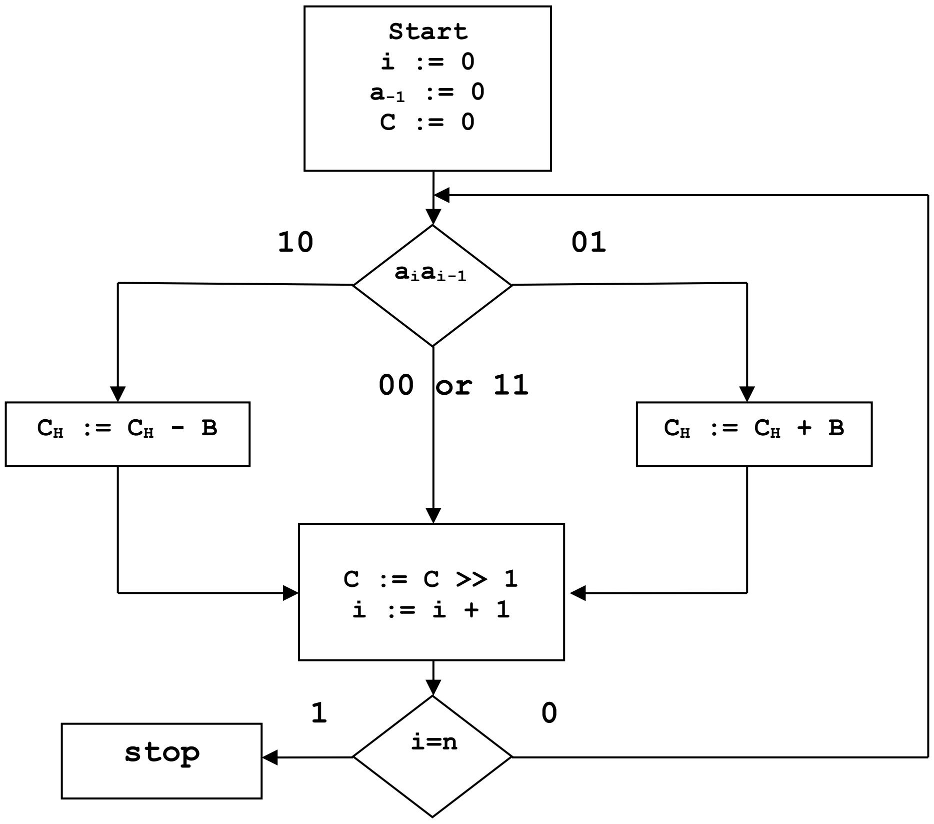
Homework #2

Assigned: 29/03/2023
Due: 09/04/2023
Rebah Özkoç 29207

1. (**30 pts**) Consider Booth's algorithm below for multiplying integers including signed ones (two' complement).



$C = A \times B$, C is $2n$-bit, A and B are n-bit registers. C_H is upper n-bit of C register.

Using Booth's algorithm with $n = 4$, do the following multiplication operations:

$6 \times 5 = ?$
$-5 \times -4 = ?$
$-4 \times 7 = ?$ (**10 pts each**)

Shows the steps of the algorithm.

6 × 5

i	B	A	a_ia_{i-1}	operation	C
0	0101	0110	00	c := c >> 1	00000000
1	0101	0110	10	C_H := C_H - B	10110000
				c := c >> 1	11011000
2	0101	0110	11	c := c >> 1	11101100
3	0101	0110	01	C_H := C_H + B	00111100
				c := c >> 1	00011110

-5 × -4

i	B	A	a_ia_{i-1}	operation	C
0	1100	1011	10	C_H := C_H - B	01000000
				c := c >> 1	00100000
1	1100	1011	11	c := c >> 1	00010000
2	1100	1011	01	C_H := C_H + B	11010000
				c := c >> 1	11101000
3	1100	1011	10	C_H := C_H - B	00101000
				c := c >> 1	00010100

-4 × 7

i	B	A	a_ia_{i-1}	operation	C
0	0111	1100	00	c := c >> 1	00000000
1	0111	1100	00	c := c >> 1	00000000
2	0111	1100	10	C_H := C_H - B	10010000
				c := c >> 1	11001000
3	0111	1100	11	c := c >> 1	11100100

2. (**30 pts**) Consider the following C language statements.

- `f = -g + h + B[i]+ C[j]` (**10 pts**)
- `f = A[B[g]+ C[h] + j]` (**20 pts**)

Assume that the local variables `f`, `g`, `h`, `i` and `j` of integer types (32-bit) are assigned to registers `$s0`, `$s1`, `$s2`, `$s3` and `$s4` respectively. Assume also the base address of the arrays `A`, `B` and `C` of integer types are in registers `$s5`, `$s6` and `$s7`, respectively (i.e. `$s0` → `f`, `$s1` → `g`, `$s2` → `h`, `$s3` → `i`, `$s4` → `j`, `$s5` → `&A[0]` , `$s6` → `&B[0]`, `$s7` → `&C[0]`).

For the C statements above, provide MIPS Assembly instructions.

Solution:

```
sub $s0, $s2, $s1      # Add h – g and store at f
add $t0, $s3, $s3      # Multiply i by 2 to get the byte offset
add $t0, $t0, $t0      # Multiply i by 4 to get the byte offset
add $t1, $t0, $s6      # Add the word offset to the base address of the array B and store at $t1
lw $t2, 0($t1)         # Load the value at B[i] into $t2
add $s0, $s0, $t2      # Add B[i] and f and store at f
add $t0, $s4, $s4      # Multiply j by 2 to get the byte offset
add $t0, $t0, $t0      # Multiply j by 4 to get the byte offset
add $t1, $t0, $s7      # Add the word offset to the base address of the array C and store at $t1
lw $t2, 0($t1)         # Load the value at C[j] into $t2
add $s0, $s0, $t2      # Add C[j] and f and store at f
# Second part starts
add $t0, $s1, $s1      # Multiply g by 2 to get the byte offset
add $t0, $t0, $t0      # Multiply g by 4 to get the byte offset
add $t1, $t0, $s6      # Add the word offset to the base address of the array B and store at $t1
lw $t2, 0($t1)         # Load the value at B[g] into $t2
add $t0, $s2, $s2      # Multiply h by 2 to get the byte offset
add $t0, $t0, $t0      # Multiply h by 4 to get the byte offset
add $t1, $t0, $s7      # Add the word offset to the base address of the array C and store at $t1
lw $t3, 0($t1)         # Load the value at C[h] into $t3
add $t2, $t2, $t3      # Add B[g] and C[h] and store at $t2
add $t2, $t2, $s4      # Add j to B[g] + C[h] and store at $t2
add $t0, $t2, $t2      # Multiply B[g]+ C[h] + j by 2 to get the byte offset
add $t0, $t0, $t0      # Multiply B[g]+ C[h] + j by 4 to get the byte offset
add $t1, $t0, $s5      # Add the word offset to the base address of the array A and store at $t1
lw $t2, 0($t1)         # Load the value at A[B[g]+ C[h] + j] into $t2
add $s0, $t2 $zero     # Assign $t2 to f
```

3. (**20 pts**) Consider the following C++ code sequence

```
t = A[0];
for (i=0; i < 5; i++)
     A[i] = A[i+1];
A[5] = t;
```

Write an Assembly language program for MIPS processor, assuming that base address of the array **A** is in **$s0**.

Solution:

```
lw $t3, 0($s0)              # Load the value at A[0] into $t3 (variable t)
add $t0, $zero, $zero       # i = 0


loop_start:
# Check i < 5 if not exit the loop
addi $t1, $zero, 5          # Load 5 into $t1
slt $t2, $t0, $t1           # $t2 = 1 if $t0 < $t1 (i < 5), otherwise $t2 = 0
beq $t2, $zero, loop_exit   # If $t2 == 0 (i >= 5), exit the loop

# Calculate the address of A[i+1] and load its value into $t4
addi $t4, $t0, 1            # $t4 = i + 1
add $t4, $t4, $t4           # Multiply by 2 to get the byte offset
add $t4, $t4, $t4           # Multiply by 4 to get the byte offset
add $t4, $s0, $t4           # Add address of A to the offset to get the address of A[i+1]
lw $t4, 0($t4)              # Load A[i+1] into $t4

# Calculate the address of A[i] and store the value of A[i+1] into it
add $t5, $t0, $t0           # Multiply i by 2 to get the byte offset
add $t5, $t5, $t5           # Multiply i by 4 to get the byte offset
add $t5, $s0, $t5           # Add base address of A to get the address of A[i]
sw $t4, 0($t5)              # Store A[i+1] into A[i]
```

```
# Increment i by one and jump back to loop_start
addi $t0, $t0, 1
j loop_start

loop_exit:
# Store t (A[0]) into A[5]
addi $t6, $zero, 20         # Load 20 (5 * 4) into $t6 (byte offset for A[5])
add $t6, $s0, $t6           # Add base address of A to get the address of A[5]
sw $t3, 0($t6)              # Store t (A[0]) into A[5]
```

4. (**20 pts**) Consider the following assembly program in MIPS, which implements a subroutine named “`func`”

```
# In: $a0 (an unsigned integer)
# Out: $v0

func:        add $s0, $zero, $a0
             addi $v0, $zero, 1

func loop:   beq $s0, $zero, func return
             add $v0, $v0, $s0
             addi $s0, $s0, -1
             j func loop

func return:jr $r
```

a. (**15 pts**) What is the C/C++ equivalent of this code? Assume that the functions argument is an unsigned integer `n` in the C/C++ version of the function. What is the value returned by this function if it is called with `$a0=5`?

```
int func (unsigned int a0){
     unsigned int s0 = a0;
     unsigned int v0 = 1;

     while (s0 != 0){
          v0 = v0 + s0;
          s0 = s0 - 1;
     }
     return v0;
}
```

If the function is called with $a0 = 5 these things happen in orderly:
$s0 = 5
$v0 = 1
$v0 = 6
$s0 = 4
# jump back to func_loop
$v0 = 10
$s0 = 3
# jump back to func_loop
$v0 = 13
$s0 = 2
# jump back to func_loop
$v0 = 15
$s0 = 1
# jump back to func_loop
$v0 = 16
$s0 = 0
# jump back to caller
Function returns 16 as the value.

b. (**5 pts**) The code in the box above contains an unconventional use of registers that violates the MIPS procedure calling convention. This may result in an error. Find this unconventional use of registers and show how it should be fixed.

Solution:
$s0 is a saved register and it should be preserved on call but the "func" procedure does not preserve it and changes its value. The func procedure should first save its value to the stack and at the end of the procedure, it should be restored from the stack. Another error is the typo at the last jr instruction. It should be $ra not $r. Also there cannot be spaces in function names. The fixed code can be written like this:

```
# In: $a0 (an unsigned integer)
# Out: $v0

func:        addi $sp, $sp, -4
             sw $s0, 0($sp)
             add $s0, $zero, $a0
             addi $v0, $zero, 1

func_loop:   beq $s0, $zero, func_return
             add $v0, $v0, $s0
             addi $s0, $s0, -1
             j func_loop

func_return:lw $s0, 0($sp)
             add $sp, $sp, 4
             jr $ra
```