Hochschule Konstanz
Fakultät Informatik

Prof. Dr.-Ing. Michael Blaich
Robotik und Künstliche Intelligenz

Übung Rechnerarchitekturen AIN 2 SoSe2025

2. Assemblerprogrammierung und MIPS Konventione

Die Abgabe erfolgt durch Hochladen der Lösung in Moodle.
Zusätzlich wird die Lösung in der Übung nach dem Abgabetermin stichprobenartig kontrolliert.

Bearbeitung in Zweier-Teams

Team-Mitglied 1: Alexander Engelhardt

Team-Mitglied 2: Timo Drexler

Hochschule Konstanz
Fakultät Informatik

Prof. Dr.-Ing. Michael Blaich
Robotik und Künstliche Intelligenz

Aufgabe 2.1 Daten und Arrays

Implementieren Sie in MARS ein Assemblerprogramm, das

1. im Speicher eine Zahl `n` und ein Array `A` mit `n` Integer-Werten anlegt und
2. die Summe der `n` Werte im Array bestimmt und in das Register `$v0` schreibt

Wählen Sie als Beispiel `n=6`.

Verwenden Sie zum Anlegen der Werte im Speicher die Assembler Direktive `.word`. Laden Sie die Variable `n` sowie die Adresse des Arrays mit den Pseudo-Instruktionen `li Register, Label` bzw. `la Register, Label`. (Kapitel 2.6 bzw. Hilfe in MARS)

Aufgabe 2.2 Erste Prozedur

Implementieren Sie eine Prozedur (Label: `ISODD`) mit einem Integer-Wert `x` als Argument. Die Prozedur soll den Wert `1` zurückliefern, falls `x` ungerade ist und den Wert `0` zurückliefern, falls `x` gerade ist.
Tipp: Verwenden Sie die Instruktion `andi`.

Implementieren Sie eine zweite Prozedur (Label:`ISEVEN`), die komplementär zur ersten Prozedur arbeitet. Die Prozedur soll den Wert`1` zurückliefern, falls`x` gerade ist und den Wert`0` zurückliefern, falls `x` ungerade ist. Implementieren Sie diese Funktion, indem Sie die Funktion `ISODD` aufrufen und das Ergebnis invertieren.

Testen Sie die beiden Funktionen in MARS, in dem Sie sie nacheinander aufrufen und das Ergebnis in die Register `$s1` und `$s2` speichern.

Achten Sie auf die MIPS-Konventionen zur Implementierung von Prozeduren.

Aufgabe 2.1

```
.data
.align 2
n: .word 6
A: .space 24

.text
.globl main

main:
	li $t6, 0 # Sum
	lw $t0, n # n
	la $t1, A # Array
	li $t2, 0 # Index

	loop:

		bge $t2, $t0, end # Checks whether the index is at the end of the Array
		addi $t3, $t2, 1
		mul $t3, $t3, $t0

		sw $t3, 0($t1)

		add $t6, $t6, $t3 # Sum
		addi $t1, $t1, 4 # Moves pointer of the address by 4 bytes
		addi $t2, $t2, 1

		j loop

	end:

	li $v0, 1
	move $a0, $t6
	syscall
```

Aufgabe 2.2

```
.data

.text
.globl main

main:
        addi $t0, $zero, 11 # x
        jal ISODD
        jal ISEVEN



ISODD:
        andi $s1, $t0, 1
        j end

        end:
                jr $ra

ISEVEN:
        jal ISODD
        seq $s2, $s1, 0

        j end
```

Aufgabe 2.3 Prozeduren und Arrays

1. Erweitern Sie den Code aus Aufgabe 2.2 und legen Sie ein zweites Array `B` an, das zu Beginn mit `n` Nullen gefüllt ist.
2. Implementieren Sie eine Prozedur, die alle geraden Elemente eines Arrays `A` in ein Array `B` schreibt und die Anzahl der geraden Elemente zurückliefert. Argumente der Prozedur sind die Adressen der Arrays `A` und `B` sowie die Anzahl `n` der Elemente im Array. Der C-Code der Funktion ist unten gegeben. Halten Sie sich bei der Implementierung des Assembler-Code strikt an die Vorgaben der C-Funktion sowie die MIPS Konventionen.

```
int evenElem(int A[], int B[], int n) {

    int i=0;
    int j=0;
    while (i<n) {
        if (isEven(A[i])) {
            B[j]=A[i];
            j++;
        }
        i++;
    }
    return j;
}
```

3. Testen Sie ihre Prozedur mit dem Array `A=[3, 4, 6, 8, 11, 13]`.

Aufgabe 2.3 Prozeduren und Arrays

1. Erweitern Sie den Code aus Aufgabe 2.2 und legen Sie ein zweites Array `B` an, das zu Beginn mit `n` Nullen gefüllt ist.
2. Implementieren Sie eine Prozedur, die alle geraden Elemente eines Arrays `A` in ein Array `B` schreibt und die Anzahl der geraden Elemente zurückliefert. Argumente der Prozedur sind die Adressen der Arrays `A` und `B` sowie die Anzahl `n` der Elemente im Array. Der C-Code der Funktion ist unten gegeben. Halten Sie sich bei der Implementierung des Assembler-Code strikt an die Vorgaben der C-Funktion sowie die MIPS Konventionen.

```
int evenElem(int A[], int B[], int n) {

    int i=0;
    int j=0;
    while (i<n) {
        if (isEven(A[i])) {
            B[j]=A[i];
            j++;
        }
        i++;
    }
    return j;
}
```

3. Testen Sie ihre Prozedur mit dem Array `A=[3, 4, 6, 8, 11, 13]`.

Aufgabe 2.3

```
.data
newline : .asciiz "\n"
isO : .asciiz "isodd:\n"
isE : .asciiz "iseven:\n"
sizeB : .asciiz "Size of B:\n"
iOfA : .asciiz "A[i]:\n"
tab : .asciiz "\t"
itemsB : .asciiz "Values of B:\n"

n: .word 6
A: .word 3, 4, 6, 8, 11, 13
B: .space 24

.text
.globl main
main:
        li $t0, 6
        lw $t1, n       # size
        la $t2, A       # Array A
        la $t3, B       # Array B
        add $t4, $t4, $zero             # index i
        li $t5, 0       # index j
        li $t6, 4
        li $t8, 0
        jal fillWzeros
        jal reset
        li $v0, 4
        la $a0, iOfA
        syscall
        jal evenElem

evenElem:
        bge $t4, $t1, printBArray       # end => i>=n

        lw $s2, 0($t2)

        li $v0, 1
        move $a0, $s2
        syscall

        li $v0, 4
        la $a0, newline
        syscall

        jal ISEVEN

increment:
        addi $t4, $t4, 1
        add $t2, $t2, $t6
        j evenElem

fillWzeros:
        bge $t4, $t1, reset
        sw $zero, 0($t3)

        addi $t4, $t4, 1
        add $t3, $t3, $t6
        j fillWzeros

reset:
        la $t3, B
        li $t4, 0
        li $t5, 0
        jr $ra

printBArray:
        jal reset
        li $v0, 4
        la $a0, itemsB # Items of B
        syscall

        loop:
                bge $t4, $t1, end
                lw $t7, 0($t3)
                li $v0, 1
                move $a0, $t7 # B[j]
                syscall
                li $v0, 4
                la $a0, tab # \t
                syscall

                addi $t4, $t4, 1
                add $t3, $t3, $t6
                j loop
ISODD:
        andi $s1, $s2, 1

        jr $ra
ISEVEN:
        jal ISODD
        seq $t9, $s1, 0
        bne $t9, 1, increment
        addi $t8, $t8, 1
        sw $s2, 0($t3)
        add $t3, $t3, $t6
        j increment
end:
        li $v0, 4
        la $a0, newline
        syscall
        la $a0, sizeB # Size of B
        syscall
        li $v0, 1
        move $a0, $t8 # sizeof(ISEVEN(B[j]))
        syscall
```

Aufgabe 2.4 Rekursive Funktion

Das Produkt zweier natürlicher Zahlen n*m lässt sich rekursiv wie folgt berechnen:

$$n*m=m*(n-1)+m=(m-1)*(n-1)+n+m-1$$

Das Programm `rekmul.asm` berechnet das Produkt zweier natürlicher Zahlen rekursiv. Es steht in Moodle zum Download zur Verfügung und lässt sich leicht mittels des MIPS-Simulators Mars ausführen.

Beantworten Sie die folgenden Fragen zu diesem Programm:

a) Was wird im Allgemeinen im Register `$ra` gespeichert? Erläutern Sie in diesem Zusammenhang die Zeilen 26 und 50 (`jal rekmul`), sowie Zeile 58 (`jr $ra`).
b) An welcher Zeile wird das Programm nach Ausführung von Zeile 58 fortgeführt?

Aufgabe 2.5 Implementierung einer rekursiven Funktion

Implementieren Sie die folgende rekursive Funktion als Prozedur in Assembler:

$$f(n, k) = \begin{cases} n + k + 5 \text{ für } k - n > 7 \\ f(n - 1, \max(8, g(k))) \text{ sonst} \end{cases}$$

Die Funktion $g(k)$ befindet sich an der Speicheradresse mit Label `G:` und Sie können davon ausgehen, dass die Funktion entsprechend der MIPS-Konventionen implementiert ist.

- Halten Sie sich bei den Implementierungen an die MIPS-Konventionen.
- Verwenden Sie keine Pseudo-Instruktionen außer *move*
- Verwenden Sie für bedingte Sprünge nur die Instruktionen *beq* und *bne*
- Unten finden Sie die Funktion in C. Sie müssen den C Code nicht eins-zu-eins nach Assembler übersetzen, sondern könne auch eine eigene Implementierung finden.
- Anbei finden Sie ein Prozedur `G`, mit der Sie ihre Implementierung testen können. Die Testprozedur liefert *G(k)=200+k* und überschreibt dabei alle Register außer den s-Registern.

```
int f(int n, int k) {
if (k-n>7) {
      return n+k+5;
}else{
      return f(n-1,max(8,g(k)));
}
```

Aufgabe 2.4

a) Im Allgemeinen wird in $ra die Return Address, also die Adresse, welche zurückgegeben werden soll, gespeichert, wenn das Programm abgeschlossen ist.
Zeile 26 springt zu rekmul und speichert die Adresse des Programms von Zeile 26 in der Return Address.
Zeile 50 setzt die Return Address auf rekmul und wird dann im Stack Pointer gespeichert.

Bei return wird die Adresse so häufig um 4 erhöht, wie sie bei rekmul um 4 veringert wurde
(so häufig wie die zweite eingegebene Zahl), springt jeweils
zu der Zeile vor j return und als letztes zu der letzten eingespeicherten Return Address bei jal rekmul
unter main (Z. 26).

b) * () => {Zeile 28;}, Mem(0x004000a0->0x00400040)

Aufgabe 2.5

```
.data

.text
.globl main

main:
        addi $s0, $zero, 3 # n
        addi $s1, $zero, 10 # k
        addi $s2, $zero, 7 # if condition

        jal f

        li $v0, 10
        syscall

f:
        # if (k-n > 7)
        addi $sp, $sp, -4
        sw $ra, 0($sp)
        sub $t0, $s1, $s0
        slt $at, $s2, $t0
        beq $at, $zero, else

        # return n + k + 5
        add $s0, $s0, $s1
        addi $s0, $s0, 5
        li $v0, 1
        move $a0, $s0
        syscall
        j return


else:

        addi $s0, $s0, -1 # n-1
        addi $t1, $t1, 8
        j max


max:
        # max(8, g(k))
        move $s3, $t1 # 8
        jal G
        add $a1, $a1, $s1 # 200 + k
        slt $at, $s3, $a1
        bne $at, $zero, g_k_larger
        move $s1, $s3
        lw $ra, 0($sp)
        addi $sp, $sp, 4
        jal f

g_k_larger:
        move $s1, $a1
        lw $ra, 0($sp)
        jal f

return:
        lw $ra, 0($sp)
        addi $sp, $sp, 4
        jr $ra


# willkürliche Prozedur G (überschreibt alle "ungesicherten" Register mit dem Wert $a0+200, liefert auch $a0+200 zurück)
G:      addi $a0,$a0,100
        addi $a1,$a0,100
        addi $a2,$a0,100
        addi $a3,$a0,100
        addi $t0,$a0,100
        addi $t1,$a0,100
        addi $t2,$a0,100
        addi $t3,$a0,100
        addi $t4,$a0,100
        addi $t5,$a0,100
        addi $t6,$a0,100
        addi $t7,$a0,100
        addi $t8,$a0,100
        addi $t9,$a0,100
        addi $v0,$a0,100
        addi $v1,$a0,100
        jr $ra
```