Student: **Collegekaartnummer:**

Tentamen Computersystemen voor AI programmeurs

baiCSAI3 2e jaar bachelor AI, 2e semester 23 maart 2009

vraag 1

U heeft een re-engineering taak gekregen: u dient de C-code te reconstrueren die hoort bij de volgende definities, en de IA32 *assembly* code die ontstaan is bij het compileren van de oorspronkelijk C code.

```
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;
};
```

Zoals u weet is de grootte van een `char` 1 byte, en een `int` 4 bytes Het is misschien handig om configuratie van de drie data-structuren hieronder te tekenen:

Student: **Collegekaartnummer:**

Vul nu voor elk van de vier C-functies aan de rechterkant de missende code in, aan de hand van de *assembly-code* aan de linkerkant. De instructie `movsbl` kopieert een enkele byte naar een 32 bits adres, en vult de andere 3 bytes op met de *sign*-bit van de gekopieerde byte:

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->_____________________ ;
}
```

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->_____________________ ;
}
```

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->_____________________ ;
}
```

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->_____________________ ;
}
```

Student: **Collegekaartnummer:**

vraag 2

Neem de volgende functie eens in ogenschouw. De functie is bedoeld om het snel het product van de elementen van een *array* uit te rekenen. De optimalizatie van de functie is gedaan door een drievoudige *loop-unrolling.*

```
int aprod(int a[], int n)
{
   int i, x, y, z;
   int r = 1;

   for (i = 0; i < n-2; i+= 3) {
      x = a[i];
      y = a[i+1];
      z = a[i+2];
      r = r * x * y * z; // Product computation
   }
   for (; i < n; i++)
      r *= a[i];
   return r;
}
```

Voor de regel met het commentaar `Product computation`, kunnen verschillende varianten bedacht worden:

```
r = ((r * x) * y) * z; // A1
r = (r * (x * y)) * z; // A2
r = r * ((x * y) * z); // A3
r = r * (x * (y * z)); // A4
r = (r * x) * (y * z); // A5
```

Nu gaan we naar de snelheid van de verschillende varianten kijken, en wel in de termen van *cycles per element* (CPE). De metingen worden gedaan op Intel Pentium III. Zoals u wellicht herinnert, is voor de *integer* vermenigvuldiging de *latency* 4 *cycles*, maar kan een vermenigvuldiging iedere *cycle* gestart worden.

De volgende tabel geeft enige waarde van de CPE weer, doch andere waarden missen.
Met “Theoretical CPE” word de snelheid bedoeld die bereikt kan worden als de latency en de issue time de enige beperkende factor zouden zijn.

Version	Measured CPE	Theoretical CPE
A1	4.00	
A2	2.67	
A3		$4/3 = 1.33$
A4	1.67	
A5		$8/3 = 2.67$

Vul de missende waarden in. Voor de ‘Measured CPE’, kunt u gebruik maken van waarden van varianten die volgens u het zelfde computationele gedrag vertonen. Voor de ‘Theoretical CPE’, kunt u het aantal *cycles* dat voor één iteratie nodig is, als u alleen rekening houden hoeft te houden met de *latency* and *issue time*, en die dan delen door de *loop-unrolling* factor 3.

Motiveer uw antwoord.

Student: **Collegekaartnummer:**

Vraag 3

Bekijk de volgende twee C datastructuren:

```
typedef struct {
    short code;
    int start;
    char raw[3];
    double data;
} OldSensorData;
```

```
typedef struct {
    short code;
    short start;
    char raw[5];
    short sense;
    short ext;
    double data;
} NewSensorData;
```

a) Beiden datastructuren zijn bij het gebruik van de Linux uitlijning niet even groot. Gebruik de onderstaande geheugenblokken om aan te geven hoe de uitlijning van beiden datastructuren er uit ziet. Markeer ongebruikt geheugen een kruis. Zoals je weet, is de grootte van een `short` 2 bytes en een `double` 8 bytes.

```
OldSensorData:
  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                                                       |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

NewSensorData:
  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                                                       |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

b) Bestudeer nu onderstaande code fragment:

```
void convertdata(OldSensorData *oldData)
{
    NewSensorData *newData;

    /* this zeros out all the space allocated for oldData */
    bzero((void *)oldData, sizeof(oldData));

    oldData->code = 0x104f;
    oldData->start = 0x80501ab8;
    oldData->raw[0] = 0xe1;
    oldData->raw[1] = 0xe2;
    oldData->raw[2] = 0x8f;
    oldData->raw[-5] = 0xff;
    oldData->data = 1.5;

    newData = (NewSensorData *) oldData;

    ...
```

Nadat dit stukje code is aangeroepen, worden de elementen van `newData` gebruikt. Vul hier beneden de waardes in van enkele element van `newData`. Neem aan dat de code wordt uitgevoerd op een Linux/x86 machine met Little-Endian bytevolgorde. De antwoorden dienen in hexidecimaal formaat te worden gegeven. **Let op de bytevolgorde!** :

(a) `newData->start = 0x`_____
(b) `newData->raw[0] = 0x`_____
(c) `newData->raw[2] = 0x`_____
(d) `newData->raw[4] = 0x`_____

Er staat een bug in bovenstaande code, die de waarde van `newData->sense` kan beïnvloeden. Wat is deze bug, en in op welke manier beinvloedt hij de waarde van `newData->sense`?

Student: **Collegekaartnummer:**

Vraag 4

Bestudeer het volgende C programma:

```
main() {
  if (fork() == 0) {
    if (fork() == 0) {
      printf("3");
    } else {
      pid_t pid; int status;
      if ((pid = wait(&status)) > 0) {
        printf("4");
      }
    }
  } else {
    if (fork() == 0) {
      printf("1");
      exit(0);
    }
    printf("2");
  }
  printf("0");
  return 0;
}
```

Geef aan wat de een mogelijke uitvoer is van het programma, en wat niet, waarbij u mag aannemen dat alle processen hun *control-flow* geheel af kunnen maken . U heeft de keus uit de volgende 5 mogelijkheden. Omcirkel alleen mogelijke uitvoer. Geef voor de andere gevallen aan **waarom** deze uitvoer niet op kan treden.

A. 2030401 B. 1234000 C. 2300140 D. 2034012 E. 3200410