UNIVERSITEIT VAN AMSTERDAM

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

Tentamen

Computer Systemen ***baiCOSY06***

Bachelor Kunstmatige Intelligentie

Deeltentamen

Datum: *25 september 2015*
Tijdstip: *13u-15u*

Aantal pagina's (inclusief voorblad): 12
Aantal vragen: 6

VOORDAT U BEGINT

- Controleer of uw versie van het tentamen compleet is.
- Schrijf **uw naam en studentnummer en indien van toepassing versienummer** op **elk vel papier** dat u inlevert en **nummer de pagina's**.
- Uw **jas en tas** moeten onder uw tafel liggen.
- Het is **niet toegestaan** de communicatie opties op je apparatuur te gebruiken: zet de **verbindingen** via ethernet, Bluetooth en het telefoonnetwerk **uit**!
- **Toegestane hulpmiddelen:** een grafische rekenmachine en de (elektronische) boeken behorende bij dit vak (Computer Systems, en Van 0 en 1 tot processor) is toegestaan. Ook is het toegestaan aantekeningen en samenvattingen te gebruiken.

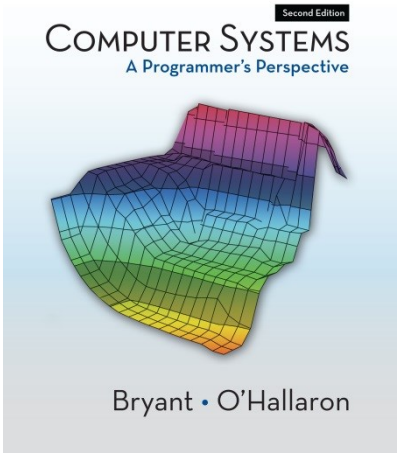
HUISHOUDELIJKE MEDEDELINGEN

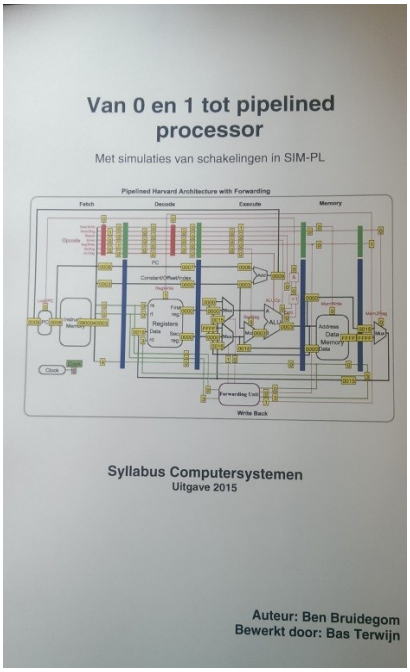
- De eerste 30 minuten en de laatste 15 minuten mag u de zaal niet verlaten, ook niet voor het bezoeken van het toilet.
- Op verzoek van de examinator (of diens vertegenwoordiger) moet u zich kunnen legitimeren met een bewijs van inschrijving of een geldig legitimatiebewijs**. Leg dit bewijs alvast klaar**.
- Tijdens het tentamen is toiletbezoek niet toegestaan, tenzij de surveillant hier toestemming voor geeft.
- 15 minuten voor het eind wordt u gewaarschuwd dat het inlevertijdstip nadert.

Succes!

Student: **Collegekaartnummer:**

De volgende boeken zijn toegestaan tijdens het open boek tentamen





Student: **Collegekaartnummer:**

Tentamen Computersystemen

baiCOSY06 2e jaar bachelor AI, 1e semester 25 september 2015, 13u-16u JWS zaal 2

vraag 1

a) Bereken met een **8 bit** *two's complement* berekening het resultaat van de expressie **100 – 10** (waar 100 en 10 decimale getallen zijn) en geef het resultaat weer in **binaire** representatie.

```
(dec)  10 = (bin) 0000 1010
        invert:   1111 0101
     plus one:          1
                ----------+
(dec) -10 = (bin) 1111 0110
```

```
carry:   111  1
 100    0110 0100
 -10    1111 0110
       ----------+
  90   0101 1010
```

b) Bereken met een ***8 bit*** *two's complement* berekening het resultaat van de expressie **-126 – 10** (waar -126 en 10 decimale getallen zijn) en geef vervolgens het resultaat weer in **decimale** representatie.

```
carry:  1     11
 -126   1000 0010
  -10   1111 0110
        ---------- +
       0111 1000
```

```
carry: 12
        8
       16
       32
       64
      ---+
      120
```

Student: **Collegekaartnummer:**

c) Bereken met een **16 *bit*** *two's complement* berekening het resultaat van de expressie **-126 – 10** (waar -126 en 10 decimale getallen zijn) en geef vervolgens het resultaat weer in **decimale** representatie.

```
carry 11111 1111        11
 -126  1111 1111 1000 0010
  -10  1111 1111 1111 0110
       ------------------------ +
       1111 1111 0111 1000
```

-256 + 120 = **-136**

d) Verklaar het verschil tussen het resultaat bij vraag b en c hierboven.

Bij vraag b treedt een overflow op omdat getallen lager dan -128 niet kunnen worden gerepreseteerd in een 8 bit two's complement representatie. Het resultaat is daar daarom 256 (2 tot de macht 8) hoger dan bij vraag c:

-136 + 256 = 120

Student: **Collegekaartnummer:**

vraag 2

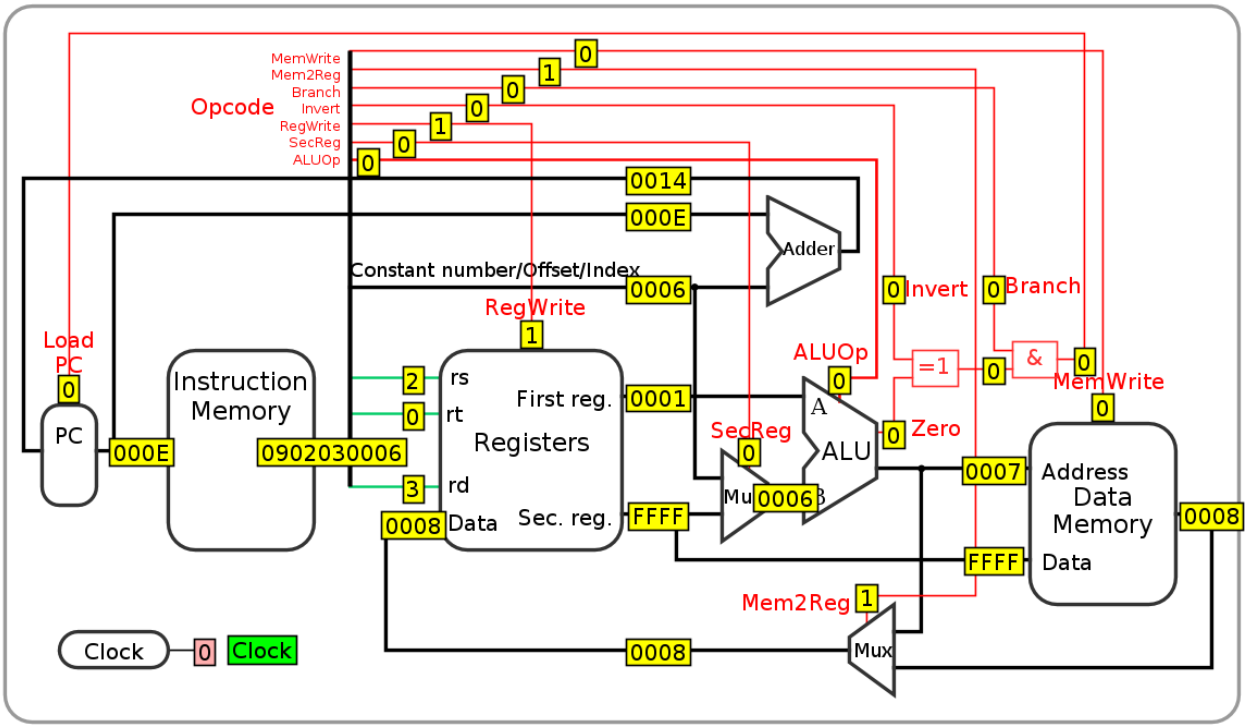
Onderstaand is de lijst met instructies van de Harvard Single Cycle processor zoals beschreven in hoofdstuk 8 van syllabus "Van 0 en 1 tot pipelined processor" en de opcode van de ALU die hierbij gebruikt wordt.

Mnemonic	Betekenis	Voorbeeld	Betekenis
ADD rd, rs, rt	Optellen registers	ADD $5, $6, $7	r5 ← r6 + r7
SUB rd, rs, rt	Aftrekken registers	SUB $5, $6, $7	r5 ← r6 – r7
AND rd, rs, rt	Bitwise AND registers	AND $5, $6, $7	r5 ← r6 & r7
OR rd, rs, rt	Bitwise OR registers	OR $5, $6, $7	r5 ← r6 | r7
XOR rd, rs, rt	Bitwise XOR registers	AND $5, $6, $7	r5 ← r6 ^ r7
SHL rd, rs, rt	Shift Left register	SHL $5, $6, $7	r5 ← r6 << r7
SHR rd, rs, rt	Shift Right register	SHR $5, $6, $7	r5 ← r6 >> r7
COPY rd, rt	Copy register	COPY $3, $2	r3 ← r2
ADDI rd, rs, imm	Optellen register en const.	ADDI $5, $6, 0x1234	r5 ← r6 + 0x1234
SUBI rd, rs, imm	Aftrekken register en const.	SUBI $7, $6, 0x1234	r7 ← r6 - 0x1234
ANDI rd, rs, imm	Bitwise AND register en const	ANDI $5, $6, 0d34	r5 ← r6 & 0d34
ORI rd, rs, imm	Bitwise OR register en const.	ORI $5, $6, 0d34	r5 ← r6 | 0d34
XORI rd, rs, imm	Bitwise XOR register en const	XORI $5, $6, 0d34	r5 ← r6 ^ 0d34
SHLI rd, rs, imm	Shift Left register	SHLI $5, $6, 5	r5 ← r6 << 5
SHRI rd, rs, imm	Shift Right register	SHRI $5, $6, 5	r5 ← r6 >> 5
LOADI rd, imm	Laad constante in register	LOADI $1, 0x 0020	r1 ← 0x0020
BZ rt, label	Branch if rt gelijk is aan 0	BZ $6, end	If (r6 == 0) goto 'end'
BNZ rt, label	Branch if rt ongelijk is aan 0	BNZ $6, end	If (r6 != 0) goto 'end'
BEQ rs, rt, label	Branch if rs gelijk is aan rt	BEQ $6, $8, loop	If (r6 == r8) goto 'loop'
BNE rs, rt, label	Branch if rs ongelijk is aan rt	BNE $6, $8, loop	If (r6 != r8) goto 'loop'
BRA label	Branch always	BRA label	PC ← PC + offset
SW rt, index, rs	Store Word to memory	SW $0, 0x1234, $1	r0 → Mem(r1 +1234_{Hex})
LW rd, index, rs	Load Word to register	LW $0, 0x1234, $1	r0 ← Mem(r1 +1234_{Hex})

ALU-code
000 +
001 -
010 AND
011 OR
100 XOR
101 SHL
110 SHR
111 B

Student: **Collegekaartnummer:**

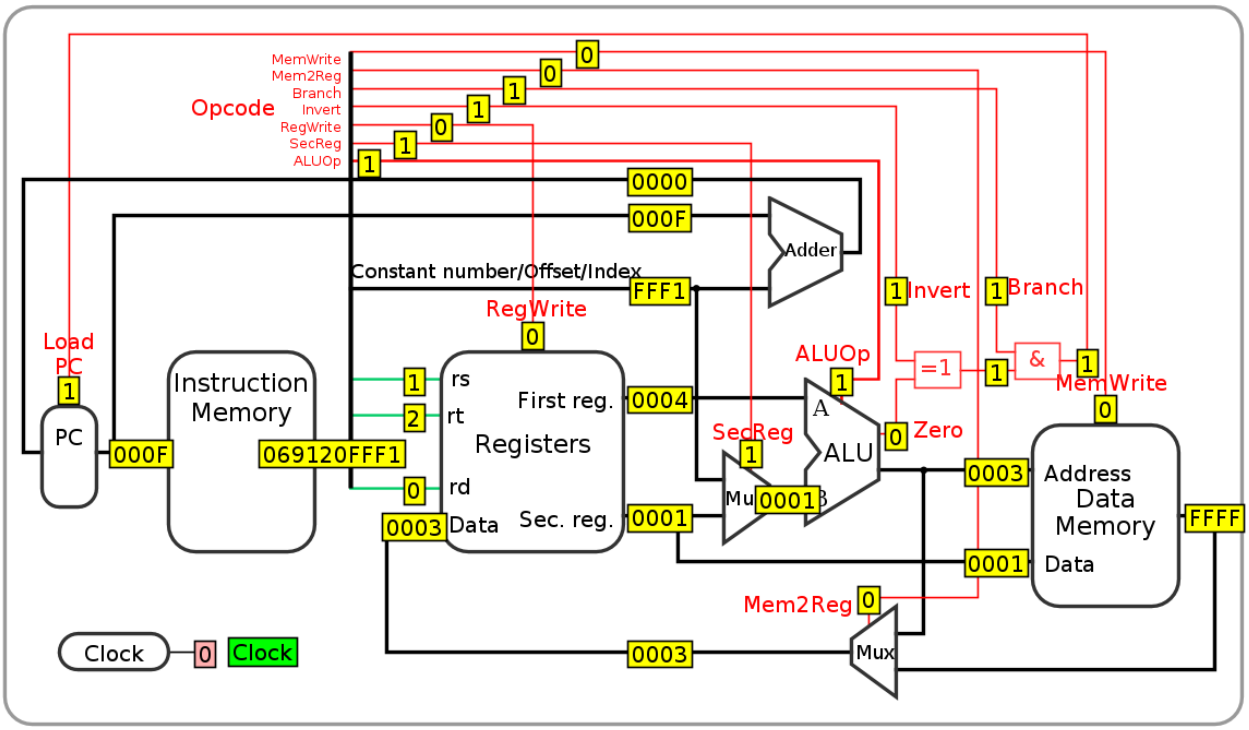
a) Geef de Mnemonic van de instructie die wordt uitgevoerd in onderstaand figuur:



LW rd, index, rs
LW $3, 0x6, $2

Student: **Collegekaartnummer:**

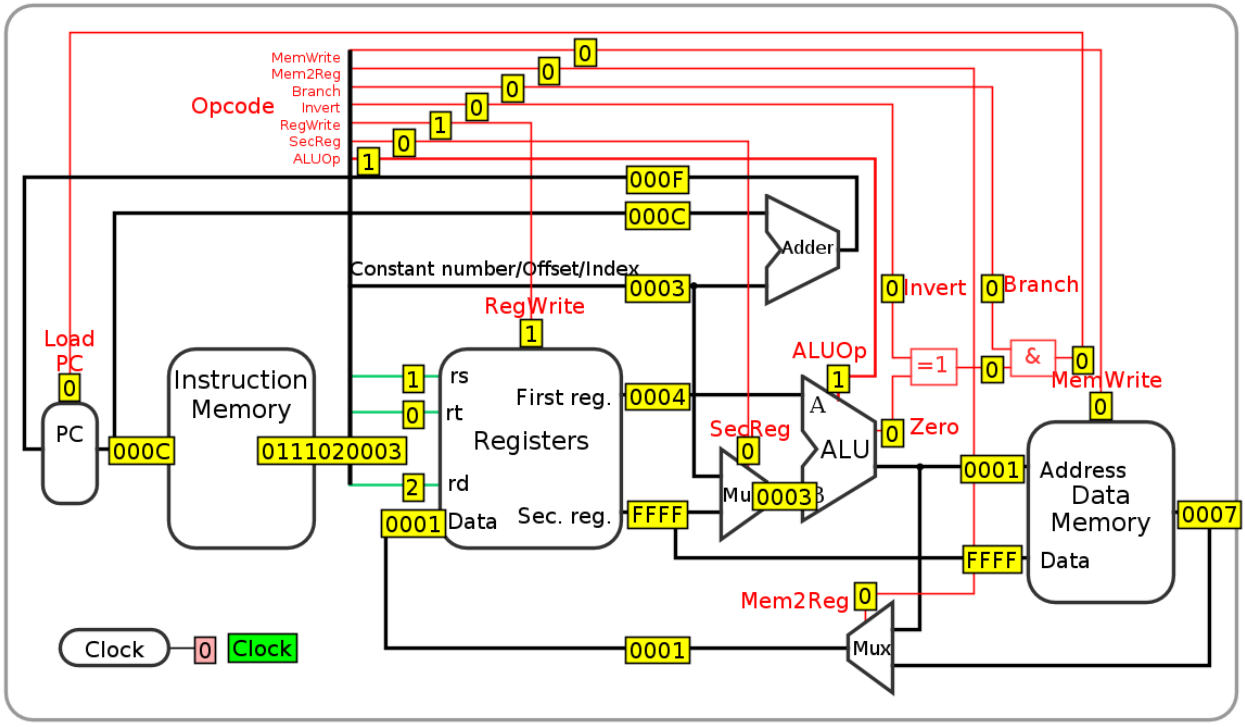
b) Geef de Mnemonic van de instructie die wordt uitgevoerd in onderstaand figuur:



BNE rs, rt, label
BNE $1, $2, 0xFFF1

Student: **Collegekaartnummer:**

c) Geef de Mnemonic van de instructie die wordt uitgevoerd in onderstaand figuur.



SUBI rd, rs, imm
SUBI $2, $1, 0x003

d) Waarom wordt bij de instructie “BNZ rt, label” het tweede register “rt” gebruikt en niet het eerste register “rs”?

Omdat de inhoud van het register door de ALU moet worden doorgestuurd zonder aangepast te worden om te testen of deze gelijk aan nul is, en dit kan alleen met het tweede register “rt” met gebruik van de ALU opcode “111”.

e) Wat is de inhoud van 16 bit register $3 na uitvoeren van het onderstaande programma?

```
LOADI $0, 0x3
SHLI $0, $0, 0x2
XORI $1, $0, 0xFFFF
ANDI $2, $1, 0x000F
SUBI $3, $2, 0x1
SW $3, 0x1, $0
COPY $2, $3
```

De inhoud van het register $3 is 2

Student: **Collegekaartnummer:**

vraag 3

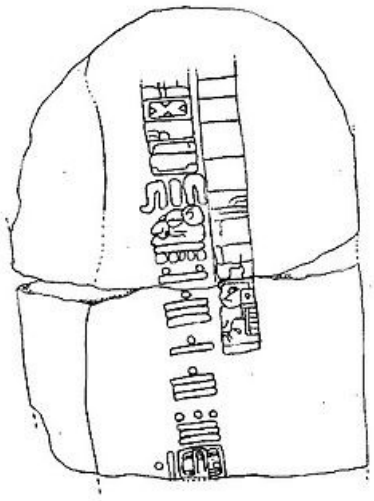
Deze vraag gaat uit van een representatie van gehele getallen m.b.v een 5-*bit two's complement* representatie. Vul in de volgende tabel de lege vakken in. Vakken met "\" hoeft u niet in te vullen. Optellen en aftrekken moeten gebaseerd zijn op de regels 5-*bit two's complement arithmetic.*

Number	***Decimal Representation***	***Binary Representation***
Zero	0	
\	-2	
\	9	
\	-14	
\		0 1100
\		1 0100
TMax		
TMin		
TMin + TMin		
TMin + 1		
TMax + 1		
-TMax		
-TMin		

Number	***Decimal Representation***	***Binary Representation***
Zero	0	0 0000
\	-2 = -16 +14 = -16 + 8 + 4 + 2	1 1110
\	9	0 1001
\	-14 = -16 + 2	1 0010
\	12 = 8 +4	0 1100
\	-12 = -16 + 4	1 0100
TMax	15	0 1111
TMin	-16	1 0000
TMin + TMin	-1 = -16 + 15	1 1111
TMin + 1	-15	1 0001
TMax + 1	-16 = 16 - 32	1 0000 (positive overflow)
-TMax	-15	1 0001
-TMin	-16 = 16 - 32	1 0000 (positive overflow)

Student: **Collegekaartnummer:**

vraag 4



Dit is een van de oudste Maya stenen met een datum,
hier afgebeeld is 7.16.6.16.18 (3 september 32 v.Chr.).

De Mayakalender bestaat uit een stelsel van tijdrekeningen die gecombineerd iedere dag vanaf het begin van de tijdsrekening (11 augustus 3114 v.Chr. in de Gregoriaanse kalender) een eigen id geven. Een dag werd aangeduid met het woord `kin`. Deze id van een `kin` lijkt op het formaat hoe men tegenwoordig ip-adressen opgeeft:

```
baktun . katun . tun . uinal . kin
```

Zo heeft vandaag, 25 september 2015, op de volgende manier door de Maya's genoteerd:

```
13 . 0 . 2 . 14 . 9
```

De Maya's gebruikten 20 als hun grondtal van hun telling, en noteerden elk vijftal als een horizontaal streepje met daarboven een nul tot vier punten. 25 september zou er dus als volgt uit zien:

a. Wat zou het symbool  betekenen?

Dat is het symbool voor nul.

Bonusvraag: Dit symbool is door de Maya's al in gebruikt in het jaar 35 v. Chr.
In welke eeuw is dit symbool in Europa geïntroduceerd?

Eind 12^{e} eeuw

Student: **Collegekaartnummer:**

b. Uit interesse voor de Maya's hebben we behoefte aan een data type `vigesimal` die van 0 tot 20 loopt. Hoeveel bits zijn er nodig voor een *unsigned* data type `vigesimal`?

5 bits (voldoende voor de range 0-31)

De kalender van de Maya's was niet puur 20-talig, de uitzondering was `uinal`, die van 0 tot 18 liep, zodat 1 `tun` bijna gelijk aan een zonnejaar is (18 `uinal` x 20 `kin` = 360 `kin`).

Het is nu je taak om een *floating point* data type `tun` te ontwerpen, dat links van de *point* het aantal Maja jaren aangeeft, en rechts van de *point* het aantal dagen (1/360 deel van een `tun`). Je hoeft hier geen voorzieningen te treffen voor negatieve getallen, getallen kleiner dan een dag (`kin`), nog voor een extreem grote getallen (groter dan 20 `baktun`).

c. Als het data type `tun` gerepresenteerd wordt met de volgende formule, welke waarde zou je dan kiezen voor *m* en *n*? Laat zien hoe je tot deze twee waardes komt.

$$d = \sum_{i=-n}^{m} 10^i \times d_i$$

n = 9, want $2^{-9} = 1/512$ (voldoende voor 1/360 deel van een `tun`)
n = 13, want $2^{13} = 8192$ (voldoende voor baktun * katun * tun = 20 * 20 * 20)

$$d = \sum_{i=-9}^{13} 10^i \times d_i$$

Student: **Collegekaartnummer:**

De periode van de Mayakalender (8000 `tun` ~ 288000 `kin`) is bijna gelijk aan de Juliaanse periode (7980 jaar ~ 2914695 dagen). Ook de begindatum van beiden periodes verschilt maar één millennium (11 augustus 3114 v. Chr. bij de Maya's, 24 november 4714 bij de Juliaanse dag).

De Juliaanse dag is nog steeds in gebruik in de techniek, vooral in de astronomie. Bijvoorbeeld de klok van de Sputnik gebruikte de Juliaanse dag (gemodificeerd, met een start datum van 17 november 1858). De Sputnik had een IBM 704 (36-bit machine) computer aan boord, en voor zijn klok is een *unsigned* data type van 18-bits gebruikt.

De eerste kunstmaan Sputnik

d. Tot welke datum was de werking van de klok van de Sputnik gegarandeerd? Laat zien hoe je tot deze datum komt (u mag daarbij er vanuit gaan dat er 365,25 Juliaanse dagen in een jaar gaan).

Wikipedia geeft augustus 2576:
https://en.wikipedia.org/wiki/Julian_day#Variants

dag 0 = 17 november 1858
dag max = 2^{18} -1 = 262143
jaar max = floor(262143 / 365.25) = 717 jaar
maand max = floor((261883 – floor(717*365.25)) /(360.25/12)=
floor((262143 - 261884)/30.4375) =
floor(259/30.4375)=8 maanden
dag max = (259 – floor(8 * 30.4375))= (259 – 243) = 16

datum max = 2 augustus 2576 (jaar en maand voldoende)

Student: **Collegekaartnummer:**

vraag 5

Vul door bestudering van de onderstaande IA32 assembler code (links) de missende stukjes C-code aan de rechterkant in. U kunt daarbij gebruik maken van de definities van `s1, s2` en `u1` onder aan de bladzijde. **Tip**: Teken eerst de *alignment* van de datastructuren.

a)
```
proc1:
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%eax
    movl 12(%eax),%eax
    movl %ebp,%esp
    popl %ebp
    ret
```
```
int proc1(struct s2 *x)
{
  return x->f[1]__________________ ;
}
```

b)
```
proc2:
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%eax
    movl 4(%eax),%eax
    movl 20(%eax),%eax
    movl %ebp,%esp
    popl %ebp
    ret
```
```
int proc2(struct s1 *x)
{
  return x->b.i->f[3]_____________ ;
}
```

c)
```
proc3:
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%eax
    movl (%eax),%eax
    movsbl 4(%eax),%eax
    movl %ebp,%esp
    popl %ebp
    ret
```
```
char proc3(union u1 *x)
{
  return x->i->e__________________ ;
}
```

d)
```
proc4:
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%eax
    movl (%eax),%eax
    movl 24(%eax),%eax
    movl (%eax),%eax
    movsbl 1(%eax),%eax
    movl %ebp,%esp
    popl %ebp
    ret
```
```
char proc4(union u1 *x)
{
  return x->i->g->d->a[1]_________ ;
}
```

Dit is de definitie van de gebruikte datastructuren `s1, s2` en `u1`:

```
struct s1 {
  char a[3];
  union u1 b;
  int c;
};
```
```
struct s2 {
  struct s1 *d;
  char e;
  int f[4];
  struct s2 *g;
};
```
```
union u1 {
  struct s1 *h;
  struct s2 *i;
  char j;
};
```

Student: **Collegekaartnummer:**

vraag 6

Deze vraag gaat over de recursieve C functie Silly:

```
int silly(int n, int *p)
{
  int val, val2;
  if (n > 0)
          val2 = silly(n >> 1, &val);
  else
          val = val2 = 0;
  *p = val + val2 + n;
   return val + val2;
}
```

Na het compileren krijgen we de volgende 32-bits machine code:

```
silly:
        pushl %ebp
        movl %esp,%ebp
        subl $20,%esp
        pushl %ebx
        movl 8(%ebp),%ebx       n -> ebx
        testl %ebx,%ebx
        jle .L3
        addl $-8,%esp           stack grows 8 bytes
        leal -4(%ebp),%eax
        pushl %eax              &val op stack
        leal (%ebx,%ebx),%eax
        pushl %eax              n >> 1 op stack
        call silly
        jmp .L4
        .p2align 4,,7
.L3:                            else branch
        xorl %eax,%eax          val = 0
        movl %eax,-4(%ebp)      val2 = 0; fall through
.L4:
        movl -4(%ebp),%edx      val
        addl %eax,%edx          val2 + val
        movl 12(%ebp),%eax      p -> eax
        addl %edx,%ebx          val2 + val + n
        movl %ebx,(%eax)        *p = val2 + val + n
        movl -24(%ebp),%ebx
        movl %edx,%eax          returnvalue = val2 + val
        movl %ebp,%esp
        popl %ebp
        ret
```

Student: **Collegekaartnummer:**

a) Op een gegeven moment wordt vanuit een main program de volgende aanroep gedaan:

```
x = silly(11, &y);
```

Hoe vaak wordt de functie in dit geval uitgevoerd? Wat zijn de uiteindelijke waarden van `x` en `y`?

De functie `silly` wordt 5x uitgevoerd.
Aan het eind van de recursie is `val = 9` en `val2 = 4`, dus is de returnwaarde 13 `(x=13)`.
`n` is dan nog steeds `11`, dus wordt de waarde 24 aan de pointer `*p` teruggegeven `(y=24)`.

b) Wordt de variabele `val2` op de *stack* bewaard?
Zo ja, welke *byte offset* (ten opzichte van `%ebp`) is er dan gebruikt, en waarom is het nodig om `val2` op de *stack* te bewaren?
Zo nee, waar wordt de variabele `val2` dan bewaard, en waarom dit een betere locatie dan de *stack?*

Nee, `val` wordt op de stack bewaard (met offset -4).
`Val2` is de returnwaarde van de functie, en die is per definitie `eax`.

c) Wat is er opgeslagen op de locatie -24(%ebp)? Als er niets opgeslagen is, verklaar dan waarvoor deze ruimte dan gealloceerd is. Als er iets op deze locatie opgeslagen is, waarom was dit nodig?

Op de locatie `-24(%ebp)` is de frame pointer van de *callers frame* opgeslagen.

d) Dezelfde vraag (als vraag c) voor de locatie -8(%ebp)?

Op de locatie `-8(%ebp)` is de `n` van de *callers frame* opgeslagen

e) Als u uit mag gaan van een *stack size* van 10 KB en een *stack frame size* van 64B, wat is dan de kleinste waarde van `n` waarbij de functie zal crashen? Waarom?

$n = 2^{27} = 134.217.728$, want 2^{26} past nog in 10x16x 64B = 10KB, dus bij de 27e recursieve aanroep volgt een crash.

U heeft het einde van het tentamen bereikt!