Compilers: Assignment #3

Due on Sunday, December 3, 2016

Genafl: Task 2

Mirza Hasanbasic

Indhold

Task 1

Det skal siges, at jeg har vedhæftet en .txt fil, så du lettere kan afprøve koden som er skrevet. Det der står i dokumentet er til det visuelle.

a)

Vi har

```
vtable = [a → v, b → w];

while (b != 0) && (a/b != 0)
        if b < a then {a := a − b}
                        else {b := b − a}
```

Hvor intermediate koden er

```
t_0 = v
t_1 = w
LABEL LoopStart
IF t_1 != 0 then NEXT0 else END (Brug rigtig syntax i tex filen !=)
LABEL NEXT0
t_2 = t_0 mod t_1
IF t_2 != 0 then NEXT1 else END
LABEL NEXT1
t_3 = t_1 − t_0
IF t_3 < 0 then NEXT2 else NEXT3
LABEL NEXT2
t_0 = t_0 − t_1
GOTO LoopStart
LABE NEXT3
t_1 = t_1 − t_0
GOTO LoopStart
LABEL END
```

og MIPS koden vil være

Task 1 continued on next page...

```
1  .data
2          a: .word 8
3          b: .word 33
4  .text
5  main:
6  lw $t0, a                        \# load 8
7  lw $t1, b                        \# load 33
8  LoopStart:                       \# LABEL
9  beq $t1, $0, END                 \# Checking if t1 == 0
10 div $t0, $t1                     \# dividing to get modulus
11 mfhi $t2                         \# Getting the remainder, moving to $t2
12 beq $t2, $0, END                 \# checking if t2 == 0
13 sub $t3, $t1, $t0                \# t3 = t1 - t0
14 bgez $t3, ELSE                   \# t3 >= 0
15 sub $t0, $t0, $t1                \# first then statement a = a - b
16 j LoopStart                      \# jumping to loopstart
17 ELSE:                               \# Now else statement
18 sub $t1, $t1, $t0                \# b = b - a
19 j LoopStart
20 END:
21                                  \# tinyurl.com/neve79o
22 li $v0, 1                     \# printer udregnet variable ud.
23 add $a0, $t0, $zero
24 syscall
25
26 li $v0, 11
27 li $a0, 10
28 syscall
29
30 li $v0, 1
31 add $a0, $t1, $zero
32 syscall
```

b)

```
1 Li t0, x
2 Li t1, y
3 Li t2, 1
4 Slt t3, t1, t0
5 Slt t4, t3, t2
```

Task 2

a)

i	succ[i]	gen[i]	kill[i]
1	2		
2	7,3	a,b	
3	4		
4	5	a	t
5	6	b	a
6	7	t	b
7	8		
8	9		z
9	10	b,a	b
10	1,11	b,z	
11			
12		a	

b)

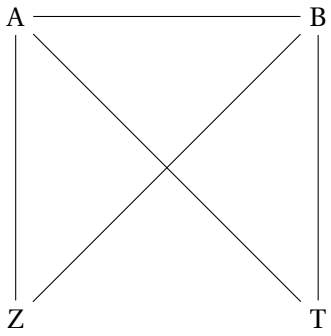
FIX:

Initial			Iteration 1		Iteration 2		Iteration 3	
i	out[i]	in[i]	out[i]	in[in]	out[i]	in[in]	out[i]	in[in]
1			a,b	a,b	a,b	a,b	a,b	a,b
2			a,b	a,b	a,b	a,b	a,b	a,b
3			a,b	a,b	a,b	a,b	a,b	a,b
4			b,t	a,b	b,t	a,b	b,t	a,b
5			a,t	b,t	a,t	b,t	a,t	b,t
6			a,b	a,t	a,b	a,t	a,b	a,t
7			a,b	a,b	a,b	a,b	a,b	a,b
8			a,b,z	a,b	a,b,z	a,b	a,b,z	a,b
9			b,z,a	a,b,z	a,b,z	a,b,z	a,b,z	a,b,z
10			a	a, b,z	a,b	a,b,z	a,b	a,b,z
11			a	a	a	a	a	a
12				a		a		a

Task 2 continued on next page...

c)

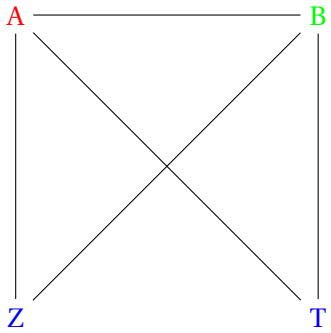
i	left	interferes with
4	t	a,b
5	a	b,t
6	b	a,t
8	z	a,b
9	b	a,z



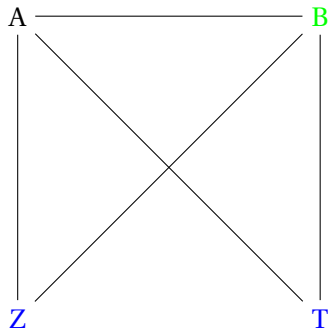
d)

FIX:

node	Neighbours	color
a		1
b	a	2
t	a, b	3
z	a,b	3



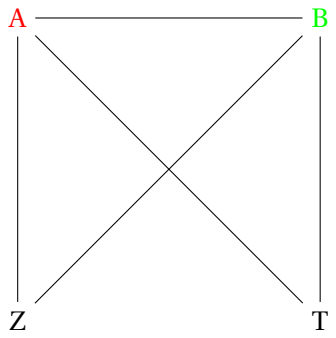
e)




```
1  gcd(a,b) {
2      M[address_a] := a
3      LABEL start
4      a_i := M[address_a]
5      IF a_i < b THEN next ELSE swap
6      LABEL swap
7      a_i := M[address_a]
8      t := a_i
9      a_i := b
10     M[address_a] = a_i
11     b := t
12     LABEL next
13     z := 0
14     a_i = M[address_a]
15     b := b mod a_i
16     IF b = z THEN end ELSE start
17     LABEL end
18     a := M[address_a]
19     RETURN a
20 }
```

og dette vil være med 2 registre

FIX:



Task 3

a)

```
1   char *y = ( char *) malloc ((1 + (n - 1) / 4) * 4);
2   int y_len = 0;
3   int i = 0;
4   while (i < n){
5   if(p(x[i])){
6   y[ y_len ] = x[i];
7   y_len ++;
8   }
9   i++;
10  }
11  return y;
```

b)

```
1   lw R_len , 0( regx ) // Load number of elements
2   move regy , R_HP // Set start of result
3   array
4   sll R_tmp , R_len , 2
5   addi R_tmp , R_tmp , 4
6   add R_HP , R_HP , R_tmp // Set HP to after result
7   addi R_it_x , regx , 4 // Initialise x iterator
8   addi R_it_y , regy , 4 // Initialise y iterator
9   move R_i , $0 // Initialise logic
10  iterator
11  loop_beg :
12  sub R_tmp , R_i , R_len // While condition
```

Task 3 continued on next page...

```
13      bgez R_tmp , loop_end // End loop when i= length
14      lb R_tmp , 0( R_it_x ) // Get element
15      R_tmp2 = CALL f( R_tmp ) // Call function f on
16      argument R_tmp and put the result in R_tmp2
17      beq R_tmp2 , $0, continue // Skip if predicate
18      failed
19      sb R_tmp , 0( R_it_y ) // Save element in y
20      addi R_it_y , R_it_y , 1 // increment y
21      continue :
22      addi R_it_x , R_it_x , 1 // increment x
23      addi R_i , R_i , 1 // increment logic
24      iterator
25      j loop_beg
26      loob_end :
```