2020-21

10. (a) Briefly explain the eight great ideas in Computer Architecture. (12)

1. Design for Moore's law:
Moore's law is the observation that the number of transistors in a dense IC doubles about every two years.

2. Use abstraction to simplify design:
Computing systems maintains a hierarchical structure.
Lower level details are hidden to the higher level.
Higher level only gets the abstract view.
Both hardware and software consist of hierarchical layers using abstraction.

3. Make the common case fast:
More efficiency in common case, more impact in overall design.

4. Performance via parallelism:
Current multiprocessor system exploits parallelism.
Often needs special care for coordination.

5. Performance via pipelining:
A special case of parallelism.
Performing multiple non-dependent operations at the same time.

6 Performance via prediction:
Perform operation just based on prediction/assumption.
Applicable when impact is not costly.

7. Hierarchy of memories:
We want faster and cheaper memory.
Faster memory is costlier.
Cheaper memory is slower.
Trade-off is memory hierarchy.

8. Dependability vs redundancy:
Redundancy means keeping multiple copies.
One fails, another exists → dependable.

(b) Suppose you want to design a 4-bit ALU where the inputs to the ALU are A $(A_3A_2A_1A_0)$ and B $(B_3B_2B_1B_0)$, and the selection variables are S_3, S_2, S_1 and S_0. Also, consider X_i, Y_i and Z_i as the inputs to the i-th parallel adder of the ALU. The ALU must satisfy the following functional design specification (given in Table 10b) that has to be realized through the parallel adders. Your task is to derive the input equations (X_i, Y_i and Z_i) for the parallel adders of the ALU. You must design using the least possible number of gates. **(18)**

S_3	S_2	S_1	S_0	Required Functions	X_i	Y_i	C_{in}
0	0	0	0	$F = A$	A_i	1	1
0	0	0	1	$F = A - 1$	A_i	1	0
0	0	1	0	$F = A + B + 1$	A_i	B_i	1
0	0	1	1	$F = A + B$	A_i	B_i	0
0	1	0	0	$F = A - B$	A_i	B_i'	1
0	1	0	1	$F = A - B - 1$	A_i	B_i'	0
0	1	1	0	$F = A + 1$	A_i	0	1
0	1	1	1	$F = A$	A_i	0	0
1	0	0	X	$F = A'$	A_i	1	X
1	0	1	X	$F = AB'$	$A_i + B_i$	B_i	X
1	1	0	X	$F = A \odot B$ [XNOR]	A_i	B_i'	X
1	1	1	X	$F = A \vee B$ [OR]	$A_i + B_i$	0	X

Table 10(b): Required functional design specification

$C_0 = S_0'$
$Z_i = S_3' C_{i-1}$
$X_i = A_i + S_3 S_1 B_i$
$Y_i = B_i \overline{S_2} + B_i' \overline{S_1}$

(c) Briefly describe how to perform a signed multiplication operation.

Convert the multiplier and multiplicand to positive numbers and remember the original signs.
Run the algorithms for 31 iterations (excluding sign bit)
Negate the final product if the original signs disagree.

(b) Consider the following figure (Figure 11b) of a processor unit employing a scratchpad memory. How many micro operations does it require to complete an addition operation with two operands, and store the result? Which changes and constraints are required to reduce the number of micro operations to only one? **(2+6=8)**

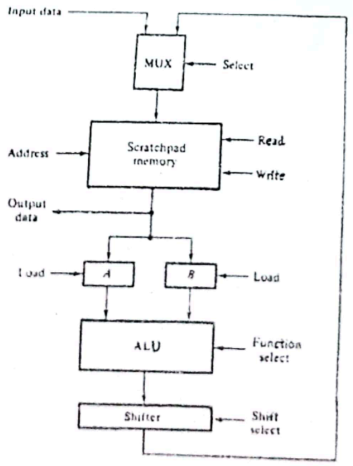


Figure 11(b) Processor unit employing a scratchpad memory

$R_1 \leftarrow R_2 + R_3$

T1 : A ← M[010] read R2 into reg A
T2 : B ← M[011] read R3 into reg B
T3 : M[001] ← A+B perform operation in ALU and transfer the result to R1

Thus, it requires 3 micro-operations.

To reduce the number of micro-operations to only one →
Dual ported scratchpad memory to allow two memory locations to be read simultaneously.
ALU with direct memory access to fetch operands directly from scratchpad memory.
Enhance the memory and control unit to support simultaneous read and write operations.

(c) Write the equivalent efficiency sequence of MIPS instructions for the following C code fragment. **(12)**

```
int testFunc (int a, int b, int c)
{
  int i=0, result=0;
    do
    {
       if(i>b)
          result += c;
       i++;
    }while (i<a)

  return result;
}
```

Assume, $s1 and $s2 will carry the values of variable *i* and variable *result*, respectively.

```
testFunc:
    addi $sp, $sp, -8
    sw $s2, 4($sp)
    sw $s1, 0($sp)
    add $s2, $zero, $zero
    add $s1, $zero, $zero
L1:
    slt $t0, $a1, $s1
    beq $t0, $zero, L2
    add $s2, $s2, $a2
L2:
    addi $s1, $s1, 1
    slt $t1, $s1, $a0
    bne $t1, $zero, L1
    add $v0, $s2, $zero
    lw $s2, 4($sp)
    sw $s1, 0($sp)
    addi $sp, $sp, 8
    jr $ra
```

(d) Provide an illustrative example for showing how MIPS architecture follows the design principle: "Good design demands good compromises." (5)

There are three instruction format instead of just one for all kind of operations.
R-type (arithmetic operations with registers)
I-type (immediate and data transfer operations)
J-type (to support long jump to remote procedure address)

12. (a) The data for the two programs A and B are given below. (8)

Instruction Type	Clock Cycle per Instruction (CPI)	Instruction Count in Program A	Instruction Count in Program B
Multiplication	3	2	3
Addition	1	1	2
Subtraction	2	4	3
Division	4	2	2

Which program has the higher average clock cycle per instruction?

Program A: $CPI_{avg} = \frac{6+1+8+8}{2+1+4+2} = \frac{23}{9} = 2.556$

Program B: $CPI_{avg} = \frac{9+2+6+8}{3+2+3+2} = 2.5$

Program A has highere avg CPI

(b) Consider the following MIPS code: **(10)**

L1: bne $t0, $t1, L3

L2: j LX

L3: addi $t0, $t1, 1

Now, if the opcode of *j*, *bne* and *addi* are 2, 5 and 8 respectively, the register *$t0* and *$t1* are indicated by register values 8 and 9 respectively, and the addresses are

L1 → 1000

L2 → 1004

L3 →1008

LX → 131072

then, write down the instruction format name and the instruction values for each of the above-mentioned instructions. The following example is provided as a hint.

Example: Consider the following MIPS instruction:

add $s1, $s2, $s3

The instruction format is R-format and instruction values should be:

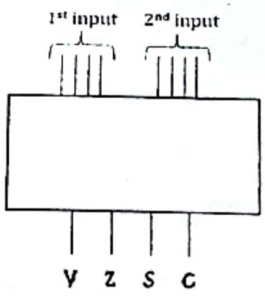
Address	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
0400	0	18	19	17	0	0

Address	6bits	5bits	5bits	5bits	5bits	6bits	
1000	5	8	9	1			I-type
1004	2	32768					J-type
1008	8	9	8	1			I-type

(c) Briefly explain the importance of guard digit, round digit, and sticky bit during rounding an intermediate floating point number. **(9)**

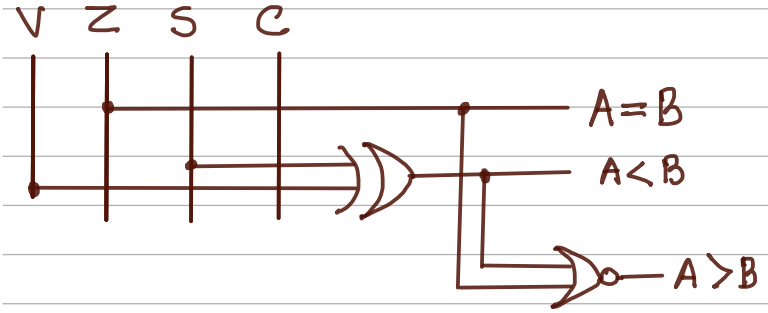
They improve accuracy of computation.
Guard bit: (first extra bit) → rounding addition results.
Round bit: (second extra bit) → rounding multiplication results.
Stricky bit: (thind extra bit) → resolving ties like 0.50---00 vs 0.500---01

(d) Consider the following module. (8)



This module takes two 4-bit inputs, and shows only the four status bits (V: Overflow, Z: Zero, S: Sign and C: Carry) considering the subtraction operation of the second input from the first input. If the first 4-bit input is expressed by A and second 4-bit input is expressed by B, draw the diagram of the complete circuit that can determine whether A>B, A=B, or A<B.

A>B	Z=0 and (S=0, V=0 or S=1, V=1)	$\overline{Z}(S \odot V)$
A=B	Z=1	Z
A<B	S=1, V=0 or S=0, V=1	$S \oplus V$



$\overline{Z}(S \odot V) = \overline{Z \cdot \overline{S \oplus V}}$
$= \overline{Z + S \oplus V}$

13. (a) Consider a 32-bit system where 24 bits are reserved for fraction, one bit for sign, and the rest for exponent. Answer the following questions. (4×4=16)

(i) Why is bias used during the floating point representation?

(ii) What should be the value of the bias for this system? Why?

(iii) What are the smallest and the largest absolute values that can be stored in this system?

(iv) What is the smallest denormalized number that can be stored using this system?

(i) Exponent = actual exponent + bias
Ensures exponent is unsigned

(ii) Bias $= 2^{7-1} - 1 = 63$

(iii) Smallest value $= 1 \times 2^{-62} = 2.1684 \times 10^{-19}$
Largest value $= 2 \times 2^{63} = 1.84467 \times 10^{19}$

(iv) Smallest denormalized number $= 0.\underbrace{000\ldots1}_{24} \times 2^{-63} = 2^{-87}$

(b) How is address calculation performed in PC-relative addressing? (7)

PC is 32-bit but the address in conditional branching is 16-bit.
Forces the program instruction count within 2^{16} instructions.
Most of the conditional branches jump nearby (forward or backward).
We use PC relative addressing to access → (PC + relative constant address)
Can support branching up to $\pm 2^{15}$ relative constant address value (1 signed bit)
PC normally increements 1 word after every instruction.
The addressing should be (PC+4 + relative constant address)

(c) Write the equivalent efficient sequence of MIPS instructions for the following C code fragment. Assume appropriate registers for each of the variables. (7)

```
float a, b, c;
double w, x, y, z;
if (a+b>c)
    x=(y*z)/w;
```

```
lwc1   $f0, a($sp)
lwc1   $f2, b($sp)
add.s  $f0, $f0, $f2
lwc1   $f2, c($sp)
c.gt.s $f0, $f2
bc1f   L1
lwc1   $f0, y($sp)
lwc1   $f1, (y+4)($sp)
lwc1   $f2, z($sp)
lwc1   $f3, (z+4)($sp)
mul.d  $f0, $f0, $f2
lwc1   $f2, w($sp)
lwc1   $f3, (w+4)($sp)
div.d  $f0, $f0, $f2
swc1   $f0, x($sp)
swc1   $f1, (x+4)($sp)
L1:
```

(d) What are the main differences between x86 architecture and MIPS architecture? (5

x64 : CISC

MIPS: RISC

2019-20

1. (a) Suppose we want to compare the performance of *Computer System A* with *Computer System B*. For this purpose, *Program 1* is run in both the computer systems and corresponding information (logs) are captured. The logs are presented below in Table 1.1 and Table 1.2. **(15)**

Operation	Required Clock Cycle(s)
Multiplication 1 of Program 1	4
Addition 1 of Program 2	2
I/O overhead time required by OS	5
Multiplication 2 of Program 1	4
Division 1 of Program 1	5
Multiplication 1 of Program 2	4
I/O overhead time required by OS	2
Division 2 of Program 1	5
I/O overhead time required by OS	4
Addition 1 of Program 3	2
Addition 1 of Program 1	2

Table 1.1: Logs from Computer System A

Operation	Required Clock Cycle(s)
Multiplication 1 of Program 4	5
Multiplication 1 of Program 1	5
I/O overhead time required by OS	1
Multiplication 2 of Program 1	5
Division 1 of Program 1	6
Multiplication 2 of Program 4	5
Addition 1 of Program 4	3
I/O overhead time required by OS	3
Addition 2 of Program 4	3
Division 2 of Program 1	6
Addition 1 of Program 1	3

Table 1.2: Logs from Computer System B

If the clock rates of *Computer System A* and *Computer System B* are 2 GHz and 2.5 GHz respectively, then compare their performances based on the execution time of *Program 1*.

CPU Time (A) = $20 \times 0.5 \times 10^{-9} = 10$ ns
CPU Time (B) = $25 \times 0.4 \times 10^{-9} = 10$ ns
Same performance

(b) What is the importance of Moore's Law? "To increase the overall performance of a computer by x amount, it is required to identify the most important aspect of the computer and improve its performance by x amount." Do you agree or disagree with the statement? Justify your answer with proper mathematical explanation. **(4+8=12)**

Moore's law is the observation that the number of transistors in a dense integrated circuit doubles every two years.
Importance: technological progress bla bla bla

According to Amdahl's law,
Execution time after improvement = Execution time affected by the improvement/amount of improvement + Execution time unaffected.
hence, the statement is incorrect.

(c) What are the advantages of using Dynamically Linked Libraries (DLL) during a C program execution? What is the role of JIT in Java program execution? **(4+4=8)**

DLL are library routines that are linked to a program during execution to incorporate the updated version of library routines.

In lazy procedure linkage, each routine is linked only when it is called.

Java programs are first compiled into byte code. JVM (interpreter) can execute Java byte code. But this process is slow.
JIT (Just In Time) Compiler makes it faster. It statistically identifies the commonly used (hot) methods and compiles these methods into native instruction set.

2. (a) Suppose you want to design a 4-bit ALU where the inputs to the ALU are A ($A_3A_2A_1A_0$) and B ($B_3B_2B_1B_0$), and the selection variables are S_3, S_2, S_1 *and* S_0. Also, consider X_i, Y_i *and* Z_i as the inputs to the i^{th} parallel adder of the ALU. The ALU must satisfy the following functional design specification (given in Table 2a) that has to be realized through the parallel adders. Your task is to derive the input equations (X_i, Y_i *and* Z_i) for the parallel adders of the ALU. You must design using the least possible number of gates. **(20)**

S_3	S_2	S_1	S_0	**Required Functions**	X_i	Y_i	Z_i
0	0	0	0	$F = A'$	A_i'	0	0
0	0	0	1	$F = A' + 1$	A_i'	0	1
0	0	1	0	$F = A' + B$	A_i'	B_i	0
0	0	1	1	$F = A' + B + 1$	A_i'	B_i	1
0	1	0	0	$F = A' - B - 1$	A_i'	B_i'	0
0	1	0	1	$F = A' - B$	A_i'	B_i'	1
0	1	1	0	$F = A' - 1$	A_i'	1	0
0	1	1	1	$F = A'$	A_i'	1	1
1	0	0	x	$F = (AB)'$ [NAND]	$A_i' + B_i'$	0	x
1	0	1	x	$F = A \odot B$ [XNOR]	A_i'	B_i	x
1	1	0	x	$F = A \oplus B$ [XOR]	A_i'	B_i'	x
1	1	1	x	$F = A \vee B$ [OR]	$A_i' B_i'$	1	x

Table 2a: Required Functional design specification

$C_0 = S_0$
$Z_i = \overline{S_3} C_{i-1}$
$X_i = S_3 S_2 S_1 \overline{A_i}\,\overline{B_i} + S_3 \overline{S_2}\,\overline{S_1}\,\overline{B_i} + \overline{S_3 S_2 S_1} \cdot \overline{A_i}$
$Y_i = \overline{S_2} S_1 B_i + S_2 \overline{S_1}\,\overline{B_i} + S_2 S_1 = S_2 \overline{B_i} + S_1 B_i$

(b) Suppose, you want to implement a hardware unit that would produce the value of *F* using the following logic (the logic is given in C syntax). **(15)**

```
if ((A>=B && C!=D) || X==0)
      F=1;
else
      F=0
```

Here, each of the variables *A, B, C, D* and *X* is of 4-bits where the variable *F* is of 1-bit. After searching the hardware stationaries, you bought some basic gates. Also, you bought some status-generator modules (see Figure 2b) each of which takes two 4-bit inputs, and shows only the four status bits (V: Overflow, Z: Zero, S: Sign and C: Carry) considering the subtraction operation of the 2nd input from the 1st input.

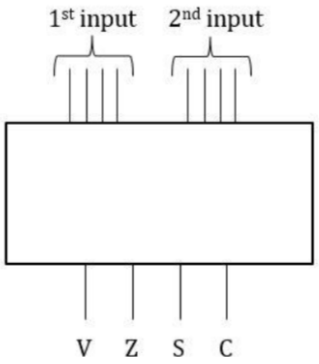
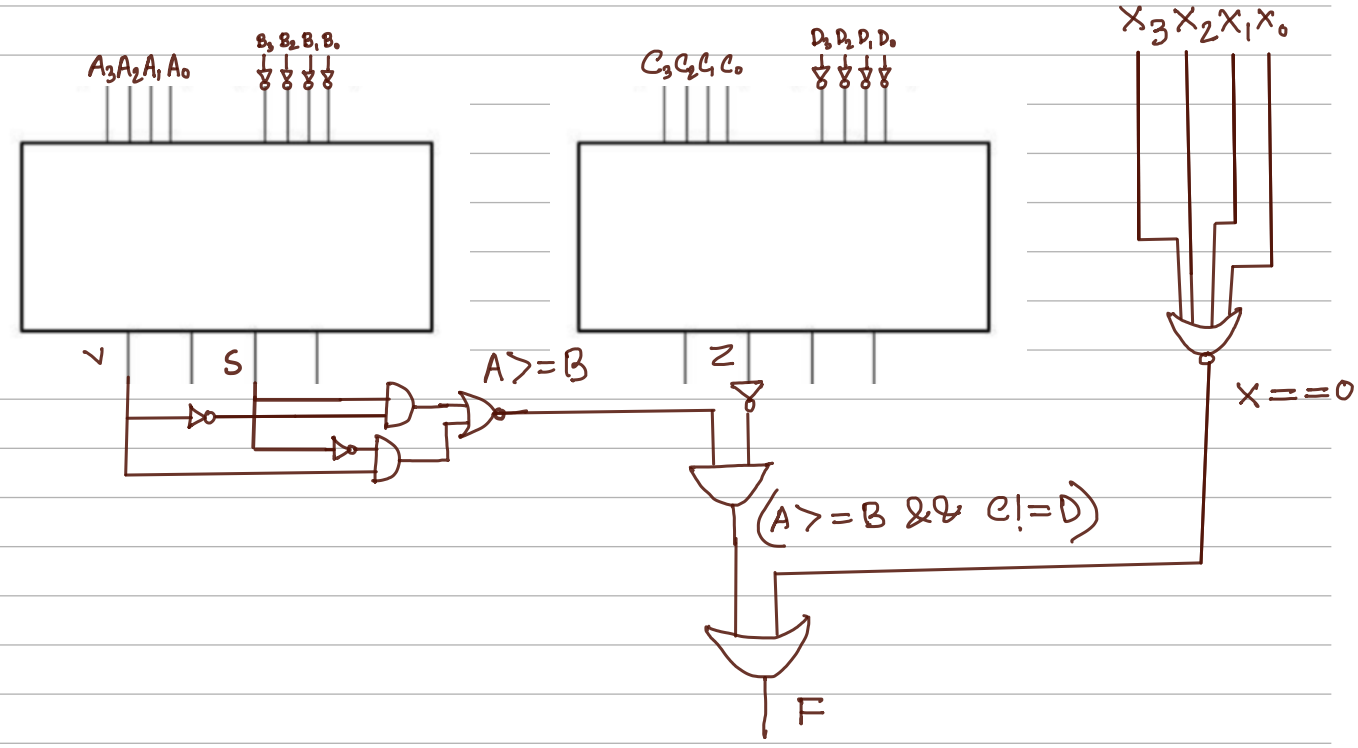


Figure 2b: Status-generator module

Now, (just) draw the circuit diagram of the full hardware unit using basic gates and status-generator modules.

A>=B SOV op: A-B
C!=D Z=0 op: C-D
X==0 Z=1 $X_i=0$



3. (a) Write down the equivalent MIPS code for each of the following the C code fragments. **(6+5+4 =15)**

i. Code Fragment 1:

```
sum = 0;
x = 100;
while (x>=-5)
{
      sum = sum + x;
      x = x-1;
}
```

ii. Code Fragment 2:

```
if(x>y+131075)
        x=1;
```

iii. Code Fragment 3:

```
x = A[131075]+5;
```

i.

```
        add  $s0, $zero, $zero      [sum in $s0]
        addi $s1, $zero, 100        [x in $s1]
L1:     slti $t0, $s1, -5
        bne  $t0, $zero, L2
        add  $s0, $s0, $s1
        addi $s1, $s1, -1
        j L1
L2:
```

ii.

```
        add $t0, $zero, $zero
        lui $t0, 2
        ori $t0, $t0, 3
        add $t0, $s1, $t0           [y in $s1]
        slt $t1, $t0, $s0           [x in $s0]
        beq $t1, $zero, L1
        addi $s0, $zero, 1
L1:
```

iii

```
        add  $t0, $zero, $zero
        lui  $t0, 2
        ori  $t0, $t0, 3            [A address in $s0]
        sll  $t0, $t0, 2
        add  $t0, $s0, $t0
        lw   $s1, 0($t0)
        addi $s1, $s1, 5            [x in $s1]
```

(b) With an appropriate example, briefly describe how pseudoinstructions work. Suppose, there is a critical section in your code where performance matters. Do you think it is always a good idea to write the assembly code by yourself for such type of scenario? Justify your answer. **(6+2+4 =12)**

```
mov  $t0, $t1   ⟶   add $t0, $t1, $zero
blt  $t0, $t1, L ⟶  slt  $t2, $t0, $t1
                    bne  $t2, $zero, L
```

It is never good to write assembly codes for such scenarios became it is easy to make errors (low level language).

(c) Why does MIPS multiplication operation always strictly demand for three operands? Why has the number of registers in MIPS architecture been restricted to 32? **(4+4=8)**

mul rd, rs, rt
least significant 32-bits of product → rd

Size of register: 32-bit because smaller is faster.

4. (a) Suppose, you are working in a system where 16 bits are used to represent a floating-point value. The largest absolute floating-point value that can be represented by the system is $1.11 \ldots 1_{two} \times 2^{63}$ while the smallest absolute floating point value that can be represented by the system is $1.00 \ldots 0_{two} \times 2^{-62}$. **(6+5+4 =15)**

i. Show the floating-point representation format (bit distribution among sign, exponent and fraction portion) of this system.

ii. Show the floating-point representation for the number $-110.11_{two} \times 2^{1}$ in this system assuming an appropriate bias.

iii. What is the smallest absolute denormalized floating-point value that can be stored in this system?

(i)

Sign	exponent	fraction
1-bit	7-bit	8-bit

Bias = 63
n = 7

(ii) -1.1011×2^3
fraction = 1011 0000
exponent = 100 0010
sign = 1
1 100 0010 1011 0000

(iii) $0.0000001 \times 2^{-63} = 2^{-70}$

(b) See the diagram (Figure 4b) of the following multiplication hardware. **(4+4+4 =12)**

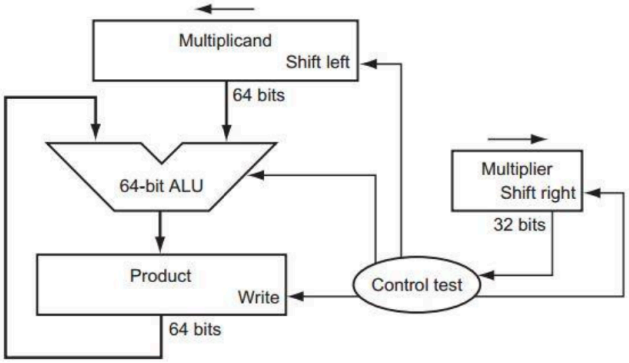
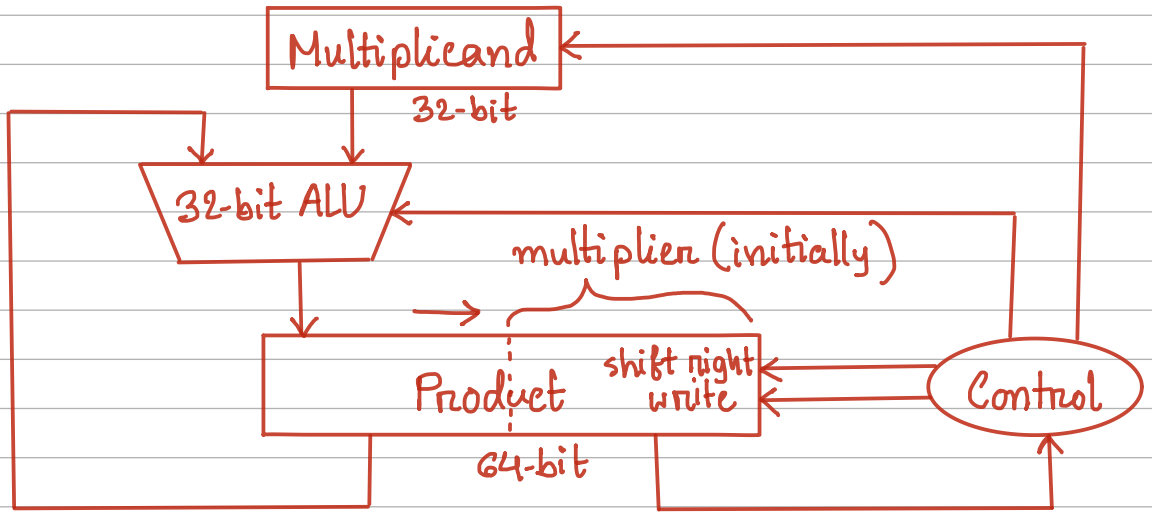


Figure 4b: Multiplication Hardware

Briefly describe the reasons why this hardware is inefficient for a multiplication operation. What changes are required to make this hardware efficient? Draw the diagram of the efficient hardware.

Inefficient because it uses 64-bit ALU and eventually as the multiplier keeps to shift right, the multiplier register becomes empty. Same goes for the 64-bit register of multiplicand, whose right half becomes eventually all 0.
Hence, it is time and space ineffieient.
Optimized multiplier:



here, add/shift are performed together.
One cycle per partial product addition. That is okay if freq. of multiplications is low.

(c) Consider the following C code fragment. **(8)**

```
if(a>b)
   c = c-(c*e)/d;
else
   c = c+d;
```

If a, b, c, d and e are variables of type *double* and their values are already loaded in registers \$f2, \$f4, \$6, \$f8 and \$f10 respectively, then write down the equivalent MIPS assembly code for it. Use the least number of MIPS instructions possible.

```
    c.gt.d $f2, $f4
    bc1f L1
    mul.d $f0, $f6, $f10
    div.d $f0, $f0, $f8
    sub.d $f6, $f6, $f0
    j L2
L1: add.d $f6, $f6, $f8
L2:
```