Student: **Collegekaartnummer:**

Tentamen Computersystemen

baiCOSY06 2e jaar bachelor AI, 2e semester 24 september 2013 13u-15u

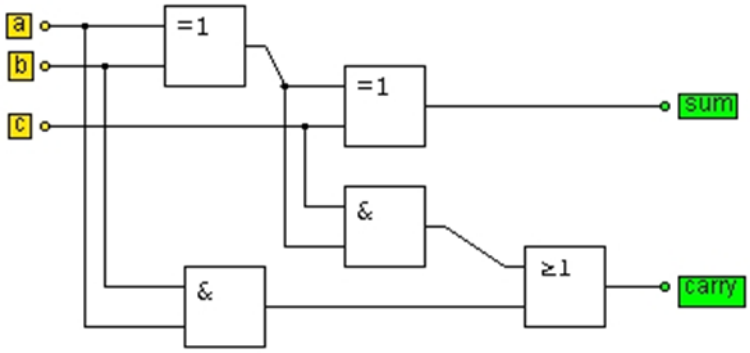
IWO 4.04A (blauw), Academisch Medisch Centrum, Meidreef 29, Amsterdam ZuidOost

Het is niet toegestaan de communicatie opties op je apparatuur te gebruiken: zet de verbindingen via ethernet, Bluetooth en het telefoonnetwerk uit!

Gebruik van een rekenmachine en de (elektronische) boeken behorende bij dit vak (Computer Systems, en Van 0 en 1 tot processor) is toegestaan. Succes!

Vraag 1 - Rekenschakeling

a. Teken de schakeling van een full-adder, zoals deze kan worden opgebouwd uit poorten.



Ook andere schakelingen zijn mogelijk, bijv. voor de carry 3 AND-poorten + een 3-input OR-poort.

b. Laat de binaire optelling zien van de getallen 37 en 61: zet de binaire representaties van deze twee getallen onder elkaar, streep eronder, en doe de optelling; noteer steeds de *carry's* boven het bovenste getal.

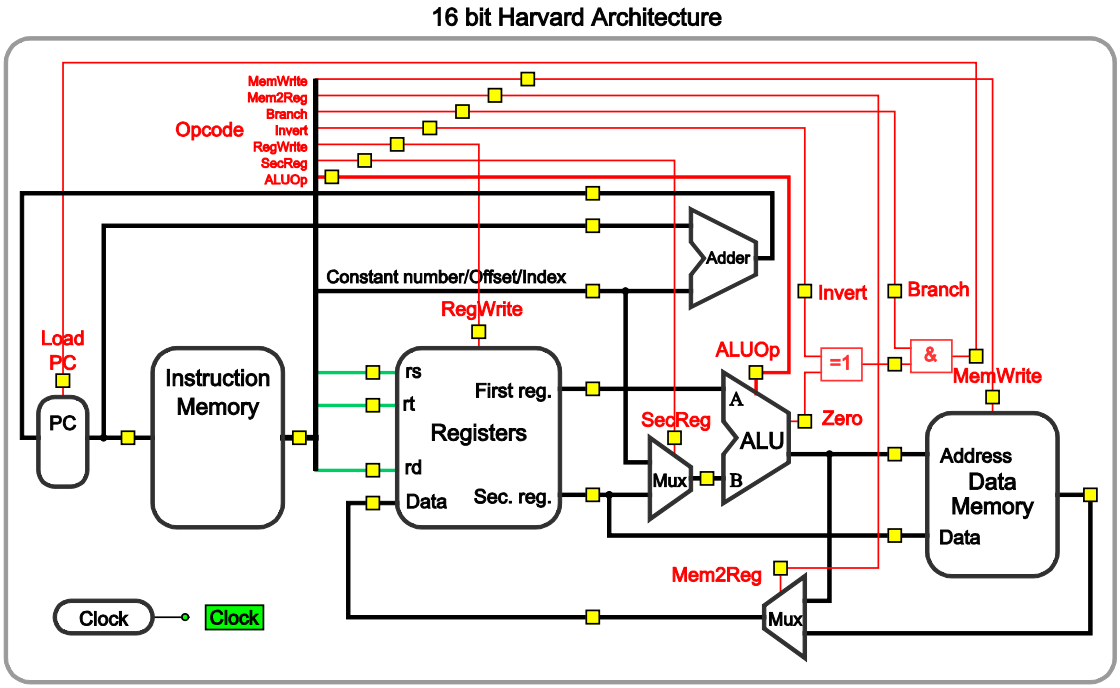
```
1111 1
 100101
 111101
-------
1100010
```

c. Wat is het grootste getal en wat is het kleinste getal in 16-bits *two's complement* representatie?

Student: **Collegekaartnummer:**

Kleinste getal = -32768 (0b1000000000000000=0x8000),
grootste getal = 32767 (0b0111111111111111= 0x7FFF)

Vraag 2 - Harvard Single Cycle Processor



a. Bij welke instructie(s) kan de lijn ‘Load PC’ actief (= 1) zijn ?

Bij de Branch-instructies

b. Bij welke instructie(s) is de RegWrite lijn = 0 ?

Deze is ‘0’ als er niet geschreven moet worden naar het destination register, dat is bij de branch-instructies, en bij de instructie SW (StoreWord).

c. Bij welke instructie(s) is de MemWrite lijn = 1 ?

Bij de instructie SW (StoreWord). Bij deze instructie wordt geschreven in het Data Memory.

d. Beschrijf voor elk van de twee multiplexers waarom deze in de schakeling van de 16 bit Harvard Architectuur zijn opgenomen.

e. *De multiplexer voor de B-ingang van de ALU maakt het mogelijk dat naast het aanbieden van een tweede register aan de ALU het nu ook mogelijk is een 'Constant number' op de B-ingang van de ALU aan te bieden: dit gebeurt bij de immediate-instructies*

De multiplexer voor de data-ingang van de registerbank maakt het mogelijk dat naast de output van de ALU het nu ook mogelijk is om data uit het Data Memory in het destination register te plaatsen: dit gebeurt bij de LOAD WORD instructie

f. Stel dat de verschillende componenten de volgende propagatietijd hebben: Memory Units 0,5 ns, Registers (read and write) 0,4 ns, ALU 0,2 ns. Wat is de maximale klokfrequentie voor deze processor?

Instruction Fetch = 0,5 ns (delay Instruction Memory)
Instruction decode/register read = 0,4 ns
Execute = 0,2 ns (delay ALU)
Memory access = 0,5 ns (delay Data memory)
Register Write-back = 0,4 ns
Opgeteld is dat 0,5+0,4+0,2+0,5+0,4 = 2,0 ns. De hoogst mogelijke klokfrequentie f = 0,5 GHz (=500 MHz).

Vraag 3 - Harvard pipelined processor

Op de 5-stage pipelined processor zonder forwarding zonder branch-prediction (geïntroduceerd in Hoofdstuk 9 van 0 en 1 tot pipelined processor) en wordt het volgende programma uitgevoerd:

```
        LOADI $6,0
        LOADI $5,5
Loop:   LW $1, tabel1, $6
        ANDI $1,$1,0xFFFE
        SW $1, tabel2, $6
        ADDI $6,$6,1
        BNE $6,$5,loop
HALT
```

a. Voeg NOPs toe (en niet meer dan noodzakelijk is) om dit programma correct te laten werken. Schrijf op hoe het programma nu wordt.

LOADI $6,0
LOADI $5,5
NOP

Student: **Collegekaartnummer:**

```
Loop:   LW $1, tabel1, $6
        NOP
        NOP
        ANDI $1,$1,0xFFFE
        NOP
        NOP
        SW $1, tabel2, $6
        ADDI $6,$6,1
        NOP
        NOP
        BNE $6,$5,loop
HALT
```

b. Als dit programma wordt uitgevoerd op een 5-stage pipelined processor met forwarding en zonder branch prediction, kunnen alle NOPs er dan weer uit? Zo nee, welke niet?

Bijna alle NOPs kunnen er nu weer eruit, behalve één NOP na de LW-instructie.

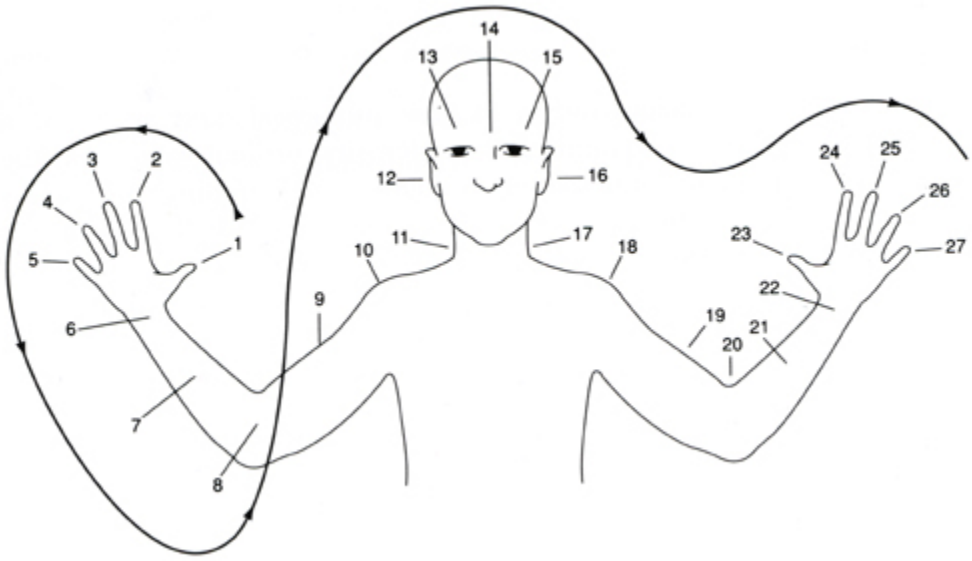
c. Als de processor nu (naast forwarding) ook branch-prediction heeft dan is een extra NOP nodig. Waar, en waarom?

Deze vraag is niet correct, en vervalt.

Student: **Collegekaartnummer:**

Vraag 4 –Base-27 representatie

De Oksapmin is een volk dat leeft in de binnenlanden van Papua Nieuw Guinea. De Oksapmin maken gebruik van een uitgebreid stelsel van tellen met behulp van lichaamsdelen; ze gebruiken niet alleen hun handen maar ook hun armen en hoofd. In totaal kunnen ze met hun lichaam tot 27 tellen:



Het Oksapmin telsysteem (Stefan Marti, MIT; gebaseerd op Geoffrey Saxe, UCB)

Het telsysteem is symmetrisch rond neus (*lum*). Getallen groter dan 14 worden aangegeven met het bijwoord *tən* (other). Als de cyclus klaar is roept men *tit fu* en gaat men via de pols weer terug.

Nu is het jouw taak om een datatype voor Oksapmin's numeriek systeem te ontwerpen.

a. Ten eerste hebben we behoefte aan een data type `titfu` die van 1 tot 27 loopt. Toon aan dat 5 bits voldoende zijn voor het data type `titfu`.

2^5=32, dus 5 bits

Student: **Collegekaartnummer:**

Bij experimenten met jonge kinderen kwam men er achter dat dit numeriek systeem tot verwarring kan leiden, omdat deze kinderen twee getallen die symmetrisch t.o.v. elkaar op het lichaam liggen, evenveel waarde toedichten. Bijvoorbeeld, als twee neven respectievelijk een opbrengst van hun akker hadden van respectievelijk *amun*, *amun tən* zoete aardappelen, dan vonden sommige kinderen dat de neven een gelijke opbrengst hadden (*amun* is de eleboog).

Het idee is om een versie van het data type `titfu` te ontwerpen waar de bit representatie van twee getallen die symmetrisch t.o.v. liggen elkaars complement zijn. Dus als we bijvoorbeeld voor het getal *amun* de representatie 01000 kiezen, dan is zou de representatie voor *amun tən* 10111 zijn. Dit kan door voor de *Most Significant Bit* de weight 13 toe te wijzen en de resterende bits hiervan bij op te tellen:

$$B2TitFu_w(\vec{x}) = 13x_{w-1} + \sum_{i=0}^{w-2} x_i 2^i$$

b. Toon aan dat met deze functie $B2TitFu_w(\vec{x})$ er twee bitpatronen zijn die het getal 14 (*lum*) opleveren en dat deze twee bitpatronen elkaars complement zijn.

00110 (6) → 11001 (22 =13+ 9)
00111 (7)→ 11000 (21=13+8)
01000 (8) → 10111 (20=13 + 7)
01001 (9) → 10110 (19=13 + 6)
01010 (10)→ 10101 (18=13 + 5)
01011 (11)→ 10100 (17=13+ 4)
01100 (12)→ 10011 (16=13 + 3)
01101 (13)→ 10010 (15=13+ 2)
01110 (14)→ 10001 (14=13+ 1) !own complement

Echter, deze functie $B2TitFu_w(\vec{x})$ werkt alleen zoals bedoelt voor de getallen tussen de 6 (*xadəp*) tot 22 (*tən xadəp*); de verwarring bij de andere getallen is geompliceerder omdat men bij beiden handen bij de duim begint te tellen. Bij Oksapmin kinderen kan dus de volgende verwarring ontstaan:

duim	*tipun*	1	23	*tən tipun*
wijsvinger	*ləwatipun*	2	24	*tən ləwatipun*
middelvinger	*bumlip*	3	25	*tən bumlip*
ringvinger	*xətlip*	4	26	*tən xətlip*
pink	*xətxət*	5	27	*tən xətxət*

c. Ontwerp een alternatieve functie $B2TitFu_w'(\vec{x})$ die zorgt dat de getallen 23 (*tən tipun*) tot 27 (*tən xətxət*) het complement zijn de getallen van getallen 1 (*tipun*) tot 5 (*xətxət*), als deze laatste de bitreprentatie 00001 tot 00101 hebben gekregen. **Hint**: deze functie $B2TitFu_w'(\vec{x})$ wordt alleen aangeroepen als $x_{w-1} \& x_{w-2} \& (x_{w-3}|x_{w-4})$

Student: **Collegekaartnummer:**

00001 → 11110 (23=37-14=29-6)
00010 → 11101 (24=37-13=29-5)
00011 → 11100 (25=13+12=37-12=29-4)
00100 → 11011 (26=37-11=29-3)
00101 → 11010 (27=37-10=29-2)
Dus $37x_{w-1} - \sum_{i=0}^{w-2} x_i 2^i$

d. Als er dus twee bitpatronen worden gebruikt voor het getal 14, hoeveel bitpatronen zijn er nu door `titfu` nog niet gebruikt? Op wat voor rekenkundige manier zou men de niet gebruikte bitpatronen nog kunnen gebruiken?

00000 → nul *10000→ 2x titfu*
01111 → 1x titfu *11111 → 3x titfu*

Vraag 5 – floating point representatie

De volgende procedure heeft als argument een *single-precision floating point* getal in IEEE formaat en print voor dit argument een aantal eigenschappen uit. Vul de missende code in zodat de eigenschappen correct geprint worden.

```
void classify_float(float f)
{
      /* Unsigned value u has same bit pattern as f */
      unsigned u = *(unsigned *) &f;
      /* Split u into the different parts */
      int sign = (u >> 31) & 0x1; // The sign bit
      int exp = _________________; // The exponent field
      int frac = _________________; // The fraction field
      /* The remaining expressions can be written in terms of the
      values of sign, exp, and frac */
      if (_____________________)
            printf("Plus or minus zero\");
      else if (_____________________)
            printf("Nonzero, denormalized\");
      else if (_____________________)
            printf("Plus or minus infinity\");
      else if (_____________________)
            printf("NaN\");
      else if (_____________________)
            printf("Greater than -1.0 and less than 1.0\");
      else if (_____________________)
            printf("Less than or equal to -1.0\");
      else
            printf("Greater than or equal to 1.0\");
}
```

Student: **Collegekaartnummer:**

bonusvraag

Welk expressie kun je gebruiken om een getal te maken dat klein genoeg is om de functie `classify_float` te testen voor *denormalized* getallen?

Antwoord

```
void classify_float(float f)
{
        /* Unsigned value u has same bit pattern as f */
        unsigned u = *(unsigned *) &f;
        /* Split u into the different parts */
        int sign = (u >> 31) & 0x1; // The sign bit
        int exp = (u >> 23) &0xff; // The exponent field
        int frac = u & 0x7fffff; // The fraction field
        /* The remaining expressions can be written in terms of the
        * values of sign, exp, and frac */
        if (frac == 0x000000 && exp == 0x00)
                printf("Plus or minus zero\n");
        else if (exp == 0x00)
        int exp = (u >> 23) &0xff; // The exponent field
        int frac = u & 0x7fffff; // The fraction field
        /* The remaining expressions can be written in terms of the
        * values of sign, exp, and frac */
        if (frac == 0x000000 && exp == 0x00)
                printf("Plus or minus zero\n");
        else if (exp == 0x00)
                printf("Nonzero, denormalized\n");
        else if (exp == 0xff && frac == 0x000000)
                printf("Plus or minus infinity\n");
        else if (exp == 0xff)
                printf("NaN\n");
        else if (exp < 127) // float bias
                printf("Greater than -1.0 and less than 1.0\n");
        else if (sign == 1)
                printf("Less than or equal to -1.0\n");
        else
                printf("Greater than or equal to 1.0\n");
}
```

Bonusantwoord

```
classify_float(+1.0/pow(2,127+22)); //+23 gives zero
```

Student: **Collegekaartnummer:**

Vraag 6 – Loop structuren in assembly

De volgende C functie is gecompileerd voor een IA32 *(x86)* machine met behulp van Linux/GAS.

```
looper:
pushl %ebp
movl %esp,%ebp
pushl %esi
pushl %ebx
movl 8(%ebp),%ebx
movl 12(%ebp),%esi
xorl %edx,%edx
xorl %ecx,%ecx
cmpl %ebx,%edx
jge .L25
.L27:
movl
(%esi,%ecx,4),%eax
cmpl %edx,%eax
jle .L28
movl %eax,%edx
.L28:
incl %edx
incl %ecx
cmpl %ebx,%ecx
jl .L27
.L25:
movl %edx,%eax
popl %ebx
popl %esi
movl %ebp,%esp
popl %ebp
ret
```

```
int looper(int n, int *a) {
  int i;
  int x = ________________;

  for(i = ______________;
    ____________________;
    i++) {

    if (_____________________)
      x = ____________________;

    _____________________;
  }
  return x;
}
```

Analyseer de assembly code en vul de missende onderdelen van de functie `looper` in .

Let op:

- Gebruik de namen van de C-variabelen (`n,a,i,x`) en niet de registernamen
- Gebruik de array notatie voor de elementen van `a`, ipv pointer notatie

Student: **Collegekaartnummer:**

Antwoord

```
int looper(int n, int *a) {
  int i;
  int x = 0;

  for(i = 0;
      x<n;
      i++) {

     if (x<=a[i])
       x = a[i];

     x++;
  }
  return x;
}
```