Die program begin deur alle noodsaaklike konfigurasies soos klokfrekwensie toe te pas. Hierdie kode word deur die IDE vir jou geskryf en lê in reset_program.asm. Nadat hierdie konfigurasie-kode uitgevoer is word die void main(void)-funksie geroep. Die main()-funksie behoort 'n oneindige lus te bevat wat sal aanhou uitvoer totdat die krag vanaf die mikroverwerker verwyder word.

Kopieër die volgende blok kode na RSPrak1.c in e2 Studio en vervang dus die bestaande main()-funksie hiermee.

```
#include "iodefne.h"

int main(void)
{
    unsigned long counter = 0;

    /*
     * Set port 7, pin 7 as an output
     * See Hardware Manual page 803: 0=output, 1=input
     */
    PM7 = 0b01111111;

    // main loop
    while (1U)
    {
        /*
         * Switch LED on
         * Make port 7, pin 7 high
         * See Hardware Manual page 187 -
         * "These registers can be set by a 1-bit or 8-bit memory
         * manipulation instruction."
         */

        P7_bit.no7 = 1;           // Shorthand to set pin high.
        //P7 = (P7 | 0b10000000); // The better way of setting
                                   // a bit in a registry high.

        // delay
        for (counter=0;counter<800000UL;counter++);

        /*
         * Switch LED off
         */
    }
}
```

```

        Make port 7, pin 7 low
        See Hardware Manual page 187
    */
    P7_bit.no7 = 0;                // Shorthand to set a pin low.
    //P7 = (P7 & 0b01111111);      // Or use the better way by reading,
                                   // bitwise AND and saving the register again.

    // delay
    for (counter=0;counter<800000UL;counter++);
}
}

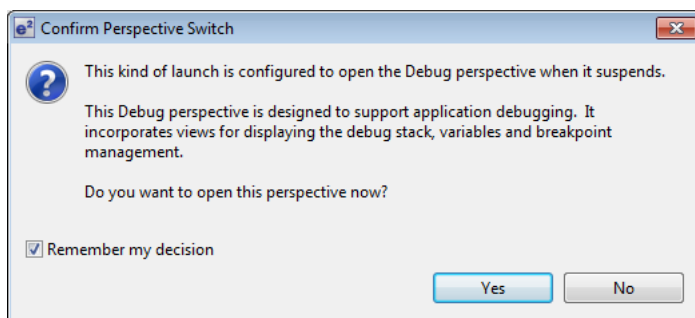
```

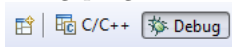
Die doel van hierdie stuk kode is om 'n LED op die ontwikkelingsbordjie aan te skakel, 'n tydjie te wag, dit af te skakel en weer 'n rukkie te wag. Dit gebeur alles in 'n oneindige lus.

Eerstens moet ons hierdie kode saamstel (*compile*). In die *Project Explorer* **regskliek op die projek** ⇒ **Build Configurations** ⇒ **Set Active** ⇒ **Hardware Debug**. Dit is moontlik om jou kode in sagteware te toets deur Debug te kies, maar vir nou kies ons dat die kode op die hardware getoets moet word.

Hardloop nou die samesteller deur in die boonste balk te gaan na **Project** ⇒ **Build All**.

Nadat die kode suksesvol saamgestel is, klik op die Debug ikoon (klein groen goggatjie). Die IDE sal nou aan die hardware verbind en dit programmeer. Indien jy 'n foutboodskap kry probeer om die ontwikkelingsbordjie uit en in te prop. Wanneer hierdie stap suksesvol gebeur het sal jy die volgende skerm sien om die IDE se uitleg te verander om in ontfouting in te gaan. Kies **Remember my decision** ⇒ **Yes**.



Vanuit die nuwe uitleg kan daar deur die kode gestap word, dit uitgevoer word, of nuwe kode geprogrammeer word. Om tussen die twee uitlegte te spring kan die kitsskakel regs bo gebruik word. 

As jy steeds 'n probleem ervaar met die programmering van die bordjie, al het jy dit uit- en ingeplug, maak seker die drywers is gelaai. Start ⇒ Regskliek op My Computer ⇒ Manage ⇒ Device Manager. Onder Ports moet 'n Renesas COM-poort gelys wees. Indien dit nie daar is nie is daar waarskynlik 'n Unknown Device afdeling met een toestel. Update Driver Software, kies per hand, Ports, Renesas UART.

Wanneer jy in die ontfoutingsperspektief is, klik op die geel/groen Play/Pause knoppie om die kode op die mikroverwerker te laat uitvoer of te stop. 

Let op dat die IDE outomaties 'n breekpunt aan die begin van die kode toegevoeg het wat veroorsaak dat dit outomaties sal Pause nadat jy hierop geklik het. Jy moet dus 'n tweede keer op die knoppie klik om die kode vry te laat voortgaan.

'n Breekpunt is 'n merker wat jy toevoeg tot 'n lyn van jou kode wat die ontfouter beveel om te wag. Dit is handig wanneer jy jou kode ontfout om te sien hoe die inhoud van jou veranderlikes en registers lyk op daardie punt.

As jou kode korrek is, die verwerker suksesvol geprogrammeer is, en die program uitvoer, behoort die een LED op die ontwikkelingsbordjie te flikker. Indien dit nie flikker nie moet jy die stappe in die tutoriaal tot hier weer deurgaen en seker maak alles is reg gedoen.

4 Funksies in C

In C plaas mens gereed kode wat baie gebruik word in funksies. Aangesien 'n LED wat aan- en afgeskakel word een van die algemeenste goed is om te doen op 'n mikroverwerker is dit slim om dit in 'n funksie te plaas. Dit help ook nogals baie vir die leesbaarheid van die kode en spaar jou die moeite om dieselfde kode oor en oor te skryf.

Kopieër die volgende blok kode na jou RSPrak.c lêer. Hardloop weer die samesteller, programmeer dit na die ontwikkelingsbordjie en maak seker die LED flikker nogsteeds.

```
#include "iodefne.h"

void switchLEDOn()
{
    P7=(P7|0b10000000);
}

void switchLEDOff()
{
    P7=(P7&0b01111111);
}

void delay(unsigned long time)
{
    unsigned long counter=0;
    for(counter=0; counter<time; counter++);
}

int main(void)
{
    /*
     * This code is more readable than the previous one. Readability and
     * reusability of the same code are two benefits of using funtions.
     */

    // Set Port 7, Pin 7 as output
    PM7 = 0b01111111;

    //main loop
    while (1) {
        switchLEDOn();
        delay(80000UL);
        switchLEDOff();
        delay(80000UL);
    }
}
```

As jy 'n foutboodskap kry in die ontfoutingsperspektief, mag dit dalk nodig wees om die verwerker te herstel (reset). Dit kan gedoen word met die ikoon wat lyk soos 'n goggatjie en 'n pyltjie wat af en regs

wys. Derde van links hier:         

5 In-lyn saamsteltaal

In-line assembly

Die taak van die samesteller (compiler) is om C-kode te vertaal na die korrekte saamsteltaal vir die spesifieke verwerker. Saamsteltaal is kodewoorde wat direk deur die logiese hekke op die verwerker geïnterpreteer word.

Saamsteltaal is kompleks en moeilik om per hand te skryf, en daarom is hoëvlak tale soos C ontwikkel. Die C-samesteller vertaal met slim reëls jou C-kode na saamsteltaal. Aangesien hierdie reëls wat die samesteller volg baie generies is en moet werk oor 'n wye verskeidenheid verwerkers en argitekture, genereer dit nie altyd die beste saamsteltaal nie.

Vir die volgende paar afdelings en praktika gaan ons fokus op saamsteltaal. Die skryf van saamsteltaal is 'n handige vaardigheid om te hê as jy eendag die taak kry om 'n stuk kode te optimiseer. Dit help ook om te verstaan as iets nie werk soos dit moet nie en jy kan lees wat die samesteller verkeerd gedoen het.

Hierdie drie stukke kode doen presies dieselfde:

C-kode	C-samesteller se uitree	Handgeskrewe saamsteltaal
P7=(P7 0b10000000);	<pre>movw 0xffef0, #0xff07 movw 0xffef2, #0xff07 movw ax, 0xffef2 movw 0xffef4, ax movw hl, 0xffef4 mov a, [hl] mov 0xffef2, a and 0xffef2, #127 movw ax, 0xffef0 movw 0xffef4, ax movw bc, 0xffef4 mov a, 0xffef2 mov 0[bc], a</pre>	<pre>movw bc, #0xff07 mov a, [bc] or a, #10000000B mov [bc], a</pre>

Om die saamsteltaal te sien wat die C-samesteller vir jou kode gegenereer het, regskliek op die nommer in die linker kantlyn langs 'n lyn in jou C-kode en sê View Disassembly.

Die volgende blok kode doen presies dieselfde as die vorige twee wat jy na die RL78 geprogrammeer het, maar die twee lyne wat in C die LED aan- en afgeskakel het is nou in-lyn vervang met saamsteltaal. Kopieër dit weer na jou RSPrak1.c, stel saam en programmeer. Het die tempo waarteen die LED flikker verander?

```
#include "iodef.h"

void switchLEDOn()
{
    /* The following block does the same as
       P7=(P7|0b10000000);
       but rewritten in assembly.
       | means bitwise OR
```

To understand this piece of code you will need to search for the keywords (mov, or) in the Software Manual.

0xFF07 is the memory mapped I/O address of register P7.
It is also called a special function register.
See Hardware Manual page 138.

```
*/

asm(";; Switch LED on          \n\t"
    "movw bc, #0xff07          \n\t"
    "mov a, [bc]                \n\t"
    "or a, #10000000B ;; Bitwise OR \n\t"
    "mov [bc], a                \n\t"
    );
}

void switchLEDOff()
{
    /* The following block does the same as
       P7=(P7&0b01111111);
       but rewritten in assembly.
       & means bitwise AND
    */

    asm(";; Switch LED off      \n\t"
        "movw bc, #0xff07        \n\t"
        "mov a, [bc]              \n\t"
        "and a, #01111111B ;; Bitwise AND \n\t"
        "mov [bc], a              \n\t"
        );
}

void delay(unsigned long time)
{
    /*
       We leave this piece of code in C for now,
       as it is complex to rewrite in asm.
    */
    unsigned long counter=0;
    for(counter=0; counter<time; counter++)
    {
        asm("nop"); // assembly for No Operation
    }
}

int main(void)
{
    // Set Port 7, Pin 7 as output
    PM7=0x7F;

    // main loop
    while (1) {
        switchLEDon();
        delay(80000UL);
        switchLEDOff();
        delay(80000UL);
    }
}
```


6 Aparte asm-lêer

Aangesien die skryf van saamsteltaal in-lyn binne C nogals deurmekaar raak, is dit beter om jou saamsteltaal in 'n aparte .asm-lêer te stoor. Die volgende twee kode lyste wys hoe dit gedoen word, en hoe die saamsteltaal-funksies vanuit C geroep word.

Julle sal die volgende week leer: Alle registers wat gebruik word in die saamsteltaal eers na die stapel gedruk word om enige aanvanklike waardes te stoor. Aan die einde van die funksie word die registers dan weer in die teenoorgestelde volgorde vanaf die stapel gehaal. Die rede dat ons dit doen is omdat die C-samesteller moontlik reeds hierdie registers gebruik het en ons nie die waardes wat in hulle gestoor is wil oorskryf nie.

Regskliek op jou projek en skep 'n nuwe lêer met die naam LED.asm

Kopiëer die volgende saamstelkode na die lêer.

```
.global _switchLEDOn
.global _switchLEDOff

_switchLEDOn:
;; Switch LED on

push bc          ;; Save the original contents of the two registers
push ax          ;; we use by pushing it to the stack.
                ;; (push works with 16bit register pairs)

movw bc, #0xff07 ;; Save the 16bit address of P7's register into
                ;; 16bit register pair BC
mov a, [bc]      ;; Get the 8bit value stored at that address
                ;; and store it in register A
or a, #10000000B ;; Bitwise OR the value in A with 10000000
mov [bc], a      ;; Set the value in P7's register to the value in A

pop ax           ;; Restore the contents of the two registers we used,
pop bc           ;; by popping it from the stack.
                ;; (pop works with register pairs)
ret              ;; exit this function

_switchLEDOff:
;; Switch LED off

push bc          ;; Save the original contents of the two registers
push ax          ;; we use by pushing it to the stack.

movw bc, #0xff07 ;; Save the 16bit address of P7's register into
                ;; 16bit register pair BC
mov a, [bc]      ;; Get the 8bit value stored at that address
                ;; and store it in register A
and a, #01111111B ;; Bitwise AND the value in A with 01111111
mov [bc], a      ;; Set the value in P7's register to the value in A

pop ax           ;; Restore the contents of the two registers we used,
pop bc           ;; by popping it from the stack.
ret              ;; exit this function
```

Verander jou RSPrak.c lêer deur die twee LED-funksies te vervang met:

```
//tell the compiler we defined these functions in another file
extern switchLEDon();
extern switchLEDoﬀ();
```

Hardloop die C-samesteller en programmeer die hardware. As alles goed is sal die LED weer flikker.

7 Huiswerk Vrae

Antwoord die volgende vrae so goed as wat jy kan en handig dit in op `learn.sun.ac.za`. Dit moet as 'n PDF-lêer gestoor wees. Die afsnydatum en -tyd is 23:55 op 6 Augustus 2015. As jy hierdie vrae kan antwoord sal jy volgende week se prakties kan voltooi.

Jy sal waarskynlik moet verwys na hoofstuk 1 van die **Sagteware Handleiding** om al die vrae te kan antwoord. Jy kan dit vind op `learn.sun.ac.za` onder “Dokumentasie”.

1. Verander LED.asm om slegs een funksie te bevat wat die LED se toestand sal omkeer (toggle). In ander woorde, dit sal aanskakel as dit af is, en afskakel as dit aan is. Om dit te doen kan jy `_switchLED` kopiëer, die naam verander na `_toggleLED`, en dan slegs die lyn wat sê “or a, #10000000B” verander.

Register P7 het 8 bisse. Ons wil slegs die hoogste bis se toestand omswaai, sonder om die ander bisse te verander. Die twee funksies om die LED aan en af te skakel het dit ook in ag geneem.

Wenk: Kyk terug na vorige semester se Rekenaarstelsels en boolse bewerkings. AND, OR, ...? Watter een sal doen wat ons wil hê? (Sien “Wenk vir huiswerk 1.txt”)

Jy kan slegs die nuwe lyn gee as antwoord op hierdie vraag.

2.
 - a) Watter tipe argitektuur gebruik die RL78 (Harvard, Von Neumann, ...)?
 - b) Wat beteken dit ten opsigte van die geheuspasie, hoeveelheid sisteem busse, en adressering van geheue op die RL78?
 - c) Afdeling 2.2.1 in die Sagteware Handleiding verwys na “mirror areas”. Is jou antwoord in 2a steeds geldig?
3.
 - a) Gebruik die RL78 'n gereduseerde instruksiestel (RISC) of 'n komplekse intruksiestel (CISC)?
 - b) Wat beteken dit ten opsigte van die aantal kloksiklusse wat elke ASM instruksie neem?
 - c) Vir die volgende lys verwerkers, sê of hulle RISC of CISC instruksiestelle gebruik:
 - i. AMD verwerkers
 - ii. Intel verwerkers
 - iii. ARM verwerkers
4. Wat is die minimum aantal kloksiklusse wat een instruksie op RL78 sal neem?
5.
 - a) Hoeveel registers het die RL78 wat ons kan gebruik as tydelike stoorplek tydens berekeninge?
 - b) Hoekom sal 'n mens eerder registers gebruik vir die stoor van waardes tydens berekeninge, en nie RAM nie?
6. Om hierdie vraag te antwoord gaan jy moet opsoek wat “little endian” en “big endian” is. Let wel, dis “**E**ndian”, nie “**I**ndian” nie. Vir 'n goeie verduideliking lees “On Holy Wars and a Plea for Peace”, wat beskikbaar is op `learn.sun.ac.za` by hierdie week se prakties.
 - a) Is die RL78 “little endian” of “big endian”?
 - b) Is jou Windows rekenaar “little” of “big endian”.

- c) Gee 'n voorbeeld waar “big endian” die logiese een sal wees om te gebruik.
7. Indien die RL78 ASM instruksies het vir die volgende, wat is dit?
- a) tel twee getalle by mekaar
 - b) trek een getal van 'n ander af
 - c) vermenigvuldig twee getalle
 - d) deel een getal in 'n ander in
 - e) Indien die RL78 nie 'n instruksie het vir enige van hierdie nie, hoe sal 'n mens dit met die bestaande instruksies doen?
8. Wat is die presiese naam van die verwerker-kern wat die RL78 ontwikkelingsbordjie wat ons gebruik op het?
9. a) Volgens die Hardware Handleiding moet bit 7 in register P7 na 1 of 0 gestel word om Poort 7, Pin 7 hoog of laag te trek. Die handleiding spesifiseer dat die geheue adres van register P7 FFF07H is (bladsy 138). In die LED.asm lêer gebruik ons die adres FF07H. Wat het geword van die hoogste 4 bits van die adres?
- Daar kan 'n aantal verduidelikings hiervoor wees, maar as presies gekyk word wat die kode doen staan afdeling 4.2.7 uit. Let op watter register ons gebruik het om die adres in te stoor, en waarom afdeling 4.2.6 se verduideliking nie geldig is nie.
- b) Waarom is dit noodsaaklik om eers P7 se waarde in register A te stoor voordat ons die AND en OR daarop toepas? Kan ons nie direk P7 gebruik om die bewerking te doen nie? (Die antwoord skuil in die lys van beskikbare formate waarop AND en OR gebruik kan word. Bladsy 47 en 48 van die Sagteware Handleiding.)