Student: **Collegekaartnummer:**

Tentamen Computersystemen

baiCOSY06 2e jaar bachelor AI, 2e semester 26 oktober 2011

vraag 1

a. Welke drie “hazards” kunnen bij een pipeline optreden?.
Geef aan op welke manier twee van die drie hazards zo goed als mogelijk is verholpen kunnen worden

b. De “instruction memory” en de “data memory” zijn twee separate componenten. Als deze twee nu worden samengevoegd in een processor met pipeline waar dient dan rekening mee gehouden te worden?
Behandel 2 punten.

c. De data-stroom door een cache moet op een doordachte manier geschieden. Beschrijf twee twee "write strategies". Bij welke is er een zgn. dirty-bit aanwezig?
Licht je antwoord toe.

d. Behandel de voor- en nadelen van de ’fully associative cache’ in vergelijking met de ’direct mapped cache’.

vraag 2

De volgende code is het hart van een *tiny* webserver die per *client* een proces opstart:

```
listenfd = Open_listenfd(port);
while (1) {
    connfd = Accept(listenfd, (SA *) &clientaddr, &clientlen);
    if (Fork() == 0) {
        Close(listenfd); /* Child closes its listening socket */
        dotit(connfd);    /* Child services client */
        Close(connfd);   /* Child closes connection with client */
        exit(0);         /* Child exits */
    }
    Close(connfd); /* Parent closes connected socket (important!) */
}
```

a) Waarom is het belangrijk dat de *parent* de *file-descriptor* `connfd` sluit?

b) Als de *parent* de *file-descriptor* `connfd` sluit, is de *child* dan nog wel in staat om met de *client* te communiceren? Verklaar uw antwoord.

c) Dat de *child* de *file-descriptor* `connfd` sluit, is netjes, doch strikt niet nodig. Waarom niet?

Student: **Collegekaartnummer:**

vraag 3

a. Bestudeer het volgende C-code. Om de code leesbaar te houden, is er geen rekening gehouden met functies die een error code teruggeven. U mag aannemen dat alle functies succesvol worden uitgevoerd.:

```
int val = 10;
void handler(sig)
{
   val += 5;
   return;
}
int main()
{
   int pid;
   signal(SIGCHLD, handler);
   if ((pid = fork()) == 0) {
      val -= 3;
   exit(0);
   }
   aitpid(pid, NULL, 0);
   printf("val = %d\n", val);
   exit(0);
}
```

Wat is de waarde die dit programma naar de `stdout` print (`val =`)?

b. Iemand geeft de volgende opmerking over de onderstaande code. Het is beter de malloc te vervangen door de declaratie `int a[200],` omdat een thread de heap deelt met alle andere threads van het process maar niet de stack.
Geef aan waarom u het u het eens bent met deze opmerking of waarom niet..:

```
int main ( void ) {
       ..
       pthread_create( &tid, NULL, thread, (void *) i );
       ..
}

void *thread( void *vptr ) {
       int *a = malloc( 200*sizeof(int) );
       ..
}
```

Student: **Collegekaartnummer:**

vraag 4

Stel dat er een procedure moet schrijven die het *inner product* van twee vectoren `x` en `y` berekent. Een naïeve implementatie van de functie levert een CPE van 16-17 voor een x86-64 machine en 26-29 CPE voor een IA32 machine voor *integer*, *single precision* and *double precision* data. Door *code movement* en het verwijderen van onnodige *geheugen gebruik* krijgen we de volgende geoptimalizeerde code:

```
1      /* Accumulate in temporary */
2      void inner4(vec_ptr x, vec_ptr y, data_t *dest)
3      {
4          long int i;
5          int length = vec_length(x);
6          data_t *xdata = get_vec_start(x);
7          data_t *ydata = get_vec_start(y);
8          data_t sum = (data_t )0;
9
10         for (i=0; i < length; i++) {
11            sum += xdata[i] * ydata[i];
12         }
13         *dest = sum;
14     }
```

Metingen geven aan dat deze functie een snelheid heeft CPE heeft van 3.00 voor zowel *integer* als *floating point* data. Voor data type `float` ziet de assembly code van de functie `inner4` voor een x86-64 machine er als volgt uit:

```
      inner4: data_t = float
      xdata in %rbx, ydata in %rax, length in %rcx
      i in %rdx, sum in %xmm1
A     .L87:                          loop:
B      movss (%rbx,%rdx,4), %xmm0    Get xdata[i]
C      mulss (%rbx,%rdx,4), %xmm0    multiply by ydata[i]
D      addss %xmm0, %xmm1            add to sum
E      addq $1, %rdx                 increment i
F      cmpq %rcx, %rdx               Compare i:length
G      jl .L87                       If <, goto loop
```

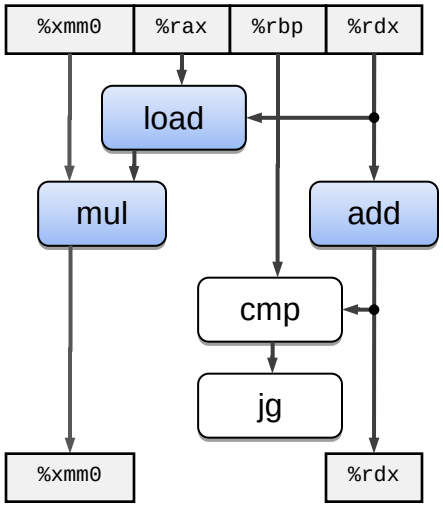
U mag er vanuit gaan dat alle functionele eenheden van de x86-64 machine de volgende eigenschappen hebben:

	Integer		Single-Precision		Double-Precision	
Operation	Latency	Issue	Latency	Issue	Latency	Issue
Addition	1	0.33	3	1	3	1
Multiplication	3	1	4	1	5	1
Division	11-21	5-13	10-15	6-11	10-23	6-19

Tabel 1: Latency *en* issue *eigenschappen van de Intel Core i7 rekenoperaties.* Latency *geeft het aantal* clock cycles *nodig om de operatie uit te voeren.* Issue *geeft een indicatie van het aantal cycles die gewacht moeten worden voordat een nieuwe operatie gestart kan worden. De waarden voor* Division *zijn afhankelijk van de waarde van de getallen waarmee gerekend wordt.*

Student: **Collegekaartnummer:**

a. Teken een data-flow in dezelfde stijl als Figuur 1, waaruit de het kritische pad van de operaties in `inner4` duidelijk wordt.




```
       combine4: data_t = float, OP=*
       i in %rdx, data in %rax, length in
%rbp, acc in %xmm0
A .L488:                        loop:
B mulss(%rax,%rdx,4),%xmm0 multiply acc
C addq $1, %rdx                 increment i
D cmpq %rdx, %rbp               Compare length:i
E jl .L448                      If >, goto loop
```

Figuur 1: Een voorbeeld van een data-flow diagram. In dit geval voor `combine4`.

b. Welke theoretische waarde van de CPE zou uit de kritische pad analyse voor `inner4` mogelijk moeten zijn voor het data-type `float` type?

c. Welke theoretische waarde van de CPE zou uit de kritische pad analyse voor `inner4` mogelijk moeten zijn voor *integer data*?
Men mag er vanuit gaan dat de compiler voor *integers* een vergelijkbare sequentie van *assembly* operaties genereert.

d. Verklaar dat voor deze functie een snelheid heeft CPE van 3.00 wordt gemeten voor *floating point* data, terwijl de latency van deze operatie voor single en double precision data respectievelijk 4 en 5 clock cycles is.

Succes!