۱)

32 bit = 4 byte = 2 word = 1 double word => dd

اگر فرض کنیم A,B,C,D هرکدام یک word هستند و اولین double word ما AB و دومی CD است، خواهیم داشت:

```
     A B
x
     C D
_____________
         w2 w1
     w4 w3
     w6 w5
w8 w7
```

به این گونه که جواب DxB برابر w2w1، جواب DxA برابر w4w3، جواب CxB برابر w6w5 و جواب CxA برابر w8w7 شده است.

پس جواب کلمه اول جواب w1،

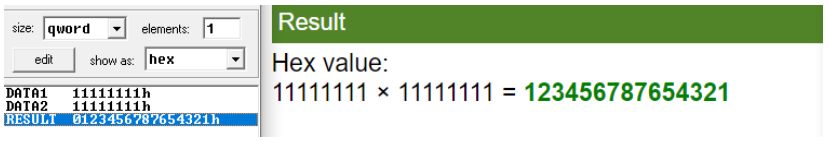
کلمه دوم جواب w2+w3+s5،

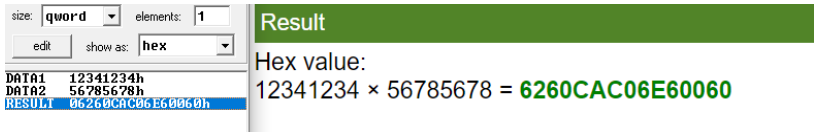
کلمه سوم w4+w6+w7+carry(from w2+w3)+carry(from (w2+w3)+w5)

کلمه چهارم w8+carry(from w6+w7)+carry(from (w6+w7)+w4)

خواهند بود. نحوه هندل شدن این موارد در رجیسترها، با کامنت‌های موجود رو به روی کد، قابل مشاهده است.

چک کردن جواب برنامه با جواب واقعی (به دست آمده از طریق محاسبه‌گر آنلاین)





۲) این کد از ۴ procedure تشکیل شده است.

MAIN: در آن یک رشته چاپ و سپس عدد در مبنای ۱۰ خوانده شده توسط DECIMAL_INPUT، در MAXCOUNT ریخته میشود. در آخر نیز تابع PRINT_PRIME را صدا میزند.

DECIMAL_INPUT: با استفاده از جدول کد ascii مشاهده میکنیم که کد "0" برابر 48 یا 30H و کد "9" برابر 57 یا 39H و کد carriage return برابر 13 یا 0DH است. پس وقتی "0" وارد شود ما عدد 30H را خواهیم داشت پس برای تبدیل کردن ascii آن به دسیمال، باید 30H را با 0FH، AND کنیم (یا میشد 30H از آن کم کنیم، که این روش در سوال ۳-۲ استفاده میشود). همچنین هر کاراکتر جدید را که میخوانیم عدد حاصل از مرحله قبل را ضربدر ۱۰ و سپس بعلاوه عدد وارد شده میکنیم تا عدد حاصل بدست آید.

DECIMAL_OUTPUT: اول برای رفتن به خط جدید باید یک خط جدید چاپ کرد و سپس carriage return کرد. کد آن به صورت زیر است:

```
mov dx,13
mov ah,2
int 21h
mov dx,10
mov ah,2
int 21h
```

که آن را به این شکل میتوان خلاصه کرد:

```
.DATA
LINEFEED DB 13, 10, "$"
.CODE
MOV AH, 09
MOV DX, OFFSET LINEFEED
INT 21H
```

(البته میشد قبل از رشته ای که چاپ میشود نیز 13,10, گذاشت – مانند سوال ۳-۲)

این تابع نیز عملاً شبیه DECIMAL_INPUT کار میکند. به طوری که برای چاپ کردن هر کاراکتر باید عددمان را تقسیم بر ۱۰ کنیم و باقی مانده را تبدیل به ascii و سپس پرینت کنیم.

PRINT_PRIME: از عدد ۲ (در MIN ریخته میشود) چک کردن خود را شروع کرده به اینگونه که از MIN/2 تا 2 را در نظر گرفته و MIN را به آن تقسیم میکند. اگر با تقسیم شدن به عدد ۲ هم باقی مانده‌ای نداشت، پس این عدد اول است، آن را با استفاده از DECIMAL_OUTPUT چاپ میکند و به PRIMECOUNT اضافه میکند و سراغ عدد بعدی (MIN++) میرود. وگرنه اگر باقی مانده صفر شد یعنی عدد اول نیست پس همینجا سراغ عدد بعدی (MIN++) میرود. این فرایند را آنقدر ادامه میدهد تا PRIMECOUNT = MAXCOUNT شود که یعنی به تعداد MAXCOUNT عدد اول پیدا شده است.

نمونه خروجی:

۳)

الف) برای حل این سوال عدد BCD و باینری را به هگز تبدیل میکنیم. عدد باینتری ۱۶ بیتی در ۲ بایت جا میشود. عدد BCD هم (به گفته TA) در ۱۶ بیت (۲ بایت) جا میشود پس n حداکثر ۱۶ است. پس باید تقسیم word بر word انجام شود.

در این کد عدد BCD باید درست وارد شود (ارور آن گرفته نمیشود یعنی با آن شبیه باینری رفتار میشود). اگر میخواستیم ارور بگیریم چون n را داریم، میتوانستیم ۴تا ۴تا از راست جدا کنیم و ببینیم اگر بیت ۴ برابر ۱ وارد شده، بیت ۳و۲ باید ۰ وارد شوند.

ب) برای تبدیل رشته به عدد در این قسمت باید بدانیم که طول رشته‌ای که کاربر وارد کرده در offset string + 1 و اولین کاراکتری که وارد کرده در offset string + 2 قرار دارد. با استفاده از این موارد میتوانیم عدد موجود در رشته را بدست بیاوریم که روش آن مثل سوال‌های قبل است.

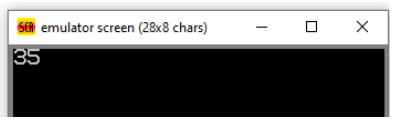
۴) برای اینکه چک کنیم عددی فرد است یا خیر کافیست باقی مانده آن به عدد ۲ را در نظر بگیریم. پس اعداد را تک تک چک کرده و اگر فرد بودند، به جواب اضافه میکنیم. برای آرایه‌ی:

ARR DW 10, 7, 10H, 11, 11H

خواهیم داشت:

7 + 11 + 11H = 7 + 11 + 17 = 35

```
.DATA
    ARR DW 10, 7, 10H, 11, 11H
    LEN DW 5
    SUM DW 0
.CODE
```



۵) برای محاسبه فاکتوریل یک عدد داریم:

n! = n * (n-1)!

و این محاسبه‌ی factorial(n-1) را تا جایی ادامه میدهیم که n>1 در غیر این صورت وقتی n=1 یا n=0 باشد جواب فکتوریل "۱" برگردانده میشود.