Student: **Collegekaartnummer:**

Tentamen Computersystemen voor AI programmeurs

baiCSAI3 2e jaar bachelor AI, 1e semester 25 juni - 15 juli 2009

vraag 1

U maakt deel uit van een programmeur's team dat probeert de snelste faculteit routine ter wereld te schrijven. Men was begonnen met een recursieve functie, maar men kwam er al snel achter dat een iteratieve versie veel sneller was:

```
int fact(int n)
{
    int i;
    int result = 1;
    for (i = n; i > 0; i--)
       result = result * i;
    return result;
}
```

Deze conversie reduceerde het aantal *cycles per element* (CPE) voor de functie van 63 naar 4. Echter, men is nog niet tevreden.

Één van de programmeurs had eens iets gehoord over *loop unrolling*. Deze programmeur probeerde dit toe te passen in de volgende versie:

```
int fact_u2(int n)
{
    int i;
    int result = 1;
    for (i = n; i > 0; i-=2)
           result = (result * i) * (i-1);
    return result;
}
```

Bij het testen kwam men erachter dat voor sommige waarden van *n* de nieuwe code 0 i.p.v. *n*! retourneert.

- **a.** Voor welke waarden van *n* verschillen de resultaten van functie van `fact` en `fact_u2` van elkaar?
- **b.** Geef aan hoe men door een kleine verandering in de code wel het correcte resultaat kan verkrijgen.
- **c.** Metingen aan de routine `fact_u2` geven nog steeds een CPE van 4.
  Kunt u dit verklaren?
- **d.** In een laatste poging verplaatst u de haakjes van de *inner loop* expressie.

  ```
  result = result * (i * (i-1));
  ```

  Plotseling heeft de *loop unrolling* wel effect, en daalt de CPE naar 2.5.
  Kunt u deze versnelling verklaren?

Student: **Collegekaartnummer:**

vraag 2

De MIPS processor in jouw Aibo heeft 32 generieke registers, terwijl men het in de IA32 met acht *integer (*en acht *floating-point* registers moet doen). Vier van de generieke MIPS-registers worden door de compiler gebruikt om functieargumenten te bewaren, terwijl men in de IA32 architectuur hiervoor de *stack* gebruikt. 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 MIPS architectuur van de Aibo, krijgen we de volgende *assembly code*:

```
swap_add:
      the set up of a new frame for the function
      .frame $sp,0,$31  # stackpointer and returnadress (reg31)
      lw    $6,0($4)    # load x with value of arg1 (reg4)
      lw    $3,0($5)    # load y with value of arg2 (reg5)
      addu  $2,$6,$3    # return value (reg2) is addition of x and y
      sw    $3,0($4)    # store y at adress of arg1
      sw    $6,0($5)    # store x at adress of arg2
      j     $31         # return
```

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

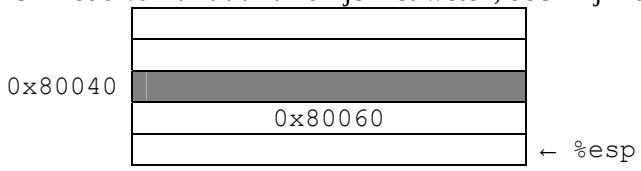
```
swap_add:
1     the set up of a new frame for the function
2     pushl %ebp        save old framepointer
3     movl %esp,%ebp    set new %ebp just after the stackpointer %esp
4     pushl %ebx        callee save
5     the body of the function
6     movl 8(%ebp),%edx
7     movl 12(%ebp),%ecx
8     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. Welk register wordt er bij de IA32 gebruikt om het eindresultaat te bewaren?
b. Waarom worden `%eax`,`%edx` en `%ecx` niet gesaved op de *stack*?
c. Is er bij de IA32 ook een register voor het *returnadress*?
d. Wordt de *stackpointer* verhoogd of verlaagd met 4 voor iedere `pushl`?
e. Als bij het aanroepen van de functie `swap_add` de stackpointer de waarde `0x80040` `had`, welke waarde krijgt `%ebp` dan op regel 3?
f. Welke waarde krijgt `%esp` op regel 4?
g. Vlak voor de aanroep naar `swap_add` staat 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.



Student: **Collegekaartnummer:**

vraag 3

Beantwoord de volgende vragen zowel voor de versie 2.95.3 en 4.1.2 van GCC op de Linux machines van de faculteit. Gebruik hiervoor de tools beschreven in Hoofdstuk 7.

- Hoeveel object files zitten opgeslagen in `libc.a` en `libm.a`?
- Genereert het commando `gcc -02` andere executeerbare code dan `gcc -02 -g`? Zo ja, wat is het verschil. Indien nee, zit het verschil dan elders?
- Welke *shared libraries* gebruikt de GCC driver?

vraag 4

Als laatste een mini buffer overflow attack. Download de volgende file.

http://csapp.cs.cmu.edu/public/ics/code/asm/bufbomb.c

en maak hier een executable van. Vind de hexidecimale string die zorgt dat `getbuf` de waarde `0xdeadbeaf` teruggeeft. Je zal hiervoor meer dan 12 bytes in moeten lezen. De extra bytes zullen de stack frames overschrijven, maar als je dit goed doet kun je de returnwaarde van getbuf veranderen. Zie voor meer details opgave 3.38 van het boek. Geef de oplossing, en een kort beschrijving hoe je het aangepakt hebt.