

VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING



COMPUTER ARCHITECTURE

Report Assignment: semester 23

Problem 3

Multiplication of 2 registers

Advisor(s): Tran Thanh Binh

Student(s): Ngo Thai Minh Tien 2252809

Huynh Minh Khoa 2252346

HO CHI MINH CITY, DECEMBER 2023



Contents

1	Giới thiệu đề tài	5
2	Giải thuật	5
2.1	Sơ lược cách giải quyết bài toán	5
2.2	Chi tiết giải thuật	5
2.2.1	Lấy thông tin từ địa chỉ	5
2.2.2	Thực hiện phép nhân	5
2.2.3	Thực hiện xuất số lớn hơn 32-bit ra màn hình theo HEX	6
2.2.4	Xuất số 128-bit ra màn hình theo decimal	7
2.3	Các hàm sử dụng trong chương trình	9
2.3.1	NhanAB	9
2.3.2	bin_to_hex_string	10
2.3.3	bin128_to_decimal	12
3	Kết quả testcases, tổng số lệnh và thời gian thực thi	14
3.1	Thời gian thực thi của chương trình	14
3.2	Kết quả các testcases	14
4	Hình ảnh INPUT/OUTPUT	18

List of Figures

2.1	Mô phỏng lấy bit bên trong register	6
2.2	Đoạn chương trình nhân AB	9
2.3	Đoạn chương trình chuyển đổi binary thành hexadecimal	11
2.4	Đoạn chương trình chuyển đổi binary thành decimal	13
4.1	Testcase 1	18
4.2	Testcase 2	18
4.3	Testcase 3	18
4.4	Testcase 4	19
4.5	Testcase 5	19
4.6	Testcase 6	19
4.7	Testcase 7	19
4.8	Testcase 8	19
4.9	Testcase 9	19



4.10 Testcase 10	20
4.11 Testcase 11	20
4.12 Testcase 12	20
4.13 Testcase 12	20
4.14 Testcase 13	20
4.15 Testcase 14	20
4.16 Testcase 15	21
4.17 Testcase 16	21
4.18 Testcase 17	21
4.19 Testcase 18	21
4.20 Testcase 19	21
4.21 Testcase 20	21
4.22 Testcase 21	22
4.23 Testcase 22	22
4.24 Testcase 23	22
4.25 Testcase 24	22
4.26 Testcase 25	22
4.27 Testcase 26	22
4.28 Testcase 27	23
4.29 Testcase 28	23
4.30 Testcase 29	23
4.31 Testcase 30	23

List of Tables

3.1 Các testcases với số lệnh R-I-J thực hiện và thời gian chạy chương trình . .	15
3.2 Các testcases với kết quả là heximal	16
3.3 Các testcases với kết quả là decimal	17

Listings

1 Giới thiệu đề tài

Đề 3: Cho 2 thanh ghi A (64-bit) và thanh ghi B (64-bit). Sử dụng hợp ngữ assembly MIPS để hiện thực phép nhân 2 thanh ghi đó. Kết quả được xuất ra console (hiển thị ở dạng HEX và dạng thập phân), bit cao của thanh ghi là bit dấu.

Thanh ghi 64-bit trong thanh ghi 32-bit Do MIPS chỉ hỗ trợ thanh ghi 32-bit nên ta sẽ tách mỗi số 64-bit ra thành hai thanh ghi 32-bit.

2 Giải thuật

2.1 Sơ lược cách giải quyết bài toán

Mỗi số 64-bit có thể được biểu diễn bằng 2 số 32-bit như sau:

$$X = a \cdot 2^{32} + b \quad (2.1)$$

$$Y = c \cdot 2^{32} + d \quad (2.2)$$

với a, b, c, d là những số 32-bit và X,Y là số 64-bit. Sau đó, ta nhân hai số 64-bit:

$$X \cdot Y = 2^{64} \cdot ac + 2^{32} \cdot ad + 2^{32} \cdot bc + bd \quad (2.3)$$

Như vậy, ta có thể nhân 2 số 64-bit trên hệ thống chỉ có thanh ghi 32-bit của MIPS

2.2 Chi tiết giải thuật

2.2.1 Lấy thông tin từ địa chỉ

Nội dung của các số 64-bit được lưu ở địa chỉ 'numA' và 'numB', ta có thể lấy về 32-bit lớn hoặc nhỏ để chứa vào từng registers. Sau đó, ta sẽ sử dụng lệnh *srl* để lấy bit dấu từ 32-bit lớn:

và dùng các lệnh *sll* cùng với *srl* để loại bỏ bit dấu từ hai số trước khi thực hiện phép nhân.

2.2.2 Thực hiện phép nhân

Phép nhân sẽ được thực hiện trong function *NhanAB* với địa chỉ hai số 64-bit nằm trong *\$a1* & *\$a2*.

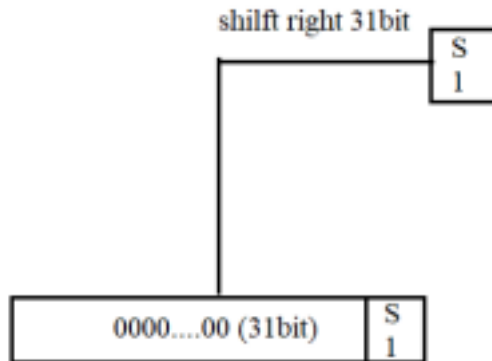


Figure 2.1: Mô phỏng lấy bit bên trong register

Trong MIPS, khi thực hiện phép nhân giữa 2 registers (dùng *mulu*) ta sẽ thu được kết quả là một số 64-bit và có thể đưa vào hai register bằng lệnh: *mfhi* & *mflo*. Từ phương trình (2.3), ta có thể sử dụng các registers 32-bit để thực hiện phép nhân cho hai số 64-bit. Ta sẽ đạt được kết quả phép nhân là một số trong khoảng từ 2^0 đến 2^{128} .

Ta chia phần kết quả của phép nhân ra làm 4 register, đi từ cao đến thấp, mỗi registers tương ứng 32-bit trong kết quả. Sau khi nhân hai registers, ta sẽ cộng dồn các bit nằm trong phần cao hơn của kết quả vào phần register cao hơn 2^{32} . Ví dụ, khi ta thực hiện phép nhân *bd* trong phương trình 2.3, có thể kết quả sẽ nhiều hơn 32-bit và phần lớn hơn đó sẽ nằm trong *mfhi* nên ta sẽ cộng 32-bit đó cho 32-bit low của phép nhân *ad* và phép nhân *bc* trong phương trình. Trong quá trình cộng các registers cần kiểm tra xem có tràn bit hay không, nếu có ta sẽ cộng 1 cho 32-bit cao hơn so với 32-bit dùng để cộng.

Sau khi thực hiện phép nhân và cộng dồn cho các registers, ta sẽ thu được kết quả trải dài trên 4 registers, ta dùng bit dấu lấy từ trước của hai số, thực hiện *xor* trên 2 bit để tìm dấu của kết quả, thêm lại vào bit cao nhất của kết quả. Cuối cùng, ta lưu 4 registers vào địa chỉ liên tiếp tạo thành kết quả hoàn chỉnh gồm tổng cộng 128-bit trong bộ nhớ.

2.2.3 Thực hiện xuất số lớn hơn 32-bit ra màn hình theo HEX

Vì MARS không hỗ trợ việc xuất số lớn hơn 32-bit ra màn hình theo định dạng HEX nên ta cần phải viết một hàm mới để hỗ trợ cho việc xuất dữ liệu. Vì HEX là hệ số $16 = 2^4$ nên 4-bit sẽ tương đương 1 HEX, ta dùng điều này để có thể tìm trong một chuỗi ký tự 4-bit tương đương với chữ số HEX nào.

Ta sẽ xuất số ra theo nhiều đợt, mỗi đợt xử lý 32-bit (hàm trong chương trình xử lý 64-bit trong 1 lần gọi hàm nhưng chia ra làm 2 đợt 32-bit). Mỗi đợt, ta lưu 32-bit từ địa chỉ vào register *\$t0* và thực hiện vòng lặp 8 lần (do 32-bit tương đương với 8 số HEX).

Mỗi vòng lặp, ta lấy ra 4-bit (1 HEX) cao nhất bằng cách sử dụng lệnh **tương đương** `andi $t3, $t0, 0xF0000000`. Sau đó thực hiện `srl $t3, $t3, 28` để có thể dùng lệnh **tương đương** `lb $t4, hex_chars($t3)` để tìm trong `hex_char::asciiz` “0123456789ABCDEF” kí tự HEX tương ứng với 4-bit đó. Cuối vòng lặp, ta lưu kết quả tìm được vào địa chỉ cần lưu, di chuyển con trỏ tại địa chỉ lưu, dịch `$t0` sang trái 4-bit và tiếp tục vòng lặp cho đến hết các kí tự HEX trong register (trong 32-bit của số).

2.2.4 Xuất số 128-bit ra màn hình theo decimal

Việc hiển thị một số 128-bit dưới dạng thập phân trong MIPS là một nhiệm vụ phức tạp và sẽ yêu cầu một chương trình tùy chỉnh để chuyển đổi và in số.

Thuật toán này tương tự như việc đọc nhưng theo thứ tự ngược lại. Ta nên tạo chuỗi từ phải sang trái, mỗi lần tạo 1 số, ta chia số 4-word (128-bit) cho 10, tính phần dư và phần nguyên. Phần dư được chuyển đổi thành ký tự tiếp theo và được lưu trữ trong một chuỗi để in sau này. Phần nguyên được ghi lại vào hai từ và được sử dụng trong lần lặp tiếp theo. Bước khó khăn là chia một số 128-bit (4-word) cho 10 để lấy phần nguyên và phần dư.

Cho A (128-bit) được tạo thành theo thứ tự từ cao nhất đến thấp nhất: x, y, z, t ; trong đó x chứa 32 bit cao nhất và t chứa 32 bit thấp nhất. Đó là, $A = 2^{96} \cdot x + 2^{64} \cdot y + 2^{32} \cdot z + t$. Sử dụng các lệnh `divu` trên x, y, z và t để xác định phần nguyên và phần dư khi chia số 128-bit A cho 10.

Ta thực hiện vòng lặp sau cho đến khi $t = 0$ để thực hiện phép chia số 128-bit A cho 10 và dùng phần dư lưu trữ trong một chuỗi để in sau này.

- Ta dùng `divu` để chia x cho 10, khi đó thu được phần nguyên trong $x1$ và phần dư trong $x2$.
- Ta sẽ lấy biến tạm tx có byte lớn nhất là phần dư trong phép chia trước, 7 byte còn lại là của y (có thể dùng kết hợp các lệnh `srl` và `or` để thực hiện điều này), sau đó chia tx cho 10.
- Tiếp theo, ta có biến tạm ty có byte lớn nhất cũng là phần dư trong phép chia trước, nhưng byte thứ 2 sẽ là byte cuối của y còn 6 byte cuối sẽ là 6 byte lớn nhất của z .
- Sau đó, ta có biến tạm tz có byte lớn nhất cũng là phần dư trong phép chia trước, byte thứ 2 & 3 là hai byte cuối của z còn 5 byte cuối sẽ là 5 byte lớn nhất của t .

- Ta có biến tạm tt có byte lớn nhất cũng là phần dư trong phép chia trước, 3 byte tiếp theo là 3 byte cuối của t . Khi đó, ta sẽ dư 4 byte cuối trong tt , ta sẽ dùng sll để dịch sang trái 4 byte. Ta thực hiện chia tt cho 10, phần dư sẽ là một số thập phân cần tìm.
- Ta lấy phần dư so với một chuỗi là *buffer*: `.asciiz "0123456789"` để quy đổi số thành char và lưu vào chuỗi để in sau này.
- Sau khi thực hiện các phép chia, ta có phần nguyên nằm trong 5 registers: x , tx , ty , tz , tt . Trong 4 registers cuối sẽ có 1 byte cao nhất sau khi thực hiện phép chia là 0 (điều này là do byte cao nhất của các registers này là phần dư của phép chia trước nên luôn đảm bảo sau khi chia 1 lần nữa các byte cao nhất này sẽ trở thành 0), khi đó ta dịch dồn các registers sang trái và sẽ đủ chỗ để tất cả các bit trong tt đưa vào trong tz .
- Cuối cùng, lưu tx , ty , tz vào lại trong y , z , t và tiếp tục thực hiện vòng lặp

Sau khi thực hiện vòng lặp, ta chỉ cần in chuỗi ngược lại số lần bằng số lần ghi vào chuỗi là ta sẽ hoàn thành đoạn chương trình đổi số 128-bit sang decimal bằng MIPS.

2.3 Các hàm sử dụng trong chương trình

2.3.1 NhanAB

Dưới đây là đoạn code mô phỏng việc nhân 2 thành ghi A và B.

- Đầu vào của hàm là hai registers $a0$ và $a1$ chứa địa chỉ của hai số 64-bit.
- Hàm xuất ra $v1$ là địa chỉ của kết quả là một số 128-bit.

```
300  NhanAB: #Bat dau nhan 2 thanh ghi 64-bit
301          #-----
302          #nhan fraction -> (t0: a) (t1: b) (t2: c) (t3: d)
303          #x = a*2^32 + b
304          #y = c*2^32 + d
305          #x * y = 2^64*ac + 2^32*ad + 2^32*bc + bd
306          lw $t0, 0($a1)
307          lw $t1, 4($a1)
308          lw $t2, 0($a2)
309          lw $t3, 4($a2)
310
311          multu $t0, $t2
312          mfhi $s0
313          mflo $s1 #2^64 ac
314
315          multu $t0, $t3
316          mfhi $s2
317          mflo $s3 #2^32 ad
318          #...
```

Figure 2.2: Đoạn chương trình nhân AB



2.3.2 bin_to_hex_string

Đoạn chương trình chuyển đổi nhị phân thành hex để xuất ra màn hình.

- Đầu vào của hàm là *a0* chứa địa chỉ của số 64-bit cần chuyển đổi thành hexadecimal.
- Đầu ra của hàm là *v1* chứa địa chỉ của chuỗi HEX là kết quả của hàm biến đổi.

```
370 bin_to_hex_string: #doi 64 bit tai dia chi luu trong $a0 thanh string tai ketqua_hex
371     # Dua vi tri luu ket qua vao $t1
372     #la $t1, ketqua_hex
373     lui $at, 0x1001
374     ori $t1, $at, 0x3f
375     lw $t0, 0($a0) #Xu ly 32 bit dau
376     # Bat dau lap de chuyen doi
377     li $t2, 8      # moi register co 8 so hex, lap 8 lan
378     convert_loop:
379         # Lay 4 bit lon nha cua $k0
380         #andi $t3, $t0, 0xF0000000
381         lui $at, 0xf000
382         ori $at, $at, 0x00000000
383         and $t3, $t0, $at
384
385         srl $t3, $t3, 28
386         # Su dung 4 bit nhu la index de tim ki tu hex tuong ung, nhu la hex_chars[$t3] trong C
387         #lb $t4, hex_chars($t3)
388         lui $at, 0x1001
389         addu $at, $at, $t3
390         lb $t4, 0x2e($at)
391         #...
```

Figure 2.3: Đoạn chương trình chuyển đổi binary thành hexadecimal



2.3.3 bin128_to_decimal

Đoạn chương trình chuyển đổi nhị phân của số 128-bit thành hệ thập phân rồi xuất ra màn hình

- Hàm lấy số 128-bit tại địa chỉ *numAB* là nơi lưu kết quả của phép nhân.
- Hàm xuất ra số 128-bit theo decimal ra màn hình.

```
175 bin128_to_decimal: #Chuyen doi so 128-bit ve he thap phan
176     #lw $t0, numAB
177     lui $at, 0x1001
178     lw $t0, 0x80($at)
179     sll $t0, $t0, 1 #Bo bit dau truoc khi chuyen doi sang thap phan
180     srl $t0, $t0, 1
181
182     #lw $t1, numAB+4
183     lui $at, 0x1001
184     lw $t1, 0x84($at)
185
186     #lw $t2, numAB+8
187     lui $at, 0x1001
188     lw $t2, 0x88($at)
189
190     #lw $t3, numAB+12
191     lui $at, 0x1001
192     lw $t3, 0x8c($at) #Luu so 128-bit ve 4 register tu cao den thap: $t0, $t1, $t2, $t3
193
194     addi $t9, $zero, 50 #Lap khoang 50 lan de dam bao khong bi lap vo han
195     #la $t8, array
196     lui $at, 0x1001
197     ori $t8, $at, 0x9b #Dua dia chi ghi ket qua vao $t8
198     #...
```

Figure 2.4: Đoạn chương trình chuyển đổi binary thành decimal

3 Kết quả testcases, tổng số lệnh và thời gian thực thi

3.1 Thời gian thực thi của chương trình

Ta có thể tính thời gian thực thi của một chương trình thông qua công thức.

Thời gian thực thi = tổng số chu kỳ thực thi \times thời gian của một chu kỳ

$$T_{CPU} = CC_2 \times T_C = CC_2 \times \frac{1}{CR} \quad (3.1)$$

Trong đó:

- CPI: Cycle Per Instruction (số chu kỳ trung bình thực thi trên một lệnh).
- CC_2 : Clock Cycle (tổng số chu kỳ đã được thực thi).
- T_C : Time of a cycle (thời gian một chu kỳ)
- CR: Clock Rate (số chu kỳ trên một giây)
- T_{CPU} : CPU time (thời gian xử lý của CPU)

Bên cạnh đó, ta còn công thức tính tổng số chu kỳ theo CPI và số lệnh như công thức bên dưới.

$$CC_2 = \sum_{k=1}^n CPI_1 \times IC_1 \quad (3.2)$$

Trong đó:

- CC_2 : tổng số chu kỳ đã được thực thi
- CPI_1 : số chu kỳ trung bình thực thi trên một lệnh thứ i
- IC_1 : số lệnh thứ i.

Dựa vào công thức (3.1) và (3.2) ta có thể suy ra công thức sau:

$$T_{CPU} = \frac{n_R + n_J + n_I}{3.4} \quad (3.3)$$

3.2 Kết quả các testcases

Ta đánh giá thời gian thực thi chương trình và tổng số lệnh R, I, J đã thông qua một số testcases sau.

Số A	Số B	R	I	J	Time (μs)
-1AC85785D3AC7B67	1144A7D4D6DF2786	2243	1189	79	1.033
36B953EACB5ED140	1D1C766DC03047F8	2242	1185	82	1.032
-3735A9E762CBCDAF	033143C83B8E4D1A	2191	1172	77	1.012
-1FC26BA3F2A8B746	1625B0C88D90CEC8	2243	1187	79	1.032
-4FFAB038DEB4213F	76C60335468CAFF3	2295	1204	81	1.053
-4AFE0CC866B26A21	0AEBAA330AE10D41	2243	1187	79	1.032
-179E9F62A055BFE6	61DF15B892E3392E	2295	1203	81	1.053
3743FB7038E22F0A	52B7CEAB87A878FD	2294	1200	84	1.052
463703744F0BEF8C	04F1999429AFAA42	2242	1185	82	1.032
6C0A8348B15E60B5	666534BA6768A144	2294	1200	84	1.052
31E2214C3C1FCA8D	5238F332751B0922	2294	1200	84	1.052
-0E22CC0946BC940D	19A2BB9C0931C88F	2243	1187	79	1.032
-0D9580F381B283B1	0DFDE9188CBA8E88	2191	1172	77	1.012
57205470A87EE8E9	03E1E885F9FD7B11	2242	1186	82	1.032
54E88F8EF63CBAE6	2D6D9F00F5DF52B3	2294	1202	84	1.053
-35D3EED4556C1819	00D4BD9D471857AB	2191	1171	77	1.011
-5A1018C9626A88C3	29CC35F19FBD91AD	2295	1202	81	1.052
5B3732C476A075F7	1A1D4E3823EBDF77	2294	1200	84	1.052
7478DA00159933C0	-3B2DAB2CBDCE7C46	2295	1202	81	1.052
-5D6A4955679BC2DB	571A6DB1C23D8C3C	2295	1202	81	1.052
54169FB8C2241F97	59205C561F8DBDA0	2294	1200	84	1.052
-1623D60D67D3AC53	3084EA79A9249E06	2243	1187	79	1.032
531BA3225FE03D48	-106EC4BA7CBB2D92	2243	1187	79	1.032
256C948BD6B70511	7B461E5BF841E8AC	2294	1201	84	1.053
-365064C2908CEB6C	6345BC8566BBB94E	2295	1202	81	1.052
-48D7A35CFDA8AD2C	5253C6F0EE6C2C8C	2295	1203	81	1.053
-5D68DE0FE64873AD	2982641E99373283	2295	1203	81	1.053
7C016E9A2F5538F1	0CF8B9BAF41BD046	2242	1185	82	1.032
1F690BFEA346F38E	0A733FF80422C782	2242	1184	82	1.032
4BA4514EB44E60EA	79DE87FF90C359F3	2294	1200	84	1.052

Table 3.1: Các testcases với số lệnh R-I-J thực hiện và thời gian chạy chương trình



Số A	Số B	Kết quả chương trình
-1AC85785D3AC7B67	1144A7D4D6DF2786	-01CE7C96181647BC146B27FFABCE48EA
36B953EACB5ED140	1D1C766DC03047F8	06391417A96441DBAD2F40D38BE37600E
-3735A9E762CBCDAF8	033143C83B8E4D1A	-00B040E175AEA69943381D62C1A286C6
-21FC26BA3F2A8B746	1625B0C88D90CEC8	-02BF624831016D592E4EBFE1FCA982B0
-4FFAB038DEB4213F	76C60335468CAFF3	-251B6A1F17A4ED472DD3D0DCF0299FCD
-4AFE0CC866B26A21	0AEBAA330AE10D41	-0332F5913842B7B6A79CB5A0B3B19F61
-179E9F62A055BFE6	61DF15B892E3392E	-0907AF92404D3C46B99C78363C14B154
3743FB7038E22F0A	52B7CEAB87A878FD	11DB74C06B7796D080C62B7D62252CE2
463703744F0BEF8C	04F1999429AFAA42	015B1FF796081A5227823BF07ADABA18
6C0A8348B15E60B5	666534BA6768A144	2B36E6B5904C1EC333E0A59D946B8514
31E2214C3C1FCA8D	5238F332751B0922	100587852E86D1A4520A5FB3E836DBBA
-0E22CC0946BC940D	19A2BB9C0931C88F	-016A624E0E88430646641AAAA27DDB43
-0D9580F381B283B1	0DFDE9188CBA8E88	-00BE10AB2E169EB0C652E09E667C2408
57205470A87EE8E9	03E1E885F9FD7B11	0152438A6E64427755BCDB73F99A6A79
54E88F8EF63CBAE6	2D6D9F00F5DF52B3	0F113CFE03DDF30C75CBA39A8CAE5AD2
-35D3EED4556C1819	00D4BD9D471857AB	-002CBB605115DE57681CEA02FCBC97B3
-5A1018C9626A88C3	29CC35F19FBD91AD	-0EB46BC657B9344946B7C300EE6BDEC7
5B3732C476A075F7	1A1D4E3823EBDF77	094E0C4786A54A833C78A21DFA15FED1
7478DA00159933C0	-3B2DAB2CBDCE7C46	-1AECA15D6BBCCC08434D47F881752680
-5D6A4955679BC2DB	571A6DB1C23D8C3C	-1FC8C3BE9779874662C896DFDB406F54
54169FB8C2241F97	59205C561F8DBDA0	1D467EAF9DF42D6C3FB16940B1113960
-1623D60D67D3AC53	3084EA79A9249E06	-043236E4299A79B58FBCF22719FD43F2
531BA3225FE03D48	-106EC4BA7CBB2D92	-0555AFEFF23C35903EC46140BC409B10
256C948BD6B70511	7B461E5BF841E8AC	12056B7FF9B2A8D60E6224F5E0DFCF6C
-365064C2908CEB6C	6345BC8566BBB94E	-150FE2A1BDE68C72933CA996E0F4C6E8
-48D7A35CFDA8AD2C	5253C6F0EE6C2C8C	-176CE8D916C09A0CC66587DBAC924410
-5D68DE0FE64873AD	2982641E99373283	-0F255F5970E497D03A9BA6163CD5FB87
7C016E9A2F5538F1	0CF8B9BAF41BD046	06488C89F7CC77EA212B44FAE4FC61E6
1F690BFEA346F38E	0A733FF80422C782	01483E819DFD4FDDC57FB705A237101C
4BA4514EB44E60EA	79DE87FF90C359F3	24026B1BDFCD0CBBB5B6E7B1D855581E

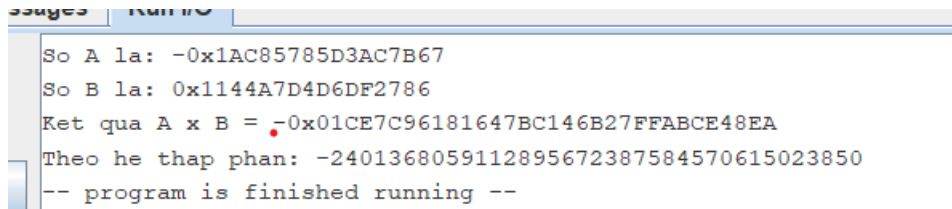
Table 3.2: Các testcases với kết quả là heximal



Số A	Số B	Kết quả chương trình
-1AC85785D3AC7B67	1144A7D4D6DF2786	2401368059112895672387584570615023850
36B953EACB5ED140	1D1C766DC03047F8	8271736418509961521643706820069651968
-3735A9E762CBCDAF8	033143C83B8E4D1A	-915160184074215060746430390508619462
-21FC26BA3F2A8B746	1625B0C88D90CEC8	-3652178087285333503699986184655176368
-4FFAB038DEB4213F	76C60335468CAFF3	-49323780258030892236855806143509405645
-4AFE0CC866B26A21	0AEBA0330AE10D41	-4252279526129765444262168074585743201
-179E9F62A055BFE6	61DF15B892E3392E	-12002959048966765337763621794154328404
3743FB7038E22F0A	52B7CEAB87A878FD	23736356944943393847127307465924095202
463703744F0BEF8C	04F1999429AFAA42	01802375380397986588941760011949488664
6C0A8348B15E60B5	666534BA6768A144	57441867188276383241259114656384648468
31E2214C3C1FCA8D	5238F332751B0922	21296358093892731994650522268544195514
-0E22CC0946BC940D	19A2BB9C0931C88F	-1881605323224962141121364437124373315
-0D9580F381B283B1	0DFDE9188CBA8E88	-986874483954721829984682873125086216
57205470A87EE8E9	03E1E885F9FD7B11	1756366227279200908678188859395828345
54E88F8EF63CBAE6	2D6D9F00F5DF52B3	20027926053095103287112502206018771666
-35D3EED4556C1819	00D4BD9D471857AB	-232261503369739862869946585883383731
-5A1018C9626A88C3	29CC35F19FBD91AD	-19545991307677953792713903951659654855
5B3732C476A075F7	1A1D4E3823EBDF77	12368300172815557696269968686596882129
7478DA00159933C0	-3B2DAB2CBDCE7C46	-35788582818530471381511374550122964608
-5D6A4955679BC2DB	571A6DB1C23D8C3C	-42248497411142102696742783790499262292
54169FB8C2241F97	59205C561F8DBDA0	38913642155282837300206835339950635360
-1623D60D67D3AC53	3084EA79A9249E06	-5577640153081690049971942309091951602
531BA3225FE03D48	-106EC4BA7CBB2D92	-7091053644079680536664973059066731280
256C948BD6B70511	7B461E5BF841E8AC	23954245765503143438332527785176715116
-365064C2908CEB6C	6345BC8566BBB94E	-27996269003437228859073620490239395560
-48D7A35CFDA8AD2C	5253C6F0EE6C2C8C	-31137734682355650697099342644425933840
-5D68DE0FE64873AD	2982641E99373283	-20132468835697010499376670569722674055
7C016E9A2F5538F1	0CF8B9BAF41BD046	8352063816816763088363911962188079590
1F690BFEA346F38E	0A733FF80422C782	1704341148323196138051395637845889052
4BA4514EB44E60EA	79DE87FF90C359F3	47864764868225066807880385787845761054

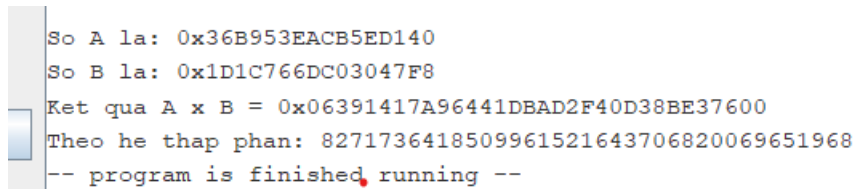
Table 3.3: Các testcases với kết quả là decimal

4 Hình ảnh INPUT/OUTPUT



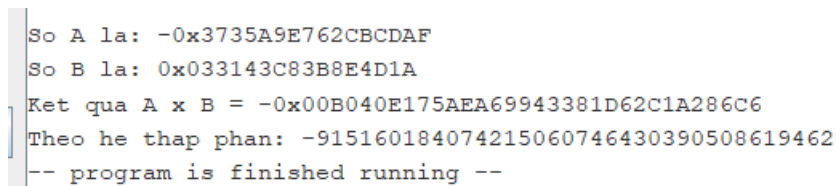
```
So A la: -0x1AC85785D3AC7B67
So B la: 0x1144A7D4D6DF2786
Ket qua A x B = -0x01CE7C96181647BC146B27FFABCE48EA
Theo he thap phan: -2401368059112895672387584570615023850
-- program is finished running --
```

Figure 4.1: Testcase 1



```
So A la: 0x36B953EACB5ED140
So B la: 0x1D1C766DC03047F8
Ket qua A x B = 0x06391417A96441DBAD2F40D38BE37600
Theo he thap phan: 8271736418509961521643706820069651968
-- program is finished running --
```

Figure 4.2: Testcase 2



```
So A la: -0x3735A9E762CBCDAF
So B la: 0x033143C83B8E4D1A
Ket qua A x B = -0x00B040E175AEA69943381D62C1A286C6
Theo he thap phan: -915160184074215060746430390508619462
-- program is finished running --
```

Figure 4.3: Testcase 3

```
So A la: -0x1FC26BA3F2A8B746
So B la: 0x1625B0C88D90CEC8
Ket qua A x B = -0x02BF624831016D592E4EBFE1FCA982B0
Theo he thap phan: -3652178087285333503699986184655176368
-- program is finished running --
```

Figure 4.4: Testcase 4

```
So A la: -0x4FFAB038DEB4213F
So B la: 0x76C60335468CAFF3
Ket qua A x B = -0x251B6A1F17A4ED472DD3D0DCF0299FCD
Theo he thap phan: -49323780258030892236855806143509405645
-- program is finished running --
```

Figure 4.5: Testcase 5

```
So A la: -0x4AFE0CC866B26A21
So B la: 0x0AEBAA330AE10D41
Ket qua A x B = -0x0332F5913842B7B6A79CB5A0B3B19F61
Theo he thap phan: -4252279526129765444262168074585743201
-- program is finished running --
```

Figure 4.6: Testcase 6

```
So A la: -0x179E9F62A055BFE6
So B la: 0x61DF15B892E3392E
Ket qua A x B = -0x0907AF92404D3C46B99C78363C14B154
Theo he thap phan: -12002959048966765337763621794154328404
-- program is finished running --
```

Figure 4.7: Testcase 7

```
So A la: 0x3743FB7038E22F0A
So B la: 0x52B7CEAB87A878FD
Ket qua A x B = 0x11DB74C06B7796D080C62B7D62252CE2
Theo he thap phan: 23736356944943393847127307465924095202
-- program is finished running --
```

Figure 4.8: Testcase 8

```
So A la: 0x463703744F0BEF8C
So B la: 0x04F1999429AFAA42
Ket qua A x B = 0x015B1FF796081A5227823BF07ADABA18
Theo he thap phan: 1802375380397986588941760011949488664
-- program is finished running --
```

Figure 4.9: Testcase 9



```
So A la: 0x6C0A8348B15E60B5
So B la: 0x666534BA6768A144
Ket qua A x B = 0x2B36E6B5904C1EC333E0A59D946B8514
Theo he thap phan: 57441867188276383241259114656384648468
-- program is finished running --
```

Figure 4.10: Testcase 10

```
So A la: 0x31E2214C3C1FCA8D
So B la: 0x5238F332751B0922
Ket qua A x B = 0x100587852E86D1A4520A5FB3E836DBBA
Theo he thap phan: 21296358093892731994650522268544195514
-- program is finished running --
```

Figure 4.11: Testcase 11

```
So A la: -0x0E22CC0946BC940D
So B la: 0x19A2BB9C0931C88F
Ket qua A x B = -0x016A624E0E88430646641AAAA27DDB43
Theo he thap phan: -1881605323224962141121364437124373315
-- program is finished running --
```

Figure 4.12: Testcase 12

```
So A la: -0x0E22CC0946BC940D
So B la: 0x19A2BB9C0931C88F
Ket qua A x B = -0x016A624E0E88430646641AAAA27DDB43
Theo he thap phan: -1881605323224962141121364437124373315
-- program is finished running --
```

Figure 4.13: Testcase 12

```
So A la: -0x0D9580F381B283B1
So B la: 0x0DFDE9188CBA8E88
Ket qua A x B = -0x00BE10AB2E169EB0C652E09E667C2408
Theo he thap phan: -986874483954721829984682873125086216
-- program is finished running --
```

Figure 4.14: Testcase 13

```
So A la: 0x57205470A87EE8E9
So B la: 0x03E1E885F9FD7B11
Ket qua A x B = 0x0152438A6E64427755BCDB73F99A6A79
Theo he thap phan: 1756366227279200908678188859395828345
-- program is finished running --
```

Figure 4.15: Testcase 14

```
So A la: 0x54E88F8EF63CBAE6
So B la: 0x2D6D9F00F5DF52B3
Ket qua A x B = 0x0F113CFE03DDF30C75CBA39A8CAE5AD2
Theo he thap phan: 20027926053095103287112502206018771666
-- program is finished running --
```

Figure 4.16: Testcase 15

```
So A la: -0x35D3EED4556C1819
So B la: 0x00D4BD9D471857AB
Ket qua A x B = -0x002CBB605115DE57681CEA02FCBC97B3
Theo he thap phan: -232261503369739862869946585883383731
-- program is finished running --
```

Figure 4.17: Testcase 16

```
So A la: -0x5A1018C9626A88C3
So B la: 0x29CC35F19FBD91AD
Ket qua A x B = -0x0EB46BC657B9344946B7C300EE6BDEC7
Theo he thap phan: -19545991307677953792713903951659654855
-- program is finished running --
```

Figure 4.18: Testcase 17

```
So A la: 0x5B3732C476A075F7
So B la: 0x1A1D4E3823EBDF77
Ket qua A x B = 0x094E0C4786A54A833C78A21DFA15FED1
Theo he thap phan: 12368300172815557696269968686596882129
-- program is finished running --
```

Figure 4.19: Testcase 18

```
So A la: -0x3B2DAB2CBDCE7C46
So B la: 0x7478DA00159933C0
Ket qua A x B = -0x1AECA15D6BBCCC08434D47F881752680
Theo he thap phan: -35788582818530471381511374550122964608
-- program is finished running --
```

Figure 4.20: Testcase 19

```
So A la: -0x5D6A4955679BC2DB
So B la: 0x571A6DB1C23D8C3C
Ket qua A x B = -0x1FC8C3BE9779874662C896DFDB406F54
Theo he thap phan: -42248497411142102696742783790499262292
-- program is finished running --
```

Figure 4.21: Testcase 20

```
So A la: 0x54169FB8C2241F97
So B la: 0x59205C561F8DBDA0
Ket qua A x B = 0x1D467EAF9DF42D6C3FB16940B1113960
Theo he thap phan: 38913642155282837300206835339950635360
-- program is finished running --
```

Figure 4.22: Testcase 21

```
So A la: -0x1623D60D67D3AC53
So B la: 0x3084EA79A9249E06
Ket qua A x B = -0x043236E4299A79B58FBCF22719FD43F2
Theo he thap phan: -5577640153081690049971942309091951602
-- program is finished running --
```

Figure 4.23: Testcase 22

```
So A la: -0x106EC4BA7CBB2D92
So B la: 0x531BA3225FE03D48
Ket qua A x B = -0x0555AFEFF23C35903EC46140BC409B10
Theo he thap phan: -7091053644079680536664973059066731280
-- program is finished running --
```

Figure 4.24: Testcase 23

```
So A la: 0x256C948BD6B70511
So B la: 0x7B461E5BF841E8AC
Ket qua A x B = 0x12056B7FF9B2A8D60E6224F5E0DFCF6C
Theo he thap phan: 23954245765503143438332527785176715116
-- program is finished running --
```

Figure 4.25: Testcase 24

```
So A la: -0x365064C2908CEB6C
So B la: 0x6345BC8566BBB94E
Ket qua A x B = -0x150FE2A1BDE68C72933CA996E0F4C6E8
Theo he thap phan: -27996269003437228859073620490239395560
-- program is finished running --
```

Figure 4.26: Testcase 25

```
So A la: -0x48D7A35CFDA8AD2C
So B la: 0x5253C6F0EE6C2C8C
Ket qua A x B = -0x176CE8D916C09A0CC66587DBAC924410
Theo he thap phan: -31137734682355650697099342644425933840
-- program is finished running --
```

Figure 4.27: Testcase 26

```
So A la: -0x5D68DE0FE64873AD
So B la: 0x2982641E99373283
Ket qua A x B = -0x0F255F5970E497D03A9BA6163CD5FB87
Theo he thap phan: -20132468835697010499376670569722674055
-- program is finished running --
```

Figure 4.28: Testcase 27

```
So A la: 0x7C016E9A2F5538F1
So B la: 0x0CF8B9BAF41BD046
Ket qua A x B = 0x06488C89F7CC77EA212B44FAE4FC61E6
Theo he thap phan: 8352063816816763088363911962188079590
-- program is finished running --
```

Figure 4.29: Testcase 28

```
So A la: 0x1F690BFEA346F38E
So B la: 0x0A733FF80422C782
Ket qua A x B = 0x01483E819DFD4FD5C57FB705A237101C
Theo he thap phan: 1704341148323196138051395637845889052
-- program is finished running --
```

Figure 4.30: Testcase 29

```
So A la: 0x4BA4514EB44E60EA
So B la: 0x79DE87FF90C359F3
Ket qua A x B = 0x24026B1BDFCD0CBBB5B6E7B1D855581E
Theo he thap phan: 47864764868225066807880385787845761054
-- program is finished running --
```

Figure 4.31: Testcase 30