

Toetsvoorblad

Naam Student: _____

Studentnummer: _____

DE HAAGSE HOOGESCHOOL

FACULTEIT TECHNOLOGIE,
INNOVATIE & SAMENLEVING

Locatie: Delft

Opleiding: Elektrotechniek	Toetsnaam: INLMIC (proeftoets)
Opsteller: J.E.J. op den Brouw Tweede lezer: B. Kuiper	Datum: 1 januari 1970 Tijd: 0:00 – 1:30
Groep: EQ1, EQ3D Cursuscode: E-INLMIC-co1	Aantal bladzijden: 17 (inclusief voorblad) Aantal vragen: 24

Bij deze toets worden verstrekt:

- | | |
|---|--|
| <input checked="" type="checkbox"/> Gelineeerd papier | <input type="checkbox"/> Opgavenbladen met ruimte om de vragen te beantwoorden |
| <input type="checkbox"/> Ruitjes papier | <input checked="" type="checkbox"/> Antwoordformulier ABCDE |
| <input checked="" type="checkbox"/> Kladpapier | <input type="checkbox"/> Antwoordformulier Ja/Nee |
| <input type="checkbox"/> Omslag voor gemaakt tentamen | <input type="checkbox"/> Antwoordformulier Ja/Nee/Vraagteken |
| <input type="checkbox"/> Overig: _____ | |
| <input type="checkbox"/> Bijlage(n): _____ | |

Toegepaste eigen hulpmiddelen bij het maken van deze toets:

- | | |
|---|--|
| <input checked="" type="checkbox"/> Eenvoudige rekenmachine | <input checked="" type="checkbox"/> Tekenbenodigdheden (liniaal, passer) |
| <input checked="" type="checkbox"/> Grafische rekenmachine | <input checked="" type="checkbox"/> Eigen aantekeningen: zie Opmerkingen |
| <input type="checkbox"/> Computer | <input checked="" type="checkbox"/> Boeken/dictaten: zie Opmerkingen |
| <input type="checkbox"/> Formuleblad(en): _____ | |

Opmerkingen:

Aantekeningen, boeken, afdrucken van de PowerPoint-slides

Cesuur (voorlopig):

Beoordeling tentamen: Elk goed antwoord levert 90/24 punten op, in totaal zijn 90 punten te behalen.
Eindcijfer = $1 + (\text{aantal behaalde punten} / 10)$

In te leveren door surveillant bij het faculteitsbureau:

- | |
|---|
| <input checked="" type="checkbox"/> Alle documenten voorzien van naam en studentnummer, per document gesorteerd |
| <input type="checkbox"/> Alle documenten voorzien van naam en studentnummer, per student gesorteerd (in omslag) |

Belangrijk:


Voor dit tentamen gelden de regels uit de toetsregeling van De Onderwijs- en Examenregeling. Dit document is aanwezig in het toetslokaal;
Je dient zelf te controleren of je alle pagina's en vragen van dit tentamen hebt ontvangen;
Dit tentamen is dubbelzijdig geprint;
Schrijf je naam en studentnummer op alle documenten.

- Vul bij StudentNummer je studentnummer in met met zwarte of blauwe pen.
- Vul ook in de blokjes eronder je StudentNummer in.
- Vul bij de vragen het antwoord in met zwarte of blauwe pen.
- Vul bij Tentamen de naam van het vak waarover de toets gehouden wordt.
- Vul bij Naam je naam en voorletters in.
- Bij dag / maand / jaar de datum van de toets.
- Vul het versienummer in (indien van toepassing).
- Eventuele opmerkingen kun je onderaan de pagina vermelden.

Opmerkingen:

niet inkleuren a.u.b.

14557



Uitwerkingen INLMIC (proeftoets)

Opgave 1

R18 heeft voor het uitvoeren van onderstaande instructie de binaire waarde 0011 0110. Wat is het resultaat na het uitvoeren van de volgende instructie?

```
ori r18, 0xA4
```

- a) R18 bevat het decimale getal 64.
- b) R18 bevat het decimale getal 22.
- c) R18 bevat het hexadecimale getal B6.**
- d) R18 bevat het hexadecimale getal FC.

Opgave 2

Een gebruiker wil van R25 bit 7 en 6 op één zetten en bit 3 en 2 op nul. Hiervoor zijn een AND- en een OR-masker nodig. Welke waarden zijn correct?

- a) AND = 0x3F, OR = 0xC0
- b) AND = 0xF3, OR = 0xC0**
- c) AND = 0xC0, OR = 0xF3
- d) AND = 0xF3, OR = 0x0C

Opgave 3

De pinnen van Port D zijn geconfigureerd als inputs, de pinnen van Port B zijn geconfigureerd als outputs. Op de pinnen van Port D wordt de waarde 0x34 gezet. Wat is de waarde die op de pinnen van Port B staat na uitvoer van de volgende instructies:

```
in  r20, PIND
lsl  r20
lsl  r20
out  PORTB, r20
```

- a) 0xF4
- b) 0xE4
- c) 0xE8
- d) 0xD0**

Opgave 4

De inhoud van R1 is na het uitvoeren van de instructie:

```
eor r1, r1
```

- a) Verdubbeld.
- b) Geïnverteerd.
- c) Nul.**
- d) Gehalveerd.

Opgave 5

De individuele pinnen van Port D kunnen als ingang of uitgang geconfigureerd worden. De eenvoudige bits van Port D worden PD7, PD6, PD5, PD4, PD3, PD2, PD1, PD0 genoemd. Welk van

de volgende alternatieven configureert PD0, PD1, PD6 en PD7 als uitgang en de overige pinnen als ingang?

- a) `out DDRD,0x3c`
- b) `ldi r20,0xc3`
`out DDRD,r20`**
- c) `out PORTD,0x3c`
- d) `ldi r20,195`
`PORTD,r20`

Opgave 6

Welke bit(s) in het statusregister SREG zijn gezet na uitvoeren van de volgende instructies, als vooraf alle bits in het SREG op '0' geïnitieerd zijn?

```
ldi r16,0x4f
ldi r17,0x3f
add r16,r17
```

- a) geen van de bits
- b) het Z-bit
- c) het N-bit
- d) het N-bit en V-bit**

Opgave 7

Bij een nieuwe microcontroller zijn de ontwerpers vergeten een ADC-instructie te maken. Er bestaat alleen een ADD-instructie die de carry niet meeneemt in de optelling maar wel wordt de carrybit gezet in het statusregister SREG gezet indien er een overloop is. Hieronder volgen twee stukjes code met 16-bit optellingen waarbij de carry toch wordt opgeteld. Ga ervan uit dat de 16-bit optelling als geheel geen overloop geeft en ga uit van unsigned getallen. Welke stukje(s) is/zijn correct?

<pre> ; code I add r20,r24 brcc over1 subi r21,-1 over1: add r21,r25 ...</pre>	<pre> ; code II add r20,r24 add r21,r25 brcc over2 subi r21,-1 over2: ...</pre>
--	---

- a) Zowel code I als code II zijn onjuist
- b) Code I is onjuist en code II is juist.
- c) Code I is juist en code II is onjuist**
- d) Zowel code I als code II zijn juist.

Opgave 8

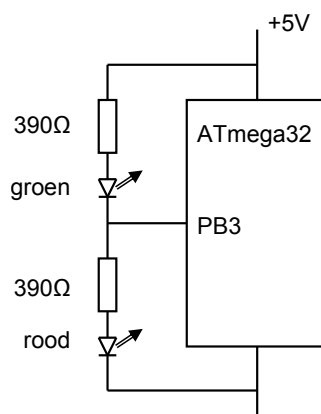
Pin 1 van port C is momenteel ingesteld als uitgang met als waarde 0, deze pin ligt dus hard aan de 0 volt. Met welke instructie-reeks kan je hier veilig een pull-up van maken?

- a) `cbi DDRC,1`
`sbi PORTC,1`**

- b) `cbi DDRC,1`
`sbi PINC,1`
- c) `sbi PORTC,1`
`cbi DDRC,1`
- d) `sbi PINC,1`
`cbi DDRC,1`

Opgave 9

Een ontwerper wil graag afwisselend een rode en groene led laten branden. Hiervoor heeft zij onderstaande oplossing bedacht. Pin 3 van Port B wordt hiervoor gebruikt. Welke van de onderstaande mogelijkheden zorgt er voor dat de groene led gaat branden?



- a) `sbi DDRB,3`
`cbi PORTB,3`
- b) `sbi DDRB,3`
`sbi PORTB,3`
- c) `cbi DDRB,3`
`cbi PORTB,3`
- d) `cbi DDRB,3`
`sbi PORTB,3`

Opgave 10

Een ATmega32 microcontroller is aangesloten op een 8 MHz kristal. Als we met onderstaand programma een wachttijd van 100 milliseconde willen realiseren op welke waarden moeten de registers `delay1`, `delay2` en `delay3` dan worden geïnitieerd?

```

loop: subi delay1,1
      sbci delay2,0
      sbci delay3,0
      brne loop

```

- a) Delay1 = 0x30, Delay2 = 4B, Delay3 = 04
- b) Delay1 = 0x00, Delay2 = 71, Delay3 = 02**
- c) Delay1 = 0xff, Delay2 = 34, Delay3 = 0B

- d) Anders dan opgegeven in a, b of c.

Opgave 11

Wat is de maximale wachttijd die met het programma van de vorige opgave kan worden gerealiseerd?

- a) 4,1 seconden
- b) 21,0 seconden
- c) 6,7 seconden
- d) **10,5 seconden**

Opgave 12

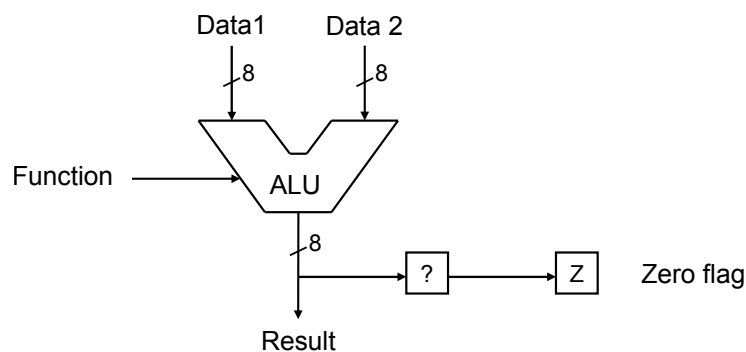
Bekijk de volgende code. Welke uitspraak is correct?

```
ldi r16,0b01100000
out GICR,r16
ldi r16,0b00001100
out MCUCR,r16
```

- a) **INT0 is geactiveerd.**
- b) INT1 reageert op beide flanken
- c) INT0 wordt geactiveerd als flankgevoelige interrupt.
- d) INT1 is geactiveerd en reageert op een opgaande flank

Opgave 13

De uitkomst van een rekenkundige bewerking kan nul opleveren, dus de uitgangen van de ALU zijn alle 0. De Z-flag van het Status Register wordt dan 1. Dit kan geregeld worden met een logische functie. Zie onderstaande figuur. Welke logische functie is dit (er zijn geen geïnverteerde signalen beschikbaar)?



- a) AND
- b) EXOR
- c) **NOR**
- d) NAND

Opgave 14

Bekijk de volgende code

```
ldi r16,0b00011011
out TCCR0,r16
```

- a) T/C0 staat in de normal mode met prescaler op 1024
- b) T/C0 staat in de CTC-mode met prescaler 64 en OC0 op Toggle On Compare Match**
- c) T/C0 staat in de normal mode met prescaler 256 en OC0 op Set On Compare Match
- d) T/C0 staat anders ingesteld dan aangegeven in a, b of c.

Opgave 15

Een ATmega32 wordt geklokt op 7,3728 MHz. T/C0 moet elke 16,67 ms een OCM-interrupt genereren. De waarde voor de prescaler en OCR0 zijn dan:

- a) prescaler = 256, OCR0 = 480
- b) prescaler = 1024, OCR0 = 119**
- c) prescaler = 64, OCR0 = 100
- d) prescaler = 1024, OCR0 = 255

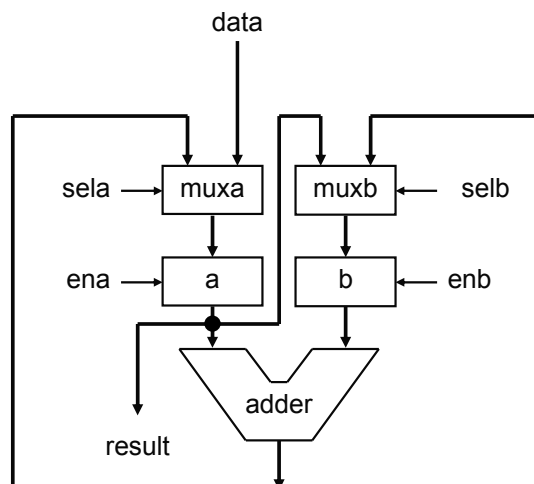
Opgave 16

Een ATmega32 loopt op 3,579545 MHz. Wat bedraagt de tijd tussen twee overflowinterrupts van T/C0 als prescalerwaarde 256 wordt gebruikt?

- a) 18,3 ms**
- b) 10,1 ms
- c) 54,6 ms
- d) 23,5 ms

Opgave 17

Hieronder is het blokschema van een eenvoudige processor te vinden. De registers werken op hetzelfde kloksignaal. De inhouden van A en B zijn onbekend. Op de ingang 'data' wordt continu het getal 5 aangeboden. De gebruiker stuurt zelf de besturingssignalen aan (muxa...).



Ontwerp nu een 'programma' dat met een minimum aan 'instructies' (aansturen van de besturingssignalen) de waarde 35 in register B plaatst. Elke instructie kost één klokpuls. Dit kost dan minimaal:

- a) 4 klokpulsen
- b) 5 klokpulsen
- c) 6 klokpulsen**
- d) 7 klokpulsen

Opgave 18

De ATmega32 microcontroller heeft 21 mogelijke interrupts. Als twee interrupts gelijktijdig optreden, hoe verloopt dan de afhandeling van de interrupts?

- a) Een interne interrupt wordt als eerste afgehandeld.
- b) Een externe interrupt wordt als eerste afgehandeld.
- c) De interrupt met het hoogste adres in de vectortabel wordt als eerste afgehandeld.
- d) De interrupt met het laagste adres in de vectortabel wordt als eerste afgehandeld.**

Opgave 19

Gegeven de volgende code:

```
; include not needed since AVR Studio 6
;.include "m32def.inc"
.org 0x000
    ldi r16,low(RAMEND)
    out SPL,r16
    ldi r16,high(RAMEND)
    out SPH,r16
    ldi r19,0xff
    ldi r18,13
    call multi
halt:  rjmp halt

multi:  push r19
        push r18
        lsl r18
        lsl r18
        pop r19
        add r18,r19
        lsl r18
        pop r19
        ret
```

Er worden drie uitspraken gedaan:

- I. De instructie `lsl` schuift de inhoud van een register naar rechts.
- II. Na het uitvoeren van `multi` is R18 vermenigvuldigd met 10.
- III. R19 heeft na het uitvoeren van de eerste `pop`-instructie de waarde 0xff.

Welke van de onderstaande uitspraken is correct?

- a) I is niet waar en II is waar**

- b) II is waar en III is waar
- c) I is waar en II en III zijn niet waar
- d) I is niet waar en III is waar

Opgave 20

Welke van de volgende uitspraken is juist/onjuist?

- I. De vlaggen kunnen geladen worden in R16.
 - II. DDRC kan in R16 worden geladen d.m.v. een lds-instructie.
- a) Zowel I als II zijn onjuist
 - b) I is onjuist en II is juist
 - c) I is juist en II is onjuist
 - d) **Zowel I als II zijn juist.**

Opgave 21

Voor een register kan een symbolische naam gedefinieerd worden. We kunnen register R20 bijvoorbeeld de symbolische naam “tentamen” geven. Wat is de correcte wijze waarop dit in een assembler-programma opgegeven kan worden?

- a) `.tentamen =R20`
- b) `.db R20 =tentamen`
- c) `.def tentamen =R20`
- d) `.equ tentamen =R20`

Opgave 22

Bij het aanroepen van een subroutine wordt het terugkeeradres op de stack opgeslagen. Dit terugkeeradres bestaat bij de ATmega32 uit twee bytes. Het lage byte van het terugkeeradres wordt eerst op de stack gezet, het hoge byte als tweede. Een gebruiker heeft onderstaand programma geschreven.

```
; include not needed since AVR Studio 6
;.include "m32def.inc"
.org 0x000
        ldi r16,low(RAMEND)
        out SPL,r16
        ldi r16,high(RAMEND)
        out SPH,r16
        call rettrick
loop:    rjmp loop

rettrick: pop r16
        pop r16
        ldi r16,0x01
        push r16
        ldi r16,0x05
        push r16
        ret

.org 0x0105
```

```
halt1: rjmp halt1

.org 0x0501
halt2: rjmp halt2
```

Na uitvoeren van de `ret`-instructie wordt gesprongen naar de instructie direct achter het label

- a) loop
- b) retrick
- c) halt1
- d) halt2**

Hint: teken de stack tijdens uitvoering van de `CALL`-instructie en de routine `retrick`.

Opgave 23

Bestudeer het programma uit vraag [22](#) aandachtig. Het Flash-ROM adres waar routine `retrick` begint is (in words):

- a) 0x006
- b) 0x007**
- c) 0x008
- d) 0x009

Hint: slides geven aanwijzingen over de grootte van instructies.

Opgave 24

Bestudeer het programma uit vraag [22](#) aandachtig. De stackpointer wordt geïnitieerd 0x085f. Wat is het laagste adres dat de stackpointer aanwijst tijdens het draaien van deze code?

- a) 0x085d**
- b) 0x085c
- c) 0x0860
- d) 0x085f

*_*_*_*_*_*_*_* einde toets *_*_*_*_*_*_*_*

Uitwerking opgaven

Opgave 1. R18 heeft de binaire waarde 0011 0110. Dit moet ge-OR-d worden met 0xA4, binair is dat 1010 0100.

0011 0110		0x36	54
1010 0100	or	0xA4	164

1011 0110		0xB6	182

Het resultaat is 182 (decimaal) of 0xB6 (hexadecimaal). Het antwoord is [c](#). Zie boek blz 177, slides week 3, pag 41.

Opgave 2. Hier wordt een combinatie van zetten en wissen van bits gevraagd. We noteren eerst wat er moet gebeuren: een 1 is zetten, een 0 is wissen en een - betekent ongewijzigd laten. Een OR levert 1-en en een AND levert 0-en.

7654 3210	bitnummers
11-- 00--	wat moet er gebeuren
1100 0000	OR-masker (1 is zetten, 0 is ongemoeid laten)
1111 0011	AND-masker (0 is wissen, 1 is ongemoeid laten)

Het resultaat is dus AND = 0xF3 en OR = 0xC0. Het antwoord is [b](#). Zie boek blz 176 – 177, slides week 3 pag 40 – 43.

Opgave 3. Op de pinnen van PORTD wordt de waarde 0x34 gezet. Dit is binair 0011 0100. In onderstaande code wordt dit patroon eerst ingelezen in R20. Daarna wordt er twee keer linksom geschoven. Zie onderstaande code en commentaar.

in r20,PIND	R20 = 0011 0100
lsl r20	R20 = 0110 1000
lsl r20	R20 = 1101 0000
out PORTB,r20	

Het resultaat is 0xD0. Het antwoord is [d](#). Zie boek blz 186, blz 64 – 67, slides week 3 pag 5.

Opgave 4. Bij de instructie eor ra,rb worden de bits van register a en register b bitsgewijs ge-exor-d. Hieronder is de bekende tabel gegeven (n staat voor bitnummer 0 t/m 7).

ra_n	rb_n	F_n
0	0	0
0	1	1
1	0	1
1	1	0

Nu wordt gevraagd wat de het resultaat is van de instructie `eor r1,r1`. Hierin wordt R1 ge-exor-d met zichzelf! Dus als bit 0 van R1 een 0 is wordt dit bit ge-exor-d met 0. En dat levert als resultaat een 0. Precies hetzelfde resultaat wordt verkregen als het bit een 1 is. Met andere woorden: alle bits worden 0! Alleen de eerste en laatste regel van de tabel zijn van toepassing. Dit is een bekende truc om een register op 0 te krijgen als een processor geen `clr`-instructie (clear) heeft. Het antwoord is **c**. Zie boek blz 177 – 178, en blz 708 (`clr`-instructie).

Opgave 5. Willen we poortpinnen als uitgang gebruiken dan moet het DDR-register worden geprogrammeerd met enen op de bitposities van de pinnen die als uitgang moeten dienen. Aangezien de pinnen 0, 1, 6 en 7 als uitgang moeten worden geprogrammeerd wordt het bitpatroon 1100.0011 en dit is gelijk aan 0xC3. I/O-registers programmeer je met een `out`-instructie, maar dat kan alleen vanuit een general purpose register, niet direct met een constante, dus je hebt (bijvoorbeeld) een `ldi`-instructie nodig om een register met een constante te laden. Zie boek blz 142 – 144, slides week 3 pag 28 – 29.

Het antwoord is **b**.

Opgave 6. De `add`-instructie telt twee registers bij elkaar op waarbij de vlaggen worden aangepast en het resultaat wordt opgeslagen.

```
ldi r16,0x4f    R16 = 0x4f
ldi r17,0x3f    R17 = 0x3f
add r16,r17     R16 = 0x4f + 0x3f = 0x8e (> 0x7f dus overflow!)
```

Nu is $0x4f + 0x3d = 0x8e > 0x7f$, dus past niet in een 2's complement 8-bit register, daarom wordt de V-flag op 1 gezet. Het msb is 1 dus de N-flag wordt ook op 1 gezet. Beschouw je de inhoud van de registers als unsigned, dat past het resultaat wel, dus de C-flag is 0. De Z-flag is natuurlijk 0. Zie boek blz 162, blz 170 – 175, blz 72 – 73, slides week 3 pag 18.

Het antwoord is **d**.

Opgave 7. De microcontroller heeft geen `adc`-instructie maar die kunnen we wel nabootsen (bij multibyte optellingen > 2 bytes wordt het lastig!). De truc is om eerst de minst significante bytes op te tellen, daaruit volgt een carry (die kan 0 of 1 zijn). Als de carry 1 is, moet er bij de meest significante bytes één worden opgeteld (dat is immers de werking van de carry). Dus als de carry 1 is verhogen we de meest significante bytes met één en anders niet. Dit is een werkje voor de `brcc`-instructie. Als laatste moeten de twee meest significante bytes worden opgeteld. De eventuele carry die daaruit volgt, verwaarlozen we. Als je de code bekijkt, zie je dat er twee register-“paren” zijn: R21 met R20 en R25 met R24.

Code I voldoet. Als eerste wordt een `add`-instructie uitgevoerd die R20 en R24 bij elkaar optelt. Als de carry 0 is, wordt over de `subi`-instructie gesprongen. Is de carry 1, dan “springt” de `brcc`-instructie niet en wordt de `subi`-instructie uitgevoerd, die 1 bij R21 optelt. Daarna wordt in beide gevallen R25 bij R21 opgeteld.

Code II werkt niet correct. Immers de tweede `add`-instructie verknoeit de waarden van de vlaggen die uit de eerste `add`-instructie zijn voortgekomen, dus ook de carry-flag.

Zie boek blz 163, slides week 2 pag 15. Het antwoord is **c**.

Opgave 8. Zie slides week 3, pagina 36, derde item. DDR eerst naar 0, dan PORT naar 1. De cbi- en sbi-instructies worden uitgelegd in het boek blz 149 – 152, zie ook boek blz 142 – 144, slides wk3 pag 39. Het antwoord is [a](#).

Opgave 9. Dit is een grappige schakeling waarbij je twee leds kan aansturen als ze niet tegelijkertijd moeten branden. Aangezien een poortpin van de ATmega32 zowel 20 mA kan leveren als opnemen, geeft dit geen problemen. Om de groene led te laten branden moet PB3 natuurlijk als een harde 0 geprogrammeerd zijn, dus DDRB3 = 1 en PORTB3 = 0. Zie boek blz 142 – 144, blz 149 – 152, slides week 3 pag 29.

Nog veel leuker is dat PB3 ook gebruikt kan worden in de Output Compare Match modus, die je kan laten toggelen. Daarmee kan je de beide leds (ogenschijnlijk) op halve intensiteit laten branden. In de PWM-modi kan je nog veel leukere dingen doen. Zie slides week 5, pag 30 – 34.

Het antwoord is [a](#).

Opgave 10. Hiervoor moet je de formule gebruiken voor het bepalen van het aantal keer dat de lus doorlopen moet worden, zie slides week 4, pag 9.

$$aantal_lussen = \frac{f_{cpu} \cdot wachttijd}{klolpulsen/lus} = \frac{8000000 \cdot 0,1}{5} = 160000$$

Het getal dat gebruik moet worden is 160000 en dat is 0x027100. Dan wordt delay1 = 0x00, delay2 = 0x71 en delay3 = 0x02.

Het antwoord is [b](#).

Noot: de manier waarop wachtlussen in het boek worden uitgelegd is niet handig, dat kost veel rekenwerk. Daarnaast levert het veel instructies op en is lastig te tunen.

Opgave 11. Om een wachtlus maximaal tijd te laten verstoken, moet je het lus-aantal maximaal zien te krijgen. Daarvoor moeten delay1, delay2 en delay3 geladen worden met 0x00, hoger kan niet. Dit kan omdat er eerst verlaagd wordt en dan pas getest op 0. Het aantal keer dat de lus dan doorlopen wordt is 16777216 keer. Nu verbouwen we de de formule uit de slides van week 4.

$$aantal_lussen = \frac{f_{cpu} \cdot wachttijd}{klolpulsen/lus} \iff wachttijd = \frac{aantal_lussen \cdot klolpulsen/lus}{f_{cpu}}$$
$$wachttijd = \frac{16777216 \cdot 5}{8000000} = 10,485... \approx 10,5 \text{ s}$$

Het antwoord is [d](#). Zie slides week 4, pag 9.

Opgave 12. Zoek eerst de registers en de betekenissen op in het boek blz 376 – 380 of de slides week 5, pagina 16 – 20. Hieronder de code met commentaar.

```
ldi r16,0b01100000    INT0 en INT2 actief
out GICR,r16
ldi r16,0b00001100    INT1 op flank, INT0 op level
out MCUCR,r16
```

A is correct want INT0 wordt geactiveerd. B is niet correct want INT1 is ingesteld op opgaande flank (en reageert helemaal niet, wat het staat uit). C is niet correct want INT0 is geactiveerd als niveau-interrupt. D is niet correct want INT1 is niet geactiveerd. Het antwoord is **a**.

Opgave 13. De Z-flag kan eenvoudig worden bepaald door alle uitkomstbits in één NOR-poort te stoppen. De formule is dus:

$$Z = \overline{R7 + R6 + R5 + R4 + R3 + R2 + R1 + R0}$$

Je kan het ook anders definiëren: Z wordt 1 als alle uitkomstbits 0 zijn. Dat is in formulevorm:

$$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$$

Maar hiervoor heb je de geïnverteerde uitgangen nodig. Met behulp van De Morgan kan je hieruit naar de bovenste formule omwerken. Zie boek blz 71, slides week 3, pag 18.

Het antwoord is **c**.

Opgave 14. Bekijk de volgende code

```
ldi r16,0b00011011    T/C0 in CTC mode, prescaler op 64
out TCCR0,r16           maar ook OCM-mode op Toggle on Compare Match
```

Zie boek blz 315, slides week 5 pag 31 – 34. Het antwoord is **b**.

Opgave 15. Deze vraag kan worden opgelost met de formule uit de slides week 5, pag 40. Hiervoor moet de frequentie van het OCM-interrupt signaal worden uitgerekend, want het staat als tijd genoteerd. Een periodetijd van 16,67 ms is gelijk aan een frequentie van 60 Hz (frequentie van het lichtnet in Amerika). De prescaler moet op 1024, de andere mogelijkheden leveren geen goede waarden voor N en OCR0.

$$prescaler \cdot N = \frac{f_{clk}}{f_{ocm}} \rightarrow N = \frac{f_{clk}}{f_{ocm} \cdot prescaler} = \frac{7372800}{60 \cdot 1024} = 120$$

Let erop dat OCR0 119 moet zijn, want de timer telt van 0 t/m 119. Het antwoord is **b**. Zie ook boek blz 328 – 330, blz 374 – 375.

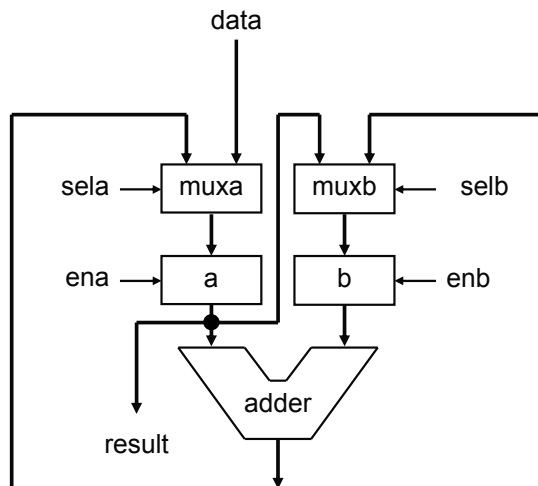
Opgave 16. Ook hier wordt een tijd gevraagd, terwijl de formule van frequentie uitgaat. De frequentie van het kristal is erg nauwkeurig gegeven en je kan deze ook echt kopen!

$$f_o = \frac{\frac{f_{clk}}{prescaler}}{256} = \frac{f_{clk}}{prescaler \cdot 256} = 54,6$$

De frequentie is 54,6 Hz, dus de periodetijd is 18,3 ms. Het antwoord is **a**. Zie boek blz 369, slides week 5 pag 38.

Opgave 17. Hieronder is wederom het blokschema van een eenvoudige processor te vinden. De registers werken op hetzelfde kloksignaal. De inhouden van A en B zijn onbekend. Op de ingang data wordt continue het getal 5 aangeboden. Er zijn twee klokpulsen (“slagen”) nodig om beide

registers met 5 geladen te hebben, immers alleen A kan data van buitenaf ontvangen. Daarna is het een kwestie van smaak. Hieronder zie je een mogelijkheid, er zijn ook andere. Je moet goed opletten dat jouw mogelijk ook echt kan. Het is bijvoorbeeld niet mogelijk om de inhoud van register B in register A te laden, omdat daar geen hardwarevoorziening voor is. Na de 6^e klokpuls heeft register B de waarde 35.



Klokpuls	A	B
0	?	?
1	5	?
2	5	5
3	5	10
4	15*	15*
5	5**	30
6	5	35

* A en B tegelijk laden.

** A wordt weer geladen met 5.

Er zijn andere mogelijkheden.

Het antwoord is **c**. Zie slides week 1, pag 37 – 38.

Opgave 18. Als tegelijk twee of meerdere interrupts worden aangevraagd, wordt degene met het laagste adres in de vectortabel als eerste uitgevoerd. Zie boek blz 381, blz 365, slides week 5, pag 13.

Het antwoord is **d**.

Opgave 19. We bekijken alleen subroutine multi, de rest is niet van belang. Hieronder nog de code. Achter de code ook gelijk commentaar.

```

1  multi:  push r19          ; R19 later nodig
2          push r18         ; R18 even saven, later weer nodig
3          lsl r18          ; Hierna is R18 = R18*2
4          lsl r18          ; Hierna is R18 = R18*2*2 (= R18*4)
5          pop r19          ; R19 laden met oude waarde R18 (let op pushes!)
6          add r18,r19       ; Hierna is R18 = R18*2*2 + R18 (= R18*5)
7          lsl r18          ; Hierna is R18 = (R18*2*2 + R18)*2 (= R18*10)
8          pop r19          ; Nu nog even R19 ophalen (eerste push)
9          ret              ; En terugkeren

```

Zoals je ziet vermenigvuldigt deze routine R18 met 10. De truc is om gebruik te maken van een combinatie van schuiven (vermenigvuldigen met 2) en optellen. Eerst wordt R19 gepusht (regel 1) op stack omdat we R19 nodig hebben voor de optelling (regel 5 en 6). Dan wordt R18 nog eens gepusht (regel 2) omdat later die waarde nodig is voor de optelling via R19 (regels 5 en 6). Nu wordt R18 twee keer geschoven wat een vermenigvuldiging met 4 betekent (regels 3 en 4). Dan moet er één keer de originele waarde van R18 bij worden opgeteld om een vermenigvuldiging met 5 te krijgen. R18 is natuurlijk allang veranderd dus we moeten een tweede register gebruiken. Dit is R19. R19 popt nu de stack (en daar werd als laatste de originele waarde van R18 opgezet! Zie slides week 4 pag 25) en telt dit op bij R18 (regels 5 en 7). Dan wordt er nog één keer geschoven en heb je als resultaat dat R18 vermenigvuldigd is met 10!

Bij de code van opgave 19 worden weer drie uitspraken gegeven:

I. De instructie `lsl` schuift de inhoud van een register naar rechts.

II. Na het uitvoeren van `multi` is R18 vermenigvuldigd met 10.

III. R19 heeft na het uitvoeren van de eerste `pop`-instructie de waarde 0xff.

I is niet waar: `lsl` schuift de inhoud naar links. Zie boek, blz 186.

II is waar: zie het verhaal hierboven.

III is niet waar: laatste `push` was die van R18 en de eerste `pop` is die van R19, m.a.w. de (originele) waarde van R18 komt in R19 (zie slides week 4, pag 24).

Het antwoord is [a](#).

Opgave 20. Er worden twee uitspraken gedaan:

I is juist: je kan met `in r16, SREG` de vlaggen laden in register 16. Zie boek blz 64, blz 194, blz 384, slides week 5, pag 10.

II is ook juist: DDRC is dan wel een I/O-register maar die zijn gemapt in het DATA-bereik vanaf adres 0x0020. Je moet dat wel het nieuwe adres even uitrekenen, en het is niet efficiënt: de `lds`-instructie is twee words en de `in`-instructie is één word. Zie boek blz 59 – 62, blz 66, slides week 2, pag 31.

Het antwoord is [d](#).

Opgave 21.

Zie slides week 2, pag 48. Het antwoord is [c](#). Noot: in het boek wordt geen gebruik gemaakt van `.def`, in alle voorbeelden worden de namen van de registers gebruikt.

Opgave 22. Hier zie je mooi een truc om het terugkeeradres aan te passen. De truc is om in de subroutine twee keer een `pop` te doen. Hierdoor wordt het gestackte terugkeeradres "van de stack" gehaald (in feite wordt alleen de stack pointer verhoogd). Dan pushen we zelf twee waarden op de stack en voeren vervolgens een `ret`-instructie uit!

```
rettrick: pop r16           ; pop stack, high byte RET-address
          pop r16           ; pop stack, low byte RET-address
          ldi r16,0x01       ; put 0x01 on stack as low byte for RET
          push r16
          ldi r16,0x05       ; put 0x05 on stack as high byte for RET
          push r16
          ret                ; And let's return!!!!
```


Hieronder de stack tijdens uitvoering:

Stack direct ná uitvoering call:

Stack vlak vóór de uitvoering van ret:

+-----+		+-----+	
..	<- sp	..	<-- sp
+-----+		+-----+	
0x00		0x05	0x85d
+-----+		+-----+	
0x05		0x01	0x85e
+-----+		+-----+	
			0x85f

Het terugkeeradres is dus 0x0501. Het antwoord is [d](#). Zie ook boek blz 118 – 127, slides week 4, pag 24.

Opgave [23](#). Deze lijkt wat lastig, maar je komt een heel eind als je bedenkt dat de meeste instructies slechts één word beslaan. Alleen lds, sts, call en jmp-instructies beslaan twee words.

Het programma begint op adres 0x000 (zie de .org-directive), ldi en out kosten ieder één word. De call kost twee words en de rjmp (relative jump) kost één word. Alles tezamen is dit 8 words en het adres subroutine rettrick is dus 0x007 want het eerste word ligt op adres 0x000. Vergelijk het met array's in C. Een array van 4 items loopt van 0 t/m 3.

Je kan één en ander opzoeken in het boek blz 118 (call), blz 87, 91 (ldi), blz 93 (out), blz 117 (rjmp) en de slides week 3, pag 2 – 15.

Het antwoord is [b](#).

Opgave [24](#). Zie het plaatje van de stack bij het antwoord van vraag [22](#). Het laagste adres is 0x085d.

Het antwoord is [a](#).

De antwoorden op een rij:

1 c	9 a	17 c
2 b	10 b	18 d
3 d	11 d	19 a
4 c	12 a	20 d
5 b	13 c	21 c
6 d	14 b	22 d
7 c	15 b	23 b
8 a	16 a	24 a