Ministry of Higher Education and Scientific Research		
Salahaddin University / Erbil		***Subject: Compilers***
College of Engineering		***Time: 90 Minutes***
Dept. of Software Engineering	***2nd Term Exam 2011-2012***	***Lecturer: Amanj Sherwany***
		Date: 6 May 2012

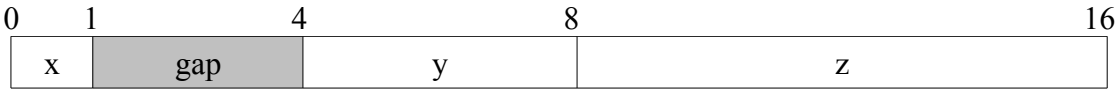
The highest obtainable mark is 100, the minimum passing mark is 50

Q1: (16 points)

For each of the following data structures, show how their instances are represented in the memory of a 32-bit machine, state their sizes in bytes, and the possible needed alignments:

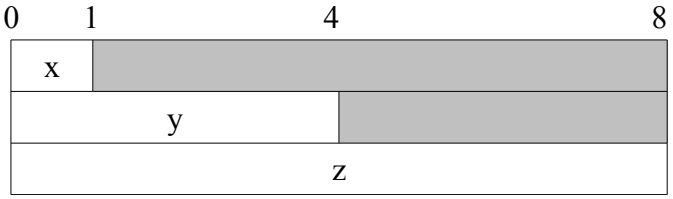
- `struct element{char x; int y; double z;}`

Answer:



Size = 16 and Alignment = 8

- `union element{char x; int y; double z;}`



Size = 8 and Alignment = 8

* * *

Q2: (30 points)

Translate the following function to assembly:

```
int factorial(int n){
     if(n == 0)
          return 1;
     if(n == 1)
          return 1;
     return (n * factorial(n-1));
}
```

The target machine is a 32-bit RISC with 4-byte integers, a stack pointer register SP, a return

address register RA, eight general purpose registers R0 to R7, and the following instruction set:

```
goto label
if reg < opnd goto label     (==, >=, etc)
move dst, opnd               assign opnd to dst
add dst, reg1, opnd2         assign reg1 plus opnd2 to dst
                             ditto for sub, mul, and, or,
                             leftshift, etc
load dst, (reg1 + opnd2)     read integer from memory at
                             reg1+opnd2
store (reg1 + opnd1), opnd3  write integer (opnd3) to memory
                             at reg1+opnd2
call label                   set RA to next instruction then
                             jump to label
return                       jump to the address in RA
```

Each *dst* must be a register, and each *opnd* must be a register or an integer constant. The function call conventions are that parameters are passed in register *R0,* the return value is returned in *R1,* and a function call may destroy any general-purpose register and *RA*. The stack grows from high to low addresses, and *SP* should always point to the lowest word of the current stack frame.
Apart from the statements and expressions, include the code for setting up the stack frame, storing registers in the stack frame , and fetching registers from the stack frame.

Answer:

```
     .text
     .global
factorial:

# Memory layout:
# ==============
# 1 * 4: old FP
# 2 * 4: RA
# 3 * 4: n


# INPUT: R0
# OUTPUT: R1

     move R7, FP
     move FP, SP
     sub SP, SP, 12
```

```
     store FP - 1 * 4, FP
     store FP - 2 * 4, RA
     store FP - 3 * 4, R0

     if R0 != 0 goto Lelseif
     move R1, 1
     goto Lreturn
Lelseif:
     if R0 != 1 goto Lelse
     move R1, 1
     goto Lreturn
Lelse:
# R0 already contains n
# decrement n
     sub R0, R0, 1
     call factorial
     load R2, FP - 3 * 4
     mul R1, R1, R2
Lreturn:
     load RA, FP - 2 * 4
     load FP, FP - 1 * 4
     add SP, SP, 12
     return
```

* * *

Q3: (30 points)
Translate the following C-code to RTL/quadruples:

```
int multiply(int x, int y){
     int result = 0;
     for(int i = y; i > 0; i--){
          result += x;
     }
     return result;
}
```

***Answer*:**

```
multiply(temp_x, temp_y):
            temp_result = 0
     Linit: temp_i = temp_y
            goto Lcond
     Lbody: temp_result = temp_result + temp_x
```

```
Lstep: temp_i = temp_i - 1
Lcond: if temp_i > 0 goto Lbody
Lnext: temp_RV = temp_result
       goto Lreturn
```

* * *

Q 4: (24 points)

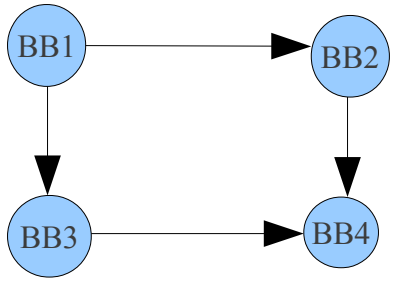
For the following intermediate code representation:

- Find the basic blocks **(6 points)**
- Draw the Control-Flow Graph (CFG) **(4 points)**
- Calculate the Liveness (LIVEIN and LIVEOUT) for each instruction **(10 points)**
- Calculate the Liveness (LIVEIN and LIVEOUT) for each basic block **(4 points)**

```
I1.      a := b + 3
I2.      if a == 10 goto L1
I3.      f := a - b
I4.      goto L2
I5. L1:  f := a + b
I6. L2:  return f
```

	BBs	USE	DEF	LIVEIN	LIVEOUT
I1. a := b + 3 I2. if a == 10 goto L1	BB1	b a	a -	b a, b	a, b a, b
I3. f := a - b I4. goto L2	BB2	a, b -	f -	a, b f	f f
I5. L1: f := a + b	BB3	a, b	f	a, b	f
I6. L2: return f	BB4	f	-	f	-

BBs	LIVEIN	LIVEOUT
BB1	b	a, b
BB2	a, b	f
BB3	a, b	f
BB4	f	-



Good Luck