Student: **Collegekaartnummer:**

Tentamen Computersystemen

baiCOSY06 2e jaar bachelor AI, 2e semester 27 september 2012 13u-15u

IWO 4.04C (rood), Academisch Medisch Centrum, Meidreef 29, Amsterdam ZuidOost

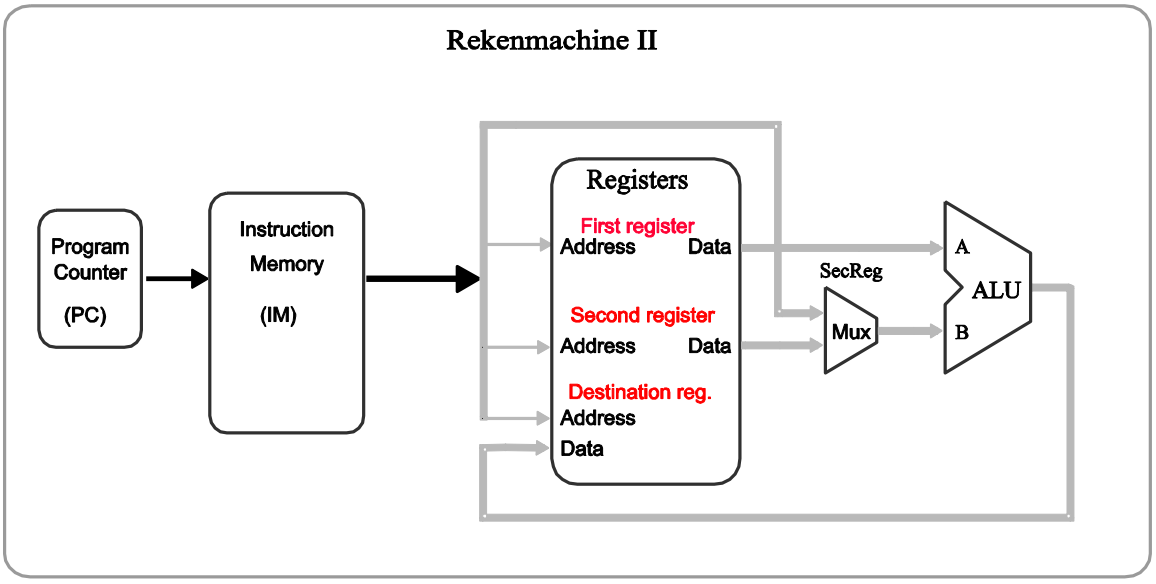
Het is niet toegestaan communicatieapparatuur zoals tablets en telefoons te gebruiken: zet deze apparatuur uit!

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

vraag 1

Vraag 1 Rekenschakelingen
Vraag 1a: Wat is een Full Adder?
Vraag 1b: Geef de waarheidstabel van een Full Adder weer.

Vraag 2 Rekenmachine II
Vraag 2a: Wat is de opcode van een instructie?
Vraag 2b: Arceer het datapad van een ADDI-instructie in figuur 1 Geef ook aan welke adresingang(en) worden gebruikt bij deze instructie.



Figuur 1: Rekenmachine II

Vraag 2c: Welke component(en) hebben een kloksignaal nodig om de Rekenmachinemachine II goed te laten werken?

Student: **Collegekaartnummer:**

Vraag 3 Harvard machine *Zie figuur 2*.

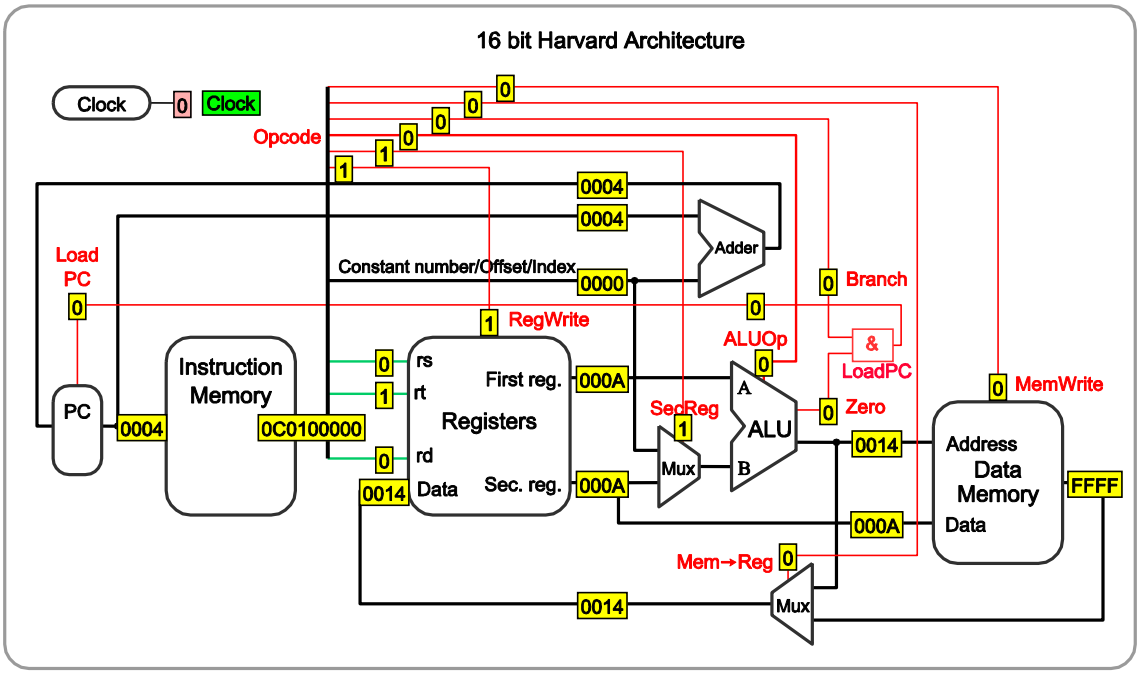
Vraag 3a: Uit welke fases bestaat het executeren van een “Store Word”-instructie?

Vraag 3b: Wat is de functie van de component registers?

Vraag 3c: Noem alle instructies waarbij tijdens het executeren ervan het SecReg-bit 0 is? Licht je antwoord toe.

Vraag 3d: Welke instructie wordt in figuur 2 uitgevoerd? Licht je antwoord toe. Geef de syntax van deze instructie weer.

Vraag 3e: Wat is de functie van de component “Adder”? Licht je antwoord toe.

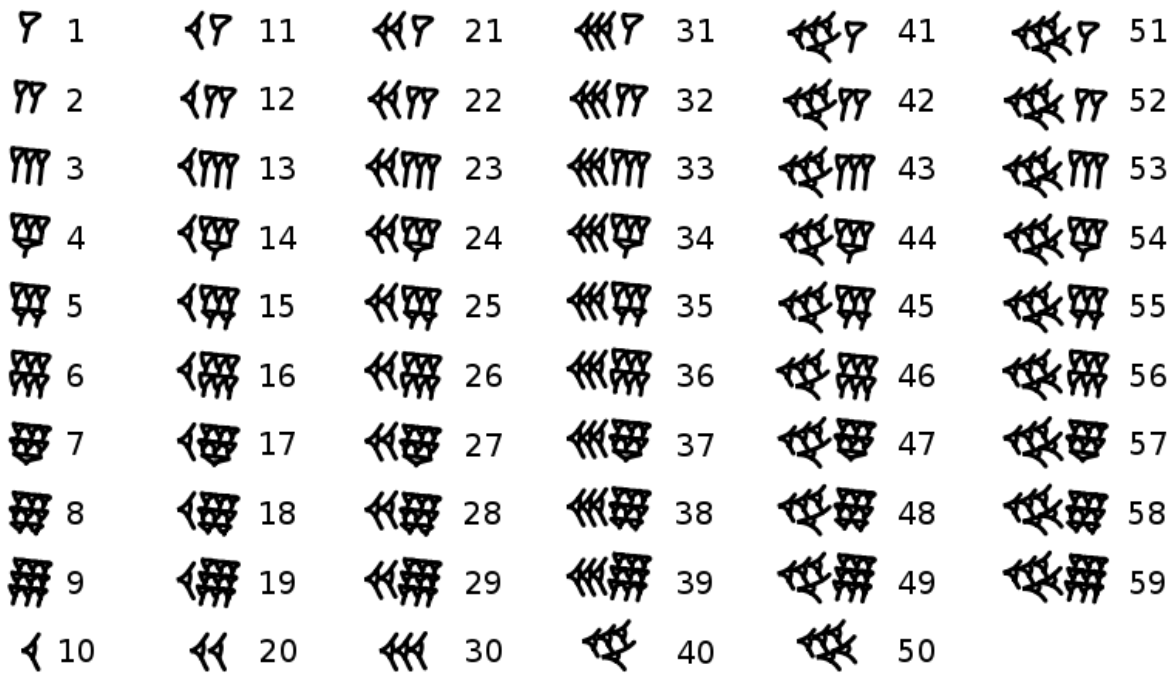


Figuur 2: Harvard machine

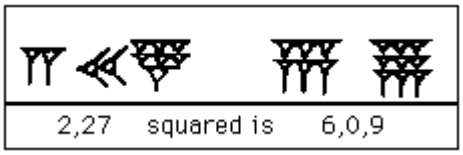
Student: **Collegekaartnummer:**

vraag 4

De Babeloniërs gebruikten een zestigtallig stelsel door twee symbolen te combineren;
 voor de eenheden en  voor de tientallen:



De Babeloniërs maakten grotere getallen door ze naast elkaar te zetten. Het begrip nul was in 3000 v.Chr. nog niet uitgevonden, daarvoor in de plaats gebruikten ze een spatie. Een voorbeeld is de volgende berekening, waar men de ruimte tussen de 6 en de 9 duidelijk groter is dan de ruimte tussen de 2 en de 27:



$147^2 = 21609$

$(2x60^1 + 27x60^0)^2 = (6x60^2 + 9x60^0)$

Om verwarring te voorkomen, plaatst men tegenwoordig komma's of dubbele punten tussen de Babelonische getallen.

a. Geef de formule **b2U$_w$** *("Babelonian to unsigned", length **w***) voor het 60-talig stelsel van de Babeloniers.

Student: **Collegekaartnummer:**

De Babeloniers kende het gegrip *floating point* ook al, hoewel ze ook daar net als voor de nul geen notatie voor hadden. Tegenwoordig zetten we een punt-komma op de juiste plaats in Babelonische getallen. Bijvoorbeeld; de Babeloniers schatten π af op 25/8 (3.125)[1], dus in hun notatie (3;7:30) = $(3x60^{0} + 7x60^{-1} + 30x60^{-2})$:

b. Breid de functie $\mathbf{b2U_w}$ uit met negatieve indices tot de formule $\mathbf{b2F_w}$ *("Babelonian to unsigned float", length* ***w***) voor het 60-talig stelsel van de Babeloniers.

c. Geef aan of de Babelonische schatting van π (25/8) in een Bineair *signed floating point* formaat van een enkele Byte (8 bits) past. Zo ja, geef een voorbeeld van zo'n 8bits *floating point* formaat representatie van 25/8. Zo nee, toon aan waarom dit getal niet exact gerepresenteerd kan worden.

De Babeloniers gebruikte de reciproke van 7 nauwelijks. Archimedes vond met behulp van de meerzijdige polygonen (96 zijden) de eerste convergent van π: 22/7.

d. Geef aan of de schatting van Archimedes (22/7) in een Bineair *signed floating point* formaat van een enkele Byte (8 bits) past. Zo ja, geef een voorbeeld van zo'n 8bits *floating point* formaat representatie van 22/7. Zo nee, toon aan waarom dit getal niet exact gerepresenteerd kan worden.

De Chinesen gebruikten een vergelijkbare methode als Archimedes om π af te schatten. In 480 na Christus vond Zu Chongzhi (**祖沖之**) de convergent 355/113 met behulp van een polygoon met 12288 zijden. Deze schatting staat bekend onder de naam **Milü** (**密率**). Deze breuk is in de toenmalige Chinese notatie geschreven als:

Pas in de 15e eeuw kwam Jamshid al-Kashi (غیاث‌الدین جمشید کاشانی

Student: **Collegekaartnummer:**

vraag 5

Een van de voordelen van de x86-64 architectuur is dat er 16 *integer* registers beschikbaar zijn, terwijl men het in de IA32 met acht *integer* en acht *floating-point* registers moet doen. Bekijk de volgende code:

```
int swap_add(int *xp, int *yp)
{
    int x = *xp;
    int y = *yp;

    *xp = y;
    *yp = x;
    return x + y;
}
int caller()
{
    int arg1 = 534;
    int arg2 = 1057;
    int sum = swap_add(&arg1, &arg2);
    int diff = arg1 - arg2;

    return sum * diff;
}
```

Als we deze code compileren voor de een 64-bits machine, krijgen we de volgende *assembly code*:

```
swap_add:
      movl    (%rdi), %eax
      movl    (%rsi), %edx
      movl    %edx, (%rdi)
      movl    %eax, (%rsi)
      leal    (%rdx,%rax), %eax
      ret
```

Als we dezelfde code compileren voor IA32, ziet de *assembly code* er als volgt uit:

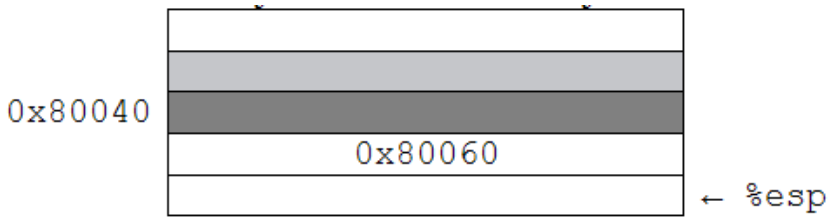
```
swap_add:
1     the set up of a new frame for the function
2     pushl %ebp        save old framepointer
3     movl %esp,%ebp    set new %ebp just after the stackpointer %esp
4     pushl %ebx        callee save
5     the body of the function
6     movl 8(%ebp),%edx
7     movl 12(%ebp),%ecx
8     movl (%edx),%ebx
9     movl (%ecx),%eax
10    movl %eax,(%edx)
11    movl %ebx,(%ecx)
12    addl %ebx,%eax
13    restoring the frame of the calling function
14    popl %ebx              callee restore
15    movl %ebp,%esp         recalcute old stackpointer
16    popl %ebp              restoring old framepointer
17    ret
```

Student: **Collegekaartnummer:**

a. Waarom maakt de 64-bits code geen gebruik van de *stack*?
b. Welk register wordt er bij de IA32 gebruikt om het eindresultaat te bewaren?
c. Waarom worden `%eax,%edx` en `%ecx` niet gesaved op de *stack*?
d. Is er bij de IA32 ook een register voor het *returnadress*?
e. Wordt de *stackpointer* verhoogd of verlaagd met 4 voor iedere `pushl`?
f. Als bij het aanroepen van de functie `swap_add` de stackpointer de waarde `0x80040` had, welke waarde krijgt `%ebp` dan op regel 3?
g. Vlak voor de aanroep naar `swap_add` staat bij de IA32 de volgende code van de `caller:`

```
movl $534,-8(%ebp)
movl $1057,-4(%ebp)
addl $-8,%esp
leal -4(%ebp),%eax
pushl %eax
leal -8(%ebp),%eax
pushl %eax
call swap_add
```

Teken de *stack* op regel 5 van de code `swap_add` van geheugenadress `0x80048` tot `0x80038`. De inhoud van `0x80040` kun je niet weten, doch zijn functie wel.



Succes!