EGC442 Class Notes 2/24/2023

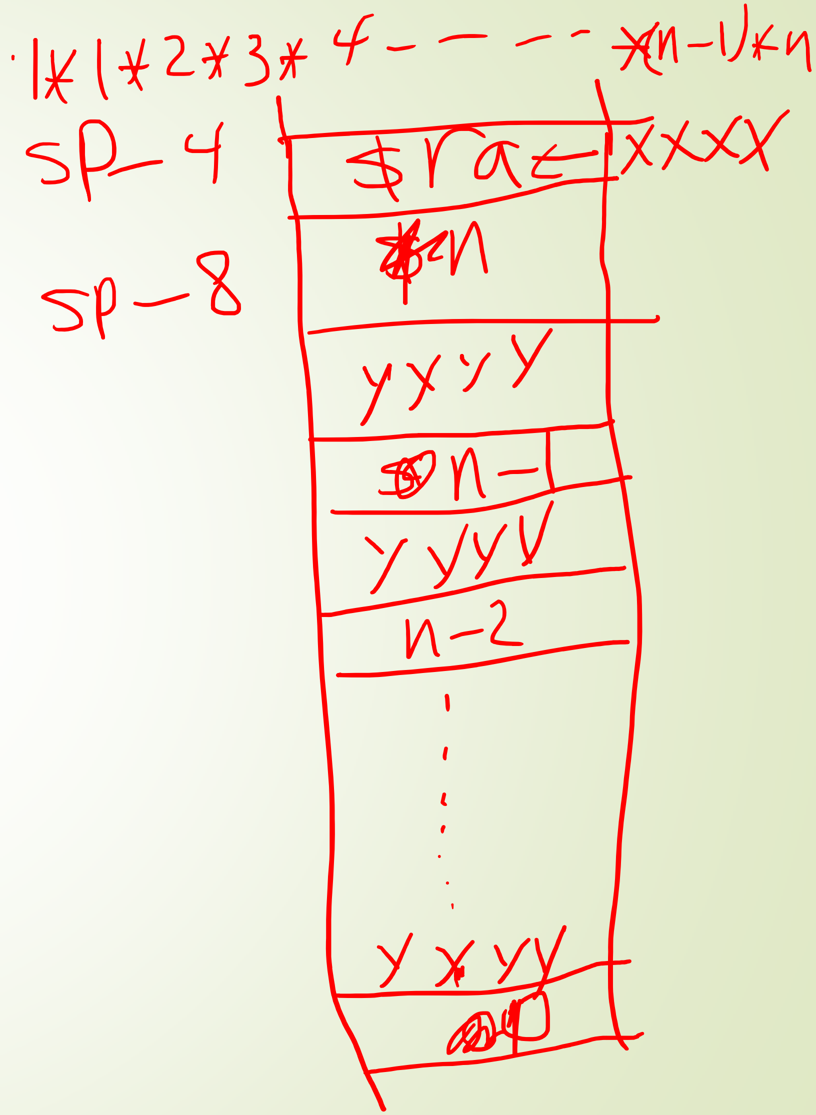
Baback Izadi
Division of Engineering Programs
bai@engr.newpaltz.edu

```
int fact (int n)
{
  if (n < 1) return 1;
  else return n * fact(n - 1);
}
```

Argument n in $a0
Result in $v0

jal fact

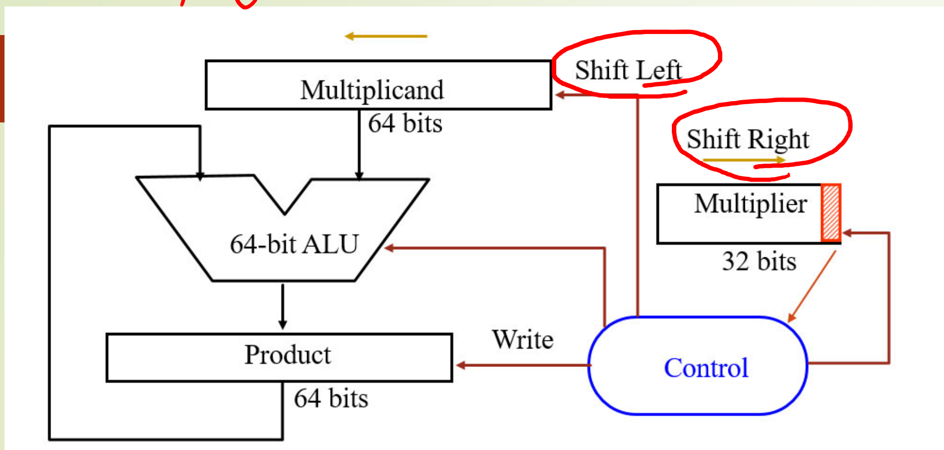
xxxx → $ra



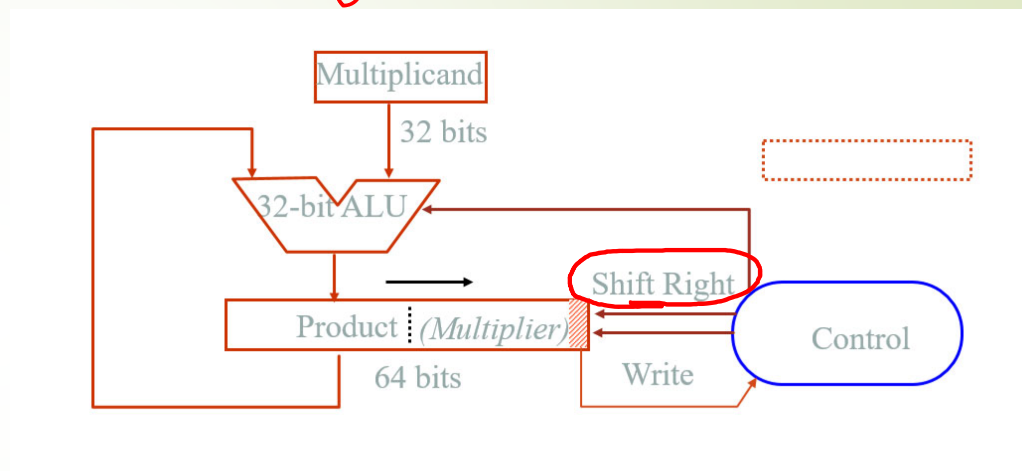
Non-Leaf Procedure Example

```
fact:
      addi    $sp, $sp, -8     # adjust stack for 2 items
      sw      $ra, 4($sp)      # save return address
      sw      $a0, 0($sp)      # save argument        if n=0 → $t0 = 1
      slti    $t0, $a0, 1      # test for n < 1
      beq     $t0, $zero, L1          ; → $t0≠1 → n<1
      addi    $v0, $zero, 1    # if so, result is 1
      addi    $sp, $sp, 8      # pop 2 items from stack
      jr      $ra              # and return
L1:   addi    $a0, $a0, -1     # else decrement n
      jal     fact             # recursive call
yyyy  lw      $a0, 0($sp)      # restore original n
      lw      $ra, 4($sp)      # and return address
      addi    $sp, $sp, 8      # pop 2 items from stack
      mul     $v0, $a0, $v0    # multiply to get result
      jr      $ra              # and return
```

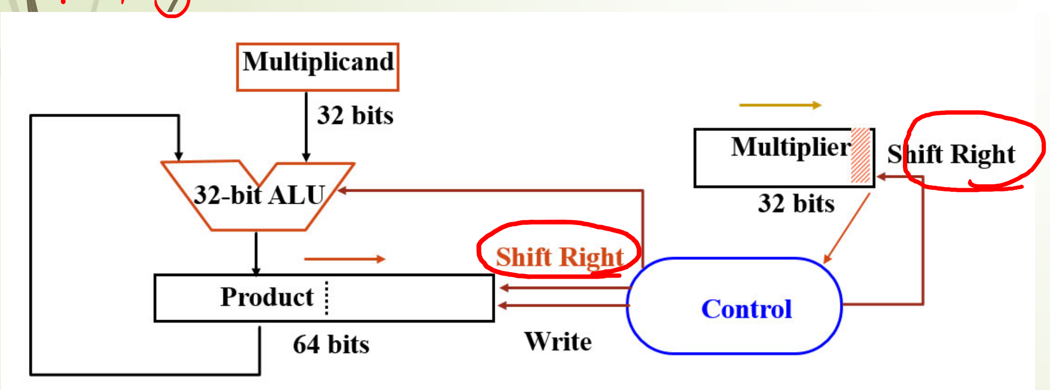
Algorithm 1

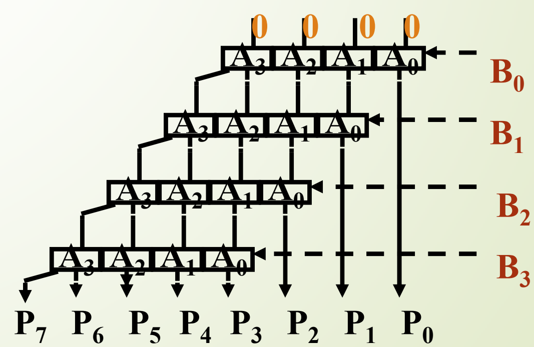


Algorithm 3

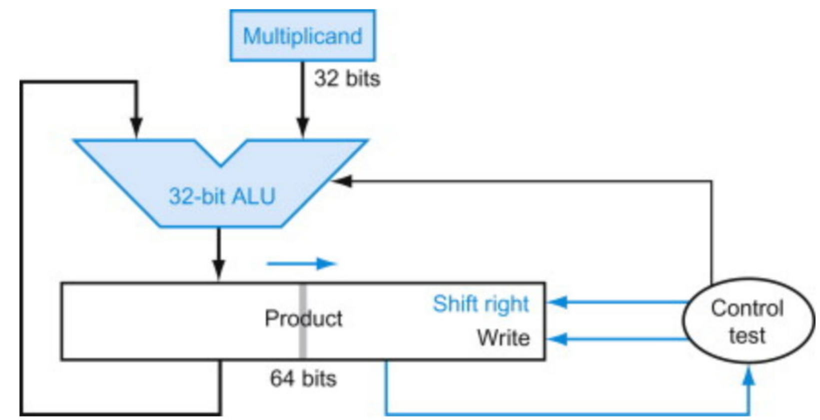


Algorithm 2





1)


Multiplicand
32 bits
32-bit ALU
Product
Shift right
Write
64 bits
Control
test

a) The refined multiplication hardware halves the width of the Multiplicand register from 64-bits to 32-bits.

- (selected) True
- False

Correct

The refined multiplication hardware shifts the Product register right 1 bit in each step instead of shifting the Multiplicand register left 1 bit in each step. Because the Multiplicand is no longer shifted left, the register width can be reduced.

b) The Multiplier register is removed and placed inside of the _____ register.

- (selected) Product
- Multiplicand

Correct

The multiplier is placed in the right half of the Product register. As the Product register is shifted to the right to allocate room for the accumulated sum of intermediate products, the least significant bit of the multiplier is no longer needed and can be shifted out of the register.

c) The ALU adds the 64-bit Product and 32-bit Multiplicand, and then stores the result into the Product register.

- True
- (selected) False

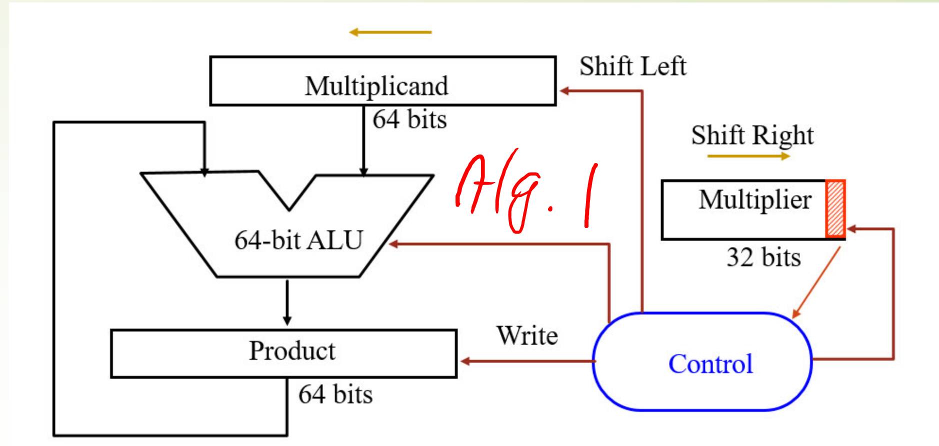
Correct

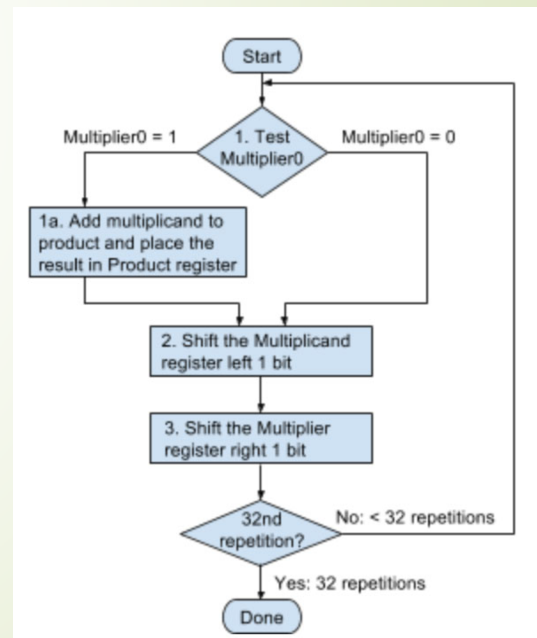
The ALU adds the upper 32-bits of the Product with the 32-bit Multiplicand. The result is then stored in the upper 32-bits of the Product register. The Product register is then shifted right 1 bit before the next step.

2

Iteration	Step	Multiplier	Multiplicand	Product
0	Initial values	1100	0000 0101	0000 0000
1	1: 0 ⇒ No operation	1100	0000 0101	0000 0000
	2: Shift left Multiplicand	1100	0000 1010	0000 0000
	3: Shift right Multiplier	0110	0000 1010	0000 0000
2	1: 0 ⇒ No (a) operation	0110	0000 1010	0000 0000
	2: Shift left Multiplicand	0110	0001 (b) 0100	0000 0000
	3: Shift right Multiplier	0011 (c)	0001 0100	0000 0000
3	1: 1 ⇒ add (d) prod + Mcand	0011	0001 0100	0001 0100
	2: Shift left Multiplicand	0011	0010 1000	0001 0100
	3: Shift right Multiplier	0001	0010 1000	0001 0100
4	1a: 1 ⇒ Prod = Prod + Mcand	0001	0010 1000	0011 (e) 1100
	2: Shift left Multiplicand	0001	0101 0000	
	3: Shift right Multiplier	0000	0101 0000	

a)





Product

1. 0000 1100 $\boxed{0}$ ↳ no add

Shift Right

2. 0000 0110 $\boxed{0}$ ↳ no add

shift Right

0000 0011 $\boxed{1}$ ↳ add

0101

0101 0011

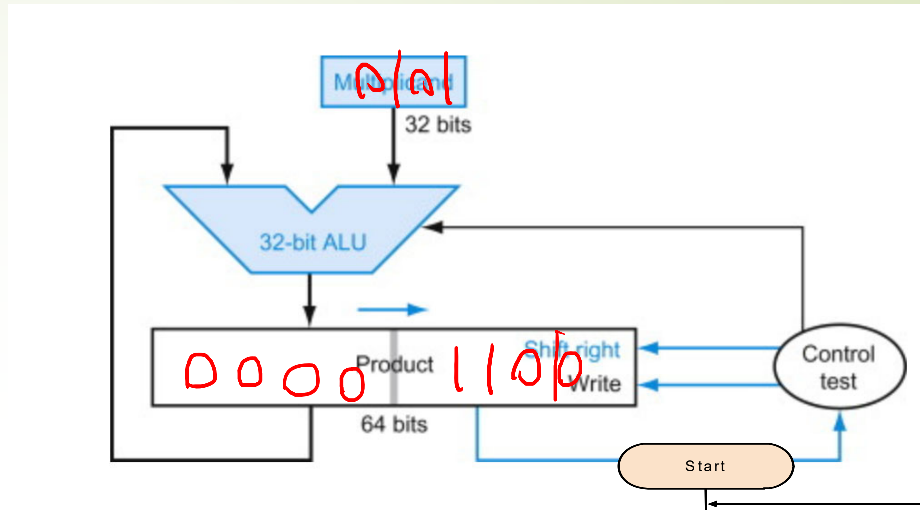
shift right

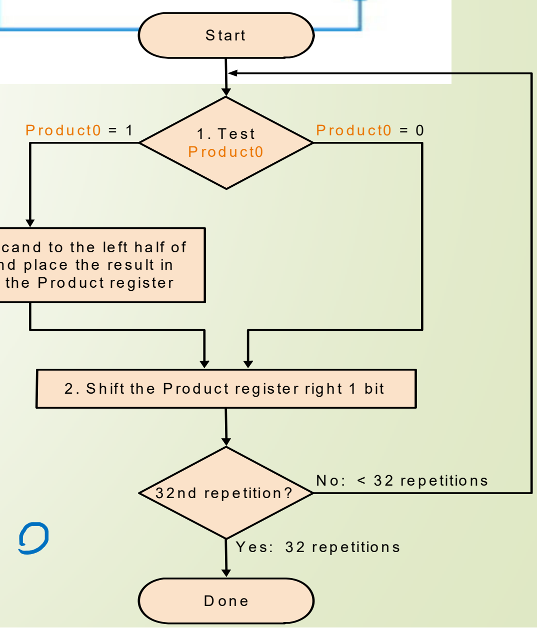
0010 1001 $\boxed{1}$ ↳ add

0101

0111 1001 → SR

0011 1100





1) The multiplication hardware supports signed multiplication.

- (●) True
- ( ) False

Correct

The multiplier and multiplicand are first converted to positive numbers, and then multiplied using the same multiplication hardware. The product is negated if the multiplier and multiplicand signs disagree.

2) The 32-bit registers, called Hi and Lo, combine to form a 64-bit product register.

- (●) True
- ( ) False

Correct

The move from low (`mflo`) and move from high (`mfhi`) instructions can be used to transfer the contents of registers Hi and Lo to a general-purpose register.

3) The multiply (`mult`) instruction ignores overflow, while the multiply unsigned (`multu`) instruction detects overflow.

- ( ) True
- (●) False

Correct

Both instructions ignore overflow, so the software must detect overflow.

Floating Point (a brief look)

- Like scientific notation
  - -2.34×10^{56}
  - $+0.002 \times 10^{-4}$
  - $+987.02 \times 10^{9}$

normalized

not normalized

In binary
$\pm 1.xxxxxxx_2 \times 2^{yyyy}$

- Representation:
  - sign, exponent, significand: $(-1)^{sign} \times \text{significand} \times 2^{exponent}$
  - more bits for significand gives more accuracy
  - more bits for exponent increases range
- IEEE 754 floating point standard:
  - single precision:

sign bit	8 bit exponent	23 bit significand

  - double precision:

sign bit	11 bit exponent	52 bit significand

4a 1) A calculation that leads to a number being too large to represent is called _____.

- (•) overflow
- ( ) underflow
- ( ) a fraction

Correct

Floating-point numbers are represented with a fixed number of bits, thus can only represent a fixed range of numbers. Floating-point arithmetic can lead to numbers that are too large to represent given the number of bits available.

b 2) Increasing the size of the _____ used to represent a floating-point number impacts the number's precision.

- (•) fraction
- ( ) exponent

Correct

Floating-point numbers are represented using a fixed number of bits, so compromise is needed between the size of the fraction and the size of the exponent.

3) A _____ precision floating-point number is represented with two MIPS words.

- ( ) single
- (•) double

Correct

A double precision floating-point number is represented with two MIPS words, or 64-bits. The exponent is increased to 11-bits to enable representation of a larger range of values; the fraction is increased to 52-bits to enable greater precision.

5

$\frac{3}{8}$ or $\frac{3}{2^3}$	**Rewrite as a fraction** The number is rewritten as a fraction whose denominator is a power of 2.	Correct
$\frac{11_{two}}{2^3}$ or 0.011_{two}	**Rewrite as a binary number** 3_{ten} becomes 11_{two}. The fraction contains 2^3 in the denominator, so the binary point is moved left 3 positions and results in 0.011_{two}.	Correct
$1.1_{two} \times 2^{-2}$	**Rewrite as normalized scientific notation** The binary point is moved to the right until a non-zero digit appears to the left of the binary point.	Correct
0	**S = ?** A sign bit of 0 results in a positive number. $(-1)^S = (-1)^0 = 1$	Correct
125	**Exponent = ?** (Exponent - 127) = -2 Exponent = 125	Correct
.1000 0000 0000 0000 0000 0000	**Fraction = ?** The leading 1-bit of a normalized binary numbers is implicit, so only the values to the right of the binary point are needed.	Correct
$(-1)^0 \times (1 + .1000\ 0000\ 0000\ 0000\ 0000\ 0000) \times 2^{(125 - 127)}$	**IEEE 754 binary single precision representation** The sign bit, exponent, and fraction are plugged into the basic single precision floating point equation.	Correct

$\frac{-15}{16}$ or $\frac{-15}{2^4}$	**Rewrite as a fraction** The number is rewritten as a fraction whose denominator is a power of 2.	Correct
$\frac{1111_{two}}{2^4}$ or 0.1111_{two}	**Rewrite as a binary number** 15_{ten} becomes 1111_{two}. The fraction contains 2^4 in the denominator, so the binary point is moved left 4 positions and results in 0.1111_{two}.	Correct
$1.111_{two} \times 2^{-1}$	**Rewrite as normalized scientific notation** The binary point is moved to the right until a non-zero digit appears to the left of the binary point.	Correct
1	**S = ?** A sign bit of 1 results in a negative number. $(-1)^S = (-1)^1 = -1$	Correct
1022	**Exponent = ?** (Exponent - 1023) = -1 Exponent = 1022 The exponent bias for double precision is 1023.	Correct
.1110 0000 ... 0000	**Fraction = ?** The leading 1-bit of a normalized binary numbers is implicit, so only the values to the right of the binary point are needed. The fraction is represented with 52 bits, only some of the bits are shown.	Correct
$(-1)^1 \times (1 + .1110\ 0000 \ldots 0000) \times 2^{(1022 - 1023)}$	**IEEE 754 binary double precision representation** The sign bit, exponent, and fraction are plugged into the basic double precision floating point equation.	Correct

7. Convert the single precision binary floating-point representation to decimal.

128 64 32 16 8 4 2 1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 bit 8 bit 23 bit

$+$

130

$-\ 127$

$= 4$

$\Rightarrow +1.1010 \times 2^{4}$

16 8 4 2 1

1 1 0 1 0

26 ✓