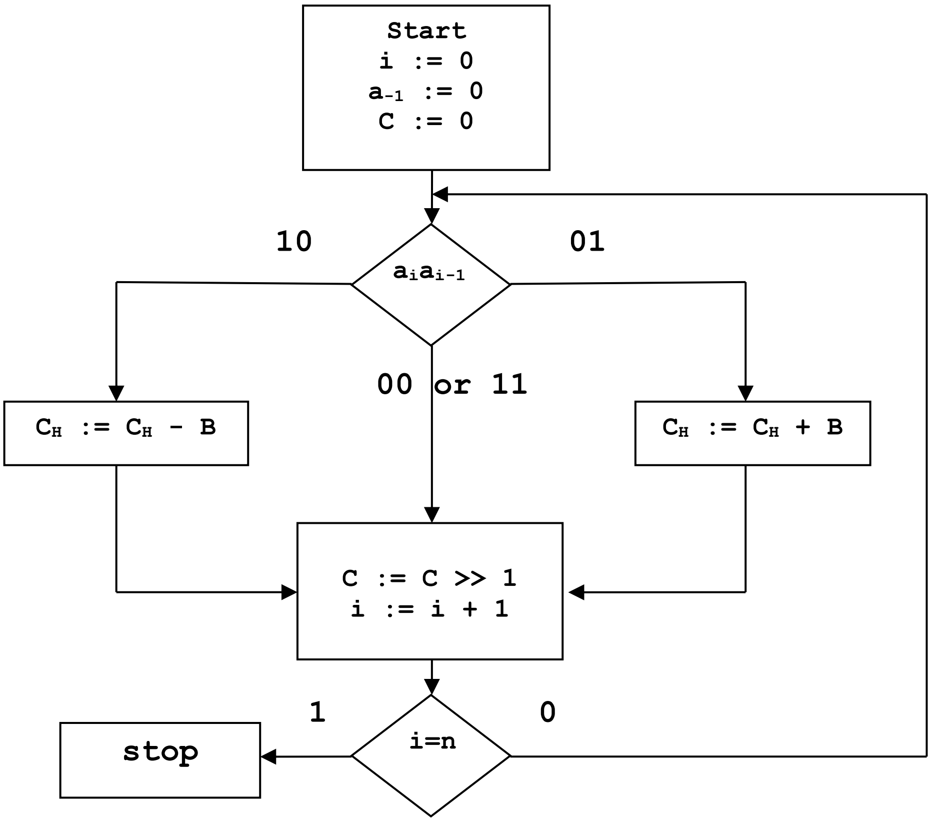
Homework #2 Oytun Kuday Duran 28357

Assigned: 29/03/2023
Due: 09/04/2023

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



C = A × 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 × 5 = ?
-5 × -4 = ?
-4 × 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	-	0000 0000
				C >> 1	0000 0000
1	0101	0110	10	Ch -= $(0101)_2$	1011 0000
				C >> 1	1101 1000
2	0101	0110	11	-	1101 1000
				C >> 1	1110 1100
3	0101	0110	01	Ch += $(0101)_2$	0011 1100
				C >> 1	0001 1110

-5 × -4

i	B	A	a_ia_{i-1}	operation	C
0	1100	1011	10	Ch -= $(1100)_2$	0100 0000
				c>>1	0010 0000
1	1100	1011	11	-	0010 0000
				C >> 1	0001 0000
2	1100	1011	01	Ch += $(1100)_2$	1101 0000
				C >> 1	1110 1000
3	1100	1011	10	Ch -= $(1100)_2$	0010 1000
				C >> 1	0001 0100

-4 × 7

i	B	A	a_ia_{i-1}	operation	C
0	0111	1100	00	-	0000 0000
				C >> 1	0000 0000
1	0111	1100	00	-	0000 0000
				C >> 1	0000 0000
2	0111	1100	10	Ch -= $(0111)_2$	1001 0000
				C >> 1	1100 1000
3	0111	1100	11	-	1100 1000
				C >> 1	1110 0100

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.

Here, we can reuse the t registers but I wanted to be easily understandable, so I used a few more t registers:

- **`f = -g + h + B[i]+ C[j]`**

```
sll $t0, $s3, 2         #t0 has 4*i

sll $t1, $s4, 2         #t1 has 4*j

add $t2, $s6, $t0               #t2 has addr of B[i]

add $t3, $s7, $t1               #t3 has addr of C[j]

lw $t4, 0($t2)                  #t4 has val of B[i]

lw $t5, 0($t3)                  #t5 has val of C[j]

sub $t6, $s3, $s1               #t6 has h-g

add $t7, $t4, $t5               #t7 has B[i]+ C[j]

add $s0, $t7, $t6               # f = -g + h + B[i]+ C[j]
```

- **`f = A[B[g]+ C[h] + j]`**

```
sll $t0, $s1, 2          #4*g

sll $t1, $s2, 2         # 4*h

add $t0, $s6, $t0               #address of B[g]

add $t1, $s7, $t1               #address of C[h]

lw $t2, 0($t0)          # B[g] value is at $t2

lw $t3, 0($t1)          # C[h] value is at $t3

add $t4, $t2, $t3       #t4 has B[g] + C[h]

add $t4, $t4, $s4        #t4 has B[g] + C[h] + j

sll $t4, $t4, 2          # t4 has 4*(B[g] + C[h] + j)

add $t4, $t4, $s5       # t4 has the address &A[0] + 4*(B[g] + C[h] + j)

lw $t5, 0($t4)          # t5 has A[B[g]+ C[h] + j]

add $s0, $t5, $zero     # f = A[B[g]+ C[h] + j]
```

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**.

```
lw $s1, 0($s0)          #s1 reg has A[0] value, local variable t in this case.
Li $s2, 0        #assuming i is local variable as well
li $t0, 5        #we use our first temporary register to store limit value for loop as 5
loop:
  sll $t1, $s2, 2          #multiplying i by 4 to go through arrays indices and storing it in $t1
  add $t2, $s0, $t1     #to get to the ith element, we add base address with 4i and store in $t2
  lw $t3, 4($t2)       # we load A[i+1] to $t3,  to reach i+1 th element, we look for 1*4=4
  sw $t3,  0($t2)       #we store A[i+1] to A[i]
  addi $s2, $s2, 1       # incrementing local var i
  blt $s2, $t0, loop     # if i is less than 5, we branch loop again.
sw $s1, 20($s0)    # Store the first element of A into A[5]  , 20 is 5*4
```

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`?

```
//The type of v0 wasn’t clear so I assumed an integer.
int func(unsigned int n){
unsigned int s0=n;
int v0 = 1;
while (s0 != 0){
        v0 += s0;
        s0--;
}
 return v0;
}
```

Initial value of v0 is 1, in loop, s0 will go through 5 to 0 and exit the loop.
1+5+4+3+2+1=16

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.

$s0 register is being used instead of temp registers, the s register should be saved and loaded the value from stack. Instead of a $s register, we can use a temporary register or use stackpointer with lw and sw instructions, moreover, we should also restore the ra before returning:

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

func:
  addi $sp, $sp, -8
  sw $ra, 4($sp)
  sw $s0, 0($sp)

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

func loop:
  beq $s0, $zero, func return
  add $v0, $v0, $s0      # adding counter to the sum
  addi $s0, $s0, -1     # decrementing the counter by 1
  j func loop           # jump back to the beginning of the loop

func return:
  lw $s0, 0($sp)        # restore s0 from stack
  lw $ra, 4($sp)        # restore the return address from stack
  addi $sp, $sp, 8      # deallocate stack space by popping last 2 elements
  jr $ra            # return from function
```