WS 2005/2006	Übungen zu 18.003 Bachelor Modul IP7 Rechnerstrukturen (RS) Teil 2	Aufgabenblatt 2.2
LV 18.004	J. Zhang	Abgabe: 30.01.06

Aufgabe 2.2.1 [20 Punkte] Anwendung des gcc-Compiler und dazugehöriger Tools:
Ziel dieser Aufgabe ist es, dass Sie selbst einmal den gcc C-Compiler und die zugehörigen Programme ausprobieren. Auf den meisten Linux-Systemen ist der gcc mit allen Tools vorinstalliert, so dass Sie die Befehle direkt ausführen können. Für Windows-Systeme könnten Sie die sogenannte Cygwin-Umgebung von www.cygwin.com herunterladen und installieren, die den gcc und die übrigen Tools enthält. Alternativ können Sie auch einen anderen C-Compiler verwenden. Sie müssen sich dann aber die benötigten Befehle und Optionen selbst heraussuchen.

Tippen Sie das folgende Programm ab (Sie können es auch als template.c von der Rechnerstrukturen-Webseite:
`http://tams-www.informatik.uni-hamburg.de/lehre/ws2005/vorlesungen/RS/` herunterladen) und tragen Sie Ihre Matrikelnummer ein. Übersetzen Sie das Programm und schauen Sie sich den Assembler- und Objektcode an:

```
/* template.c
 * Einfaches Programm zum Test des gcc-Compilers und der zugehörigen Tools.
 * Bitte setzen Sie in das Programm ihre Matrikelnummer ein und probieren
 * Sie alle der folgenden Operationen aus:
 *
 * C -> Assembler:   gcc -O2 -S template.c    (erzeugt template.s)
 * C -> Objektcode:  gcc -O2 -c template.c    (erzeugt template.o)
 * C -> Programm:    gcc -O2 -o template.exe template.c  (erzeugt template.exe)
 * Disassembler:     objdump -d template.o
 * Ausführen:        template.exe
 *
 * Natürlich können Sie auch einen anderen C-Compiler benutzen, müssen
 * sich dann aber die benötigten Befehle und Optionen selbst heraussuchen.
 */


#include <stdio.h>


int main( int argc, char** argv ) {
  int matrikelnummer = 7654321;

  printf( "Meine Matrikelnumer ist %ld\n", matrikelnummer );
  return 0;
}
```

Schicken Sie uns den erzeugten Assemblercode und die Ausgabe von objdump -d ein.

Aufgabe 2.2.2 [20 Punkte] x86-Assembler:
Angenommen, die folgenden Werte sind in den angegebenen Registern bzw. Speicheradressen gespeichert:

Register	Wert
%eax	0x00000100
%ecx	0x00000001
%edx	0x0000000C

Adresse	Wert
0x100	0x0000CAFE
0x104	0x000000AB
0x108	0x00000013
0x10c	0x00098700

Überlegen Sie sich, welche Adressen bzw. Register als Ziel der folgenden Befehle ausgewählt werden und welche Resultatwerte sich ergeben:

Befehl	Ziel (Adresse/Register)	Wert
addl %ecx, (%eax)		
subl %edx,4(%eax)		
imull $16,(%eax,%edx)		
incl 8(%eax)		
decl %ecx		
subl %edx,%eax		

Hinweis:
Beim gnu-Assembler steht der Zieloperand rechts, z.B. bewirkt
der Befehl `addl %ecx,12(%eax)`
die Operation: `MEM[0x10c] = 0x00098701`

Literaturhinweise zum x86-Assembler und den Adressierungsarten finden Sie auf der Rechnerstrukturen-Webseite des AB TAMS.

Aufgabe 2.2.3 [20 Punkte] x86-Assembler:

2.2.3.1 [5]: Wie kann man den Inhalt eines Registers auf Null setzen, wenn dafür kein separater Befehl zur Verfügung steht? Geben Sie x86-Beispielcode an, der ohne Immediate-Operand auskommt.

2.2.3.2 [15]: Wie kann man die Inhalte von zwei Registern vertauschen, ohne ein zusätzliches Register oder eine zusätzliche Speicherstelle zu verwenden? Geben Sie als Beispiel den x86 Assemblercode an, um die Werte in den Registern %eax und %edx zu vertauschen.

Hinweis:
Denken Sie auch über die XOR-Operation nach. Der x86 Befehl dafür lautet `xorl src, dest`.

Aufgabe 2.2.4 [10 Punkte] x86-Assembler:
Es gibt keinen Assemblerbefehl, der es erlaubt, den Programmzähler `%eip` direkt auszulesen. Schreiben Sie ein kurzes Assemblerprogramm, das den Programmzähler in das Register `%eax` kopiert.
Hinweis: Sie dürfen den Stack zur Zwischenspeicherung verwenden.

Aufgabe 2.2.5 [20 Punkte] x86-Assembler:
Eine klassische Aufgabe zur Demonstration einfacher numerischer Operationen ist die Umrechnung zwischen Grad Celsius C und Grad Fahrenheit F nach der Formel $C = (F - 32) * (5/9)$.

Da im bisher eingeführten Befehlssatz für unseren x86-Prozessor noch kein Befehl für die Division enthalten ist, nähern wie den Umrechnungsfaktor $5/9$ durch den Wert $5/9 \approx 142/256$ an, der sich zum Beispiel mit einer Multiplikation und einem Rechts-Shift effizient umsetzen lässt.

Schreiben Sie x86-Assemblercode für eine Funktion `int celsius( int fahrenheit )`, die ihr Argument (in Grad Fahrenheit), wie in der Vorlesung erläutert, auf dem Stack übergeben bekommt und ihren Rückgabewert entsprechend der Konvention im Register %eax hinterläßt. Nach Ausführung der Funktion sollen die relevanten Datenregister wieder ihren vorherigen Wert enthalten. Bedenken Sie dabei, daß laut Konvention die Register %eax, %edx und %ecx als "caller save" klassifiziert sind. Die Inhalte der für die Berechnung benötigten Register müssen also von der Funktion teilweise ebenfalls auf den Stack gerettet und am Ende wiederhergestellt werden.

Aufgabe 2.2.6 [40 Punkte] Rekursion:

2.2.6.1 [5]: Analysieren Sie das folgende in C-Syntax notierte Programm, und geben Sie einen mathematischen Ausdruck für das Ergebnis der Funktion an:

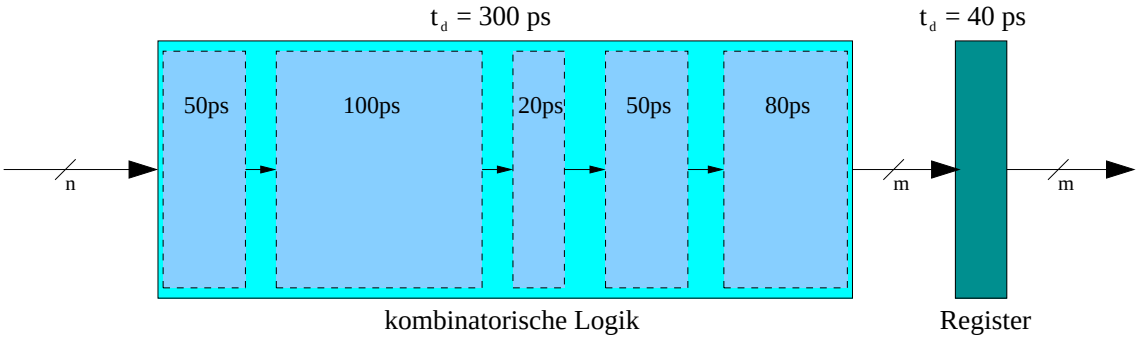
```
int wsdw(int a, int b)
{
    if (b == 0)
                return 1;
    else
                return wsdw(a, b-1) * a;
}
```

Für die Variablen a, b gilt die Beschränkung: a,b $\geq$ 0.

2.2.6.2 [20]: Schreiben Sie ein rekursives x86-Assemblerprogramm, das die Funktion des o.a. C-Programms erfüllt. Die Parameter a und b werden durch die rufende Prozedur/Funktion auf dem Stack hinterlegt. Der Resultatwert soll gemäß Konvention wieder in dem Register `%eax` hinterlegt werden.

2.2.6.3 [15]: Fertigen Sie eine Skizze des Stacks bei maximaler Verschachtelungstiefe an. Verwenden sie dabei für den ersten Aufruf von `wsdw` die Werte: a = 4 und b = 3 für die Parameter.

Aufgabe 2.2.7 [20 Punkte] Pipelining:



2.2.7.1 [10]: Gegeben sei oben gezeigte Funktionseinheit. Die Zeitangabe $t_d = 300ps$ sagt aus, dass die Reaktion auf einen Signalwechsel am Eingang des kombinatorischen Logikblockes nach $300ps$ am Ausgang erscheint. Man könnte auch formulieren, dass eine Operation $300ps$ benötigt. Für das Register soll der Einfachheit halber lediglich eine Zeitbedingung eingeführt werden. Die Angabe $t_d = 40ps$ soll hier bedeuten, dass das Register $40ps$ benötigt, um den Eingabewert zu verarbeiten.
Erläutern Sie die Begriffe Durchsatz und Verweildauer (Latenzzeit) und berechnen Sie die Werte für das o.g. Beispiel.

2.2.7.2 [10]: Der kombinatorische Logikblock der obigen Schaltung lässt sich, wie in der Skizze angegeben, in kleinere Logikblöcke aufteilen, wobei die Reihenfolge beizubehalten ist. Zeigen Sie, wie durch Einführung von Pipelineregistern (mit gleichen Zeitbedingungen wie das vorhandene Register) der Durchsatz maximiert werden kann. Geben Sie für diese Konfiguration Durchsatz und Verweildauer an.

Aufgabe 2.2.8 [30 Punkte] Pipeline Hazards:
Erläutern Sie möglichst kurz und prägnant folgende Fragen.

2.2.8.1 [10]: Was verstehen Sie unter einem *Pipeline Hazard*?

2.2.8.2 [10]: Pipeline Hazards lassen sich grundsätzlich in Daten- und Kontrollfluss-Hazards unterscheiden. Erläutern Sie einen Daten-Hazard an einem kleinen von Ihnen konstruierten Beispiel anhand einer vierstufigen Pipeline.

2.2.8.3 [10]: Nennen Sie zwei Strategien zur Vermeidung von Hazards und erläutern Sie diese kurz.