Deca Lab2 Challenge Task

For the challenge we have used a circuit that multiplies two 8-bit inputs (A and B) into a 16-bit product without using the traditional shift-and-add approach. Instead, it uses a chain of 8-bit adders (each extended to 9 bits with carry), and multiplexers to form and combine partial products.

1. **ADD889 Block (8-bit Adder + Carry)**

* Each ADD889 takes two 8‑bit inputs (P and Q) and produces a 9‑bit result (8 bits of sum + 1 carry bit).
* In the top‐level schematic, five of these adders are cascaded to accumulate partial products.

A diagram of a computer

AI-generated content may be incorrect.

A diagram of a circuit

AI-generated content may be incorrect.

**This ADDER889 sheet simplifies the design so that you don’t have to add a constant 0 connected to a MUX in each addition on the multiplication sheet.**

**A diagram of a computer

AI-generated content may be incorrect.**

1. **MUXs**

* The multiplexers decide whether to feed in A, B, or zero/other control signals at each stage.
* By selecting which inputs go into each adder, the design effectively includes or excludes partial products without explicit shifting.

1. **Control Signals (C1, C2, SEL, etc.)**

* These signals configure the MUXs and control whether each stage adds the relevant partial product or bypasses it.
* They also handle how the 9‑bit outputs (sum + carry) get passed along from one stage to the next.

1. **Final Output**

* After the chain of adders, the final 9‑bit result of the last stage is combined (or split) into the lower and upper product bits (LSB and MSB).
* The result is ultimately presented as a 16‑bit product output.

Step Simulation checking if the multiplier is working correctly:

hex 
Inputs 
B (8 bits) 
A (8 bits) 
dec 
bin 
Outputs & Viewers 
OUT (16 bits) 
255 
255 
65025 

hex 
Inputs 
B (8 bits) 
A (8 bits) 
dec 
bin 
127 
Outputs & Viewers 
OUT (16 bits) 

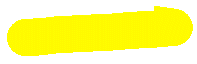
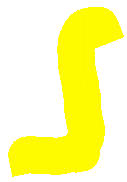
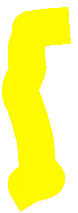
hex 
Inputs 
B (8 bits) 
A (8 bits) 
dec 
bin 
Outputs & Viewers 
OUT (16 bits) 
54 
143 
7722 

hex 
Inputs 
B (8 bits) 
A (8 bits) 
dec 
bin 
Outputs & Viewers 
OUT (16 bits) 
215 
215 

Now we add the MUL Block to the ALU sheet:

A diagram of a computer

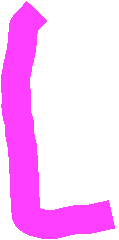
AI-generated content may be incorrect.



When the ALU is instructed to perform a multiply, both operands are routed to the MUL block. The resulting 16‑bit product then flows through MUX4 to become the ALU’s final output, with any relevant flags updated accordingly. MOVC is INS (4:2), MOVC = 001, the bus compare output will be 1 and hence sending the output of the multiplier to input 0 in MUX2.

Slight change to the Data Path:

INS 
DPEN 
FLAGCIN 
PCIN 
(15:0) 
DECODE 
DPDECODE 
WENI 
REGFILE 
ADI SELC 
SEL 
ALUOPC 
SHIFTOPC 
WENI 
XI 
ADDI 
Adder 
(15:0) 
SHIFTOPC 
scNT 
SCNT 
ALU 
MEMSTR 
MEMSTR 
IMMS5 
IMMss 
IMMS8 
MEMLDR 
OP2SEL 
OP2SEL 
ALU 
OP2SEL 
REG16X8 
MUX3 
IMMS5 
Q 
ALUOPC 
FLAGCIN 
MEMSTR 
EXrEND 
EXTEND 
O SEL 
MUX2 
IMMEXT 
NZGEN 
MEMWEN 
MEMDIN 
(15:0) 
MEMDOUT 
(15:0) 
DATA 
NZGEN 
IMM16 
IMMEXT 
(15:0) 
FLAGC 
FLAGV 
RAOUT 
(15:0) 
FLAGN 
FLAGZ 
MUX4 
IMM16 
MEMADDR 
(15:0) 
O SE 
OP2SEL 



This way, MOVC1 Ra, Rb means Ra := Ra x Rb

Testing the multiplier:

We upload mul3.txt onto the eep1:

MOV R0, #0x0016   // initialize op1 = 22

MOV R1, #0x0080   // initialize op2 = 128

MOVC1 R1, R0

Waveform simulation:

A screenshot of a computer

AI-generated content may be incorrect.

On clock cycle 2: instead of 0x0080, 0xFF80 is stored in R1 due to sign extension. However, this does not change the final result, 0x0B00 which is correct.

Moving onto testing 16-bit multiplication:

Mul4.txt:

// R1 and R3 are the 16 bits input

// R4 and R5 are the 32 bits output

// (31:16) (15:0) corresponding to R4 and R5 respectively

EXT 0x35

MOV R1, #123 // R1:= 0x357B

EXT 0x56

MOV R3, #109 // R3:= 0x566D

LSR R0, R1, #8 // R0 = 0x35, R1 = 0x7B

LSR R2, R3, #8 // R2 = 0x56, R3 = 0x6D

// \_\_\_R0(b)\_\_\_ \_\_\_R1(a)\_\_\_

// \_\_\_R2(d)\_\_\_ \_\_\_R3(c)\_\_\_

// note: only the last 8 bits are taken for operations

MOV R6, R1

MOVC1 R6, R3            // a x c

MOV R5, R1

MOVC1 R5, R2            // a x d

MOV R7, R0

MOVC1 R7, R3            // b x c

ADD R5, R7              // ad + bc

LSR R4, R5, #8          // add overflow of 2^8 (ad + bc)

LSL R5, R5, #8          // 2^8 (ad + bc)

ADD R5, R6              // ac + 2^8 (ad + bc)

ADC R4, #0              // add overflow of ac + 2^8 (ad + bc)

MOVC1 R2, R0            // b x d

ADD R4, R2              // bit (31:16) of the result

Therefore,

A math equations on a grid

AI-generated content may be incorrect.

Waveform Simulation:

A screenshot of a computer

AI-generated content may be incorrect.