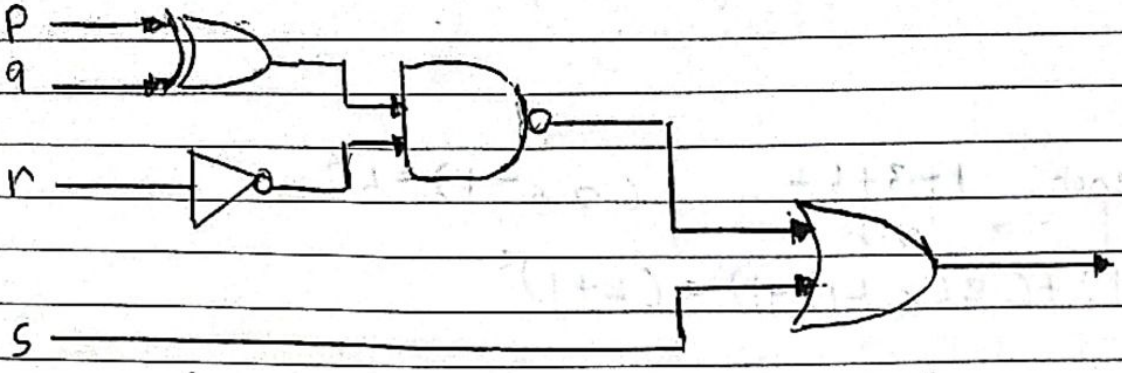
UJI KOMPETENSI 2 Mohammad Farid Hendianto 2200018401 A

1.



Dari gerbang logika tersebut, didapatkan persamaan boolean
$((P \oplus q) \uparrow \neg r) \vee s$

Berikut adalah tabel kebenarannya

P	q	r	s	$P \oplus q$	$\neg r$	$(P \oplus q) \neg r$	$((P \oplus q) \neg r) \vee s$
1	1	1	1	0	0	1	1
1	1	1	0	0	0	1	1
1	1	0	1	0	1	1	1
1	1	0	0	0	1	1	1
1	0	1	1	1	0	1	1
1	0	1	0	1	0	1	1
1	0	0	1	1	1	0	1
1	0	0	0	1	1	0	0
0	1	1	1	1	0	1	1
0	1	1	0	1	0	1	1
0	1	0	1	1	1	0	1
0	1	0	0	1	1	0	0
0	0	1	1	0	0	1	1
0	0	1	0	0	0	1	1
0	0	0	1	0	1	1	1
0	0	0	0	0	1	1	1

Catatan:

- $\oplus$ XOR (Exclusive OR) adalah operasi logika yang menghasilkan benar jika jumlah input yang benar adalah ganjil.
- $\uparrow$ NAND (NOT AND) adalah operasi logika yang menghasilkan salah jika semua inputnya benar, dan benar jika setidaknya satu input salah.
- $\neg$ NOT adalah operasi logika yang menghasilkan kebalikan dari inputnya
- $\vee$ OR adalah operasi logika yang menghasilkan benar jika setidaknya satu inputnya benar.

2. Berikut adalah program dengan bahasa mesin (menggunakan TASM) dengan ketentuan A, B, C, D, E, Y adalah register dengan perintah program Y = (A x B + C) / (D x E)

```
name "Uji Kompetensi 2 - Oleh Mohammad Farid Hendianto 2200018401"

; this macro prints a char in AL and advances
; the current cursor position;
PUTC  MACRO char
      PUSH  AX
      MOV   AL, char
      MOV   AH, 0Eh
      INT   10H
      POP   AX
ENDM
;;;;;;;;;;;;;;;;;;;;;;
precision = 30      ; max digits after the dot
org 100h
jmp start
; define variables:
msg0 db "Fitur: ", 0Dh, 0Ah
     db "+ Bisa menggunakan bilangan bulat positif maupun negatif: ", 0Dh, 0Ah
     db "+ Hasil bisa berupa Floating number: ", 0Dh, 0Ah
     db "Pembatasan:" 0Dh, 0Ah
     db "- Nilai pada variabel A,B,C,D,E tidak bisa mengandung Floating number", 0Dh, 0Ah, 0Dh, 0Ah
     db "Y = (A x B + C) / (D x E)" 0Dh, 0Ah, '$'
msg1 db 0dh, 0Ah, 'Maka nilai Y = $'
msg2 db 0dh, 0Ah, 'Arigatov Nee OwO~', 0Dh, 0Ah, '$'
; Variable on register
A dw ?
B dw ?
C dw ?
D dw ?
E dw ?
string_input_A  DB  'Masukkan A : $'
string_input_B  DB  13, 10, 'Masukkan B : $'
string_input_C  DB  13, 10, 'Masukkan C : $'
string_input_D  DB  13, 10, 'Masukkan D : $'
```

Source Code:
https://github.com/IRedDragonICY/Computer-Architecture

```
string_input_E DB 13,10,'Masukkan E : $'
string_res     DB 13,10,'Y = $'
TEMP dw ?
TEMP2 dw ?
ten            dw 10    ;used as multiplier
four           dw 4     ;used as divider
make_minus     db ?     ;used as a flag in procedures
start:
mov dx, offset msg0
mov ah, 9
int 21h
lea dx, string_input_A  ;store A number
mov ah, 09h             ;output string at ds:dx
int 21h
;get the multidigit signed number
;from the keyboard, and store
;the result in cx register:
call scan_num
mov A, cx
;store B number:
lea dx, string_input_B
mov ah, 09h
int 21h
call scan_num
mov B, cx
;store C number:
lea dx, string_input_C
mov ah, 09h
int 21h
call scan_num
mov C, cx
;store D number
lea dx, string_input_D
mov ah, 09h
int 21h
call scan_num
mov D, cx
```

```
; store E number:
lea dx, string_input_E
mov ah, 09h
int 21h
call scan_num
mov E, cx
; result step
mov ah, 09h
mov dx, OFFSET string_res
int 21h
; print (
mov ah, 02h
mov dl, '('
int 21h
; print var A
mov ax, A
call print_num
; print x
mov ah, 02h
mov dl, 'x'
int 21h
; print var B
mov ax, B
call print_num
; print +
mov ah, 02h
mov dl, '+'
int 21h
; print var C
mov ax, C
call print_num
; print )
mov ah, 02h
mov dl, ')'
int 21h
; print /
mov ah, 02h
mov dl, '/'
```

```
int 21h
;print (
mov ah, 02h
mov dl, '('
int 21h
;print var D
mov ax, D
call print_num
;print x
mov ah, 02h
mov dl, 'x'
int 21h
;print var E
mov ax, E
call print_num
;print )
mov ah, 02h
mov dl, ')'
int 21h
;new line:
putc 0Dh
putc 0Ah
lea dx, msg1
mov ah, 09h    ;output string at ds:dx
int 21h
;A x B
mov ax, A
imul B ; (dx ax) = ax * A.
mov TEMP, ax
;dx is ignored (calc with tiny numbers only).
; +C
mov ax, TEMP
add ax, C
mov TEM, ax
; D x E
mov ax, D
imul E
mov TEMP2, ax
```

```
; Y = (A x B + C) / (D x C)
; dx is ignored (calc works with tiny integer numbers only).
mov ax, TEMP
xor dx, dx
; check the sign, make dx:ax negative if ax is negative:
cmp ax, 0
jns not_signed
not dx
not_signed:
mov bx, TEMP2   ; divider is in bx.
; 'A' is in dx:ax
; 'B' is in bx.
idiv bx   ; ax = dx:ax / bx   (dx - remainder)
; 'A/B' is in ax
; remainder is in dx
push dx   ; store the remainders
pop dx
; print 'A/B' as float:
; ax - whole part
; dx - remainder
; bx - divider
call print_float   ; print ax value
jmp exit
exit:
; output of a string at ds:dx
lea dx, msg2
mov ah, 09h
int 21h
; wait for any key....
mov ah, 0
int 16h
ret   ; return back to os.
; ********** ********
; gets the multi-digit SIGNED number from the keyboard,
; and stores the result in CX register:
SCAN_NUM   PROC NEAR
PUSH DX
PUSH AX
```

```
PUSH SI
MOV CX,0
;reset flag:
MOV CS:make_minus,0
next_digit:
;get char from keyboard
;into AL:
MOV AH,00h
INT 16h
;and print it
MOV AH,0Eh
INT 10h
;check for MINUS:
CMP AL,'-'
JE  set_minus
;check for ENTER key
CMP AL,0Dh  ;carriage return?
JNE not_cr
JMP stop_input
not_cr:
CMP AL,8                ;'BACKSPACE' pressed?
JNE backspacechecked
MOV DX,0                ;remove last digit by
MOV AX,CX               ;division:
DIV CS:ten              ;AX=DX:AX/10 (DX-rem)
mov CX,AX
PUTC ' '                ;clear position
PUTC 8                  ;backspace again.
JMP next_digit
backspace_checked:
;allow only digits:
CMP AL,'0'
JAE ok_AE_0
JMP remove_not_digit
ok_AE_0:
CMP AL,'9'
JBE ok_digit
```

```
remove_not_digit:
PUTC 8     ; backspace
PUTC ' '   ; clear last entered not digit
PUTC 8     ; backspace again.
JMP next_digit ; wait for next input
ok_digit:
; multiply CX by 10 (first time the result is zero)
PUSH AX
MOV AX, CX
MUL CS:ten          ; DX:AX = AX * 10
MOV CX, AX
POP AX
; check if the number is too big
; (result should be 16 bits)
CMP DX, 0
JNE too_big
; convert from ASCII code:
SUB AL, 30h
; add AL to CX:
MOV AH, 0
MOV DX, CX      ; backup, in case the result will be too big.
ADD CX, AX
JC too_big2     ; jump if the number is too big
JMP next_digit
set_minus:
MOV CS:make_minus, 1
JMP next_digit
too_big2:
MOV CX, DX ; restore the backuped value before add
MOV DX, 0  ; DX was zero before backup!
too_big:
MOV AX, CX
DIV CS:ten ; reverse last DX:AX = AX * 10, make AX = DX:AX / 10
MOV CX, AX
PUTC 8     ; backspace
PUTC ' '   ; clear last entered digit
PUTC 8     ; backspace again.
JMP next_digit ; wait for Enter/Backspace.
```

```
stop_input:
;check flag
CMP CS:make_minus,0
JE not_minus
NEG CX
not_minus:
POP SI
POP AX
POP DX
RET
SCAN_NUM ENDP

GET_STRING PROC NEAR
PUSH AX
PUSH CX
PUSH DI
PUSH DX
MOV CX,0          ;char counter
CMP DX,1          ;buffer too small?
JBE empty_buffer
DEC DX            ;reserve space for last zero.
;==========
;eternal loop to get
;and processes key presses:
wait_for_key:
MOV AH,0          ;get pressed
INT 16h
CMP AL,0Dh        ;'RETURN' pressed?
JZ exit_GET_STRING
CMP AL,8          ;'BACKSPACE' pressed?
JNE add_to_buffer
JCXZ wait_for_key ;nothing to remove!
DEC CX
DEC DI
PUTC 8            ;backspace
PUTC ' '          ;clear position
PUTC 8            ;backspace again
JMP wait_for_key
```

```
add_to_buffer:
CMP CX,DX       ; buffer is full?
JAE wait_for_key ; if so wait for 'BACKSPACE' or 'RETURN'.
MOV [DI], AL
INC DI
INC CX
;print the key
MOV AH,0Eh
INT 10h
JMP wait_for_key
;=>=>=>=>=>=>=>=>=>=>
exit_GET_STRING:
;terminate by null:
MOV [DI],0
empty_buffer:
POP DX
POP DI
POP CX
POP AX
RET
GET_STRING ENDP

;*****************
; prints number in ax and it's fraction in dx.
; used to print reminder of 'div/idiv bx'.
; ax = whole part.
; dx = remainder
; bx = the divider that was used to get the remainder from divident
Print_Float PROC near
Push cx
Push dx
;because the remainder takes the sign of divident
;its sign should be inverted when divider is negative
; (-)/(-) = (+)
; (+)/(-) = (-)
; (-)/(+) = (-)
; (+)/(+) = (+)
```

```
cmp bx,0
jns div_not_signed
neg dx                  ; make remainder positive
div_not_signed:
;Print_num Procedure does not print the '-'
;when the whole part is '0' (even if the remainder is
;negative) this code fixes it:
cmp ax,0
jne checked     ;ax<>0
cmp dx,0
jns checked     ;ax=0 and dx>=0
Push dx
mov dl,'-'
call write_char ;print '-'
Pop dx
checked:
;Print whole part:
call Print_num
;if remainder=0, then no need to print it:
cmp dx,0
je done
Push dx
;Print dot, after the number:
mov dl,'.'
call write_char
Pop dx
;Print digits after the dot:
mov cx,precision
call Print_Fraction
done:
Pop dx
Pop cx
ret
Print_Float endP
```

```
;=========== ==========
; Prints dx as fraction of division by dx
; dx - remainder
; bx - divider
; cx - maximum number of digits after the dot.
print_fraction proc near
push ax
push dx
next_fraction:
;check if all digits are already printed
cmp cx,0
jz end_rem
dec cx ;decrease digit counter.
;when remainder is '0' no need to continue:
cmp dx,0
je end_rem
mov ax,dx
mov dx,dx
cmp ax,0
jns not_sig1
not dx
not_sig1:
imul ten ;dx:ax = ax*10
idiv bx ;ax=dx:ax/bx (dx - remainder)
push dx ;store remainder.
mov dx,ax
cmp dx,0
jns not_sig2
neg dx
not_sig2:
add dl,30h ;convert to ascii code.
call write_char ;print dl.
pop dx
jmp next_fraction
end_rem:
pop dx
pop ax
ret
print_fraction endp
```

```
;**********************
; this procedure prints number in ax
; used with print_numx to print "0" and sign
; this procedure also stores the original ax,
; that is modified by print_numx.
print_num proc near
push dx
push ax
cmp ax,0
jnz not_zero
mov dl,'0'
call write_char
jmp printed
not_zero:
; the check sign of ax,
; make absolute if it's negative:
cmp ax,0
jns positive
neg ax
mov dl,'-'
call write_char
positive:
call print_numx
printed:
pop ax
pop dx
ret
print_num endp
;**********************
; prints out a number in ax (not just a single digit)
; allowed values 1 to 65535 (FFFF)
; (result of /10000 should be the left digit or "0").
; modifies ax (after the procedure ax=0)
print_numx proc near
push bx
push cx
push dx
```

```
; flag to prevent printing zeros before number:
mov cx, 1
mov bx, 10000       ; 2710h - divider.
; check if ax is zero, if zero go to end_show
cmp ax, 0
jz end_show
begin_print:
; check divider (if zero go to end_show):
cmp bx, 0
jz end_show
; avoid printing zeros before number:
cmp cx, 0
je calc
; if ax < bx then result of div will be zero:
cmp ax, bx
jb skip
calc:
xor cx, cx          ; set flag.
xor dx, dx
div bx              ; ax = dx:ax/bx  (dx = remainder)
; print last digit
; ah is always zero, so it's ignored
push dx
mov dl, al
add dl, 30h         ; convert to ascii code
call write_char
pop dx
mov ax, dx          ; get remainder from last div.
skip:
; calculate bx = bx/10
push ax
xor dx, dx
mov ax, bx
div ten             ; ax = dx:ax/10  (dx = remainder)
mov bx, ax
pop ax
jmp begin_print
```

```
end_show:
Pop dx
Pop cx
Pop bx
ret
Print_numx endp
;**********************
; reads 1 char from the keyboard into al
; (modifies nx!!)
read_char proc near
mov ah, 01h
int 21h
ret
read_char endp
;**********************
; prints out single char (ascii code should be in dl)
write_char Proc near
Push ax
mov ah, 02h
int 21h
Pop ax
ret
write_char endp
```

edit: E:\Program Files\emu8086\MySource\soal2integerbanyak.asm

file edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator convertor options help About

```
001 name "Uji Kompetensi 2 - Oleh Mohammad Farid Hendianto 2200018401"
002
003 ; this macro prints a char in AL and advances
004 ; the current cursor position:
005 PUTC    MACRO   char
006         PUSH    AX
007         MOV     AL, char
008         MOV     AH, 0Eh
009         INT     10h
010         POP     AX
011 ENDM
012 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
013
014 precision=30    ; max digits after the dot.
015
016 org 100h
017
018 jmp start
019
020 ; define variables:
021 msg0 db "Fitur:",0Dh,0Ah
022      db "+ Bisa menggunakan bilangan bulat positif maupun negatif:",0Dh,0Ah
023      db "+ Hasil bisa berupa floating number:",0Dh,0Ah
024      db "Pembatasan:",0Dh,0Ah
025      db "- Nilai pada var A,B,C,D,E tidak bisa mengandung floating number",0Dh,0Ah,0Dh,0Ah
026      db "Y = (A x B + C) / ( D x E)",0Dh,0Ah,'$'
027 msg1 db  0dh,0ah , 'Maka nilai Y =  $'
028 msg2 db  0dh,0ah ,'Arigatou Nee OwO ~ ', 0Dh,0Ah, '$'
029
030 ; Varible on Registers
031 A dw ?
032 B dw ?
033 C dw ?
034 D dw ?
```

line: 90 col: 86 drag a file here to open

emulator screen (118x39 chars)

```
Fitur:
+ Bisa menggunakan bilangan bulat positif maupun negatif:
+ Hasil bisa berupa floating number:
Pembatasan:
- Nilai pada var A,B,C,D,E tidak bisa mengandung floating number

Y = (A x B + C) / ( D x E)
Masukkan A: 1
Masukkan B: 2
Masukkan C: 3
Masukkan D: 4
Masukkan E: 5
Y=(1x2+3)/(4x5)

Maka nilai Y =  0.25
Arigatou Nee OwO ~
```

clear screen change font 0/16

((1(2)+3))/(4(5))

symbolab.com/solver/step-by-step/%5Cfrac%7B%5Cleft(1%5Cl...

Update

Apps AI Software School Book Event Anime Coding Draw Security All Bookmarks

Sy EN Upgrade

$\frac{(1(2)+3)}{4(5)}$ Go

Steps Examples

$\frac{(1(2)+3)}{4(5)}$

Solution

$\frac{1}{4}$ +1

decimal 0.25

Save Noteb Sig Send us Fee

emulator: Calculating.com

file math debug view external virtual devices virtual drive help

Load reload step back waiting for input stop step delay ms: 0

registers

	H	L
AX	00	24
BX	00	14
CX	00	05
DX	01	ED

CS	F400
IP	01C0
SS	0700
SP	FFF8
BP	0000
SI	0000
DI	0000
DS	0700
ES	0700

F400:01C0

```
F41C0: FF 255 RES
F41C1: FF 255 RES
F41C2: CD 205 =
F41C3: 16 022 ▬
F41C4: CF 207 ╧
F41C5: 00 000 NULL
F41C6: 00 000 NULL
F41C7: 00 000 NULL
F41C8: 00 000 NULL
F41C9: 00 000 NULL
F41CA: 00 000 NULL
F41CB: 00 000 NULL
F41CC: 00 000 NULL
F41CD: 00 000 NULL
F41CE: 00 000 NULL
F41CF: 00 000 NULL
F41D0: FF 255 RES
F41D1: FF 255 RES
F41D2: CD 205 =
F41D3: 11 017 ◄
F41D4: CF 207 ╧
F41D5: 00 000 NULL
F41D6: 00 000 NULL
F41D7: 00 000 NULL
F41D8: 00 000 NULL
F41D9: 00 000 NULL
F41DA: 00 000 NULL
F41DB: 00 000 NULL
F41DC: 00 000 NULL
F41DD: 00 000 NULL
```

F400:01C0

```
BIOS DI
INT 016h
IRET
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
ADC DI, CX
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
ADC AX, 000CFh
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
...
```

screen source reset aux vars debug stack flags

edit: E:\Program Files\emu8086\MySource\soal2integerbanyak.asm

file edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator convertor options help About

```
D dw ?
E dw ?

string_input_A    DB 'Masukkan A : $'
string_input_B    DB 13,10,'Masukkan B : $'
string_input_C    DB 13,10,'Masukkan C : $'
string_input_D    DB 13,10,'Masukkan D : $'
string_input_E    DB 13,10,'Masukkan E : $'
string_res        DB 13,10,'Y=$'
sring_undefined   DB 'Undefined$'
; Temporary Variables;
TEMP dw ?
TEMP2 dw ?
ten               dw      10        ; used as multiplier.
four              dw      4         ; used as divider.
make_minus        db      ?         ; used as a flag in procedures.

start:

        mov dx, offset msg0
        mov ah, 9
        int 21h

        ; store A number:
        lea dx, string_input_A
        mov ah, 09h     ; output string at ds:dx
        int 21h

        ; get the multi-digit signed number
        ; from the keyboard, and store
        ; the result in cx register:
        call scan_num
        mov A,cx
```

line: 90 col: 86 drag a file here to open

emulator screen (118x39 chars)

```
Fitur:
+ Bisa menggunakan bilangan bulat positif maupun negatif:
+ Hasil bisa berupa floating number:
Pembatasan:
- Nilai pada var A,B,C,D,E tidak bisa mengandung floating number

Y = (A x B + C) / ( D x E)
Masukkan A: 42
Masukkan B: -52
Masukkan C: 33
Masukkan D: 5
Masukkan E: 12
Y=(42x-52+33)/(5x12)

Maka nilai Y =  -35.85
Arigatou Nee OwO ~
```

clear screen change font

((42(-52)+33))/(5(12))

symbolab.com/solver/step-by-step/%5Cfrac%7B%5Cleft(42%5...

Apps AI Software School Book Event Anime Coding Draw Security All Bookmarks

Update

Sy EN Upgrade

$$\frac{(42(-52)+33)}{5(12)}$$

Solution

$$-35\frac{17}{20}$$

+1

decimal -35.85

Show Steps

Send us Fee

emulator: Calculating.com

file math debug view external virtual devices virtual drive help

Load reload step back waiting for input stop step delay ms: 0

registers

	H	L
AX	00	24
BX	00	3C
CX	00	0C
DX	01	ED

CS	F400
IP	01C0
SS	0700
SP	FFF8
BP	0000
SI	0000
DI	0000
DS	0700
ES	0700

F400:01C0

```
F41C0: FF 255 RES
F41C1: FF 255 RES
F41C2: CD 205 =
F41C3: 16 022 ▬
F41C4: CF 207 ╧
F41C5: 00 000 NULL
F41C6: 00 000 NULL
F41C7: 00 000 NULL
F41C8: 00 000 NULL
F41C9: 00 000 NULL
F41CA: 00 000 NULL
F41CB: 00 000 NULL
F41CC: 00 000 NULL
F41CD: 00 000 NULL
F41CE: 00 000 NULL
F41CF: 00 000 NULL
F41D0: FF 255 RES
F41D1: FF 255 RES
F41D2: CD 205 =
F41D3: 11 017 ◄
F41D4: CF 207 ╧
F41D5: 00 000 NULL
F41D6: 00 000 NULL
F41D7: 00 000 NULL
F41D8: 00 000 NULL
F41D9: 00 000 NULL
F41DA: 00 000 NULL
F41DB: 00 000 NULL
F41DC: 00 000 NULL
F41DD: 00 000 NULL
```

F400:01C0

```
BIOS DI
INT 016h
IRET
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
ADC DI, CX
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
ADC AX, 000CFh
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
...
```

screen source reset aux vars debug stack flags

edit: E:\Program Files\emu8086\MySource\soal2integerbanyak.asm

file edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator convertor options help About

```
D dw ?
E dw ?

string_input_A    DB 'Masukkan A : $'
string_input_B    DB 13,10,'Masukkan B : $'
string_input_C    DB 13,10,'Masukkan C : $'
string_input_D    DB 13,10,'Masukkan D : $'
string_input_E    DB 13,10,'Masukkan E : $'
string_res        DB 13,10,'Y=$'
sring_undefined   DB 'Undefined$'
; Temporary Variables;
TEMP dw ?
TEMP2 dw ?
ten               dw      10        ; used as multiplier.
four              dw      4         ; used as divider.
make_minus        db      ?         ; used as a flag in procedures.

start:

        mov dx, offset msg0
        mov ah, 9
        int 21h

        ; store A number:
        lea dx, string_input_A
        mov ah, 09h     ; output string at ds:dx
        int 21h

        ; get the multi-digit signed number
        ; from the keyboard, and store
        ; the result in cx register:
        call scan_num
        mov A,cx
```

line: 90 col: 86 drag a file here to open

emulator screen (118x39 chars)

```
Fitur:
+ Bisa menggunakan bilangan bulat positif maupun negatif:
+ Hasil bisa berupa floating number:
Pembatasan:
- Nilai pada var A,B,C,D,E tidak bisa mengandung floating number

Y = (A x B + C) / ( D x E)
Masukkan A: 5
Masukkan B: 2
Masukkan C: 12
Masukkan D: 0
Masukkan E: 1
Y=(5x2+12)/(0x1)

Maka nilai Y =  Undefined
Arigatou Nee OwO ~
```

clear screen change font 0/16

((5(2)+12))/(0(1))

symbolab.com/solver/step-by-step/%5Cfrac%7B%5Cleft(5%5Cl...

Update

Apps AI Software School Book Event Anime Coding Draw Security All Bookmarks

Sy EN Upgrade

$\frac{(5(2)+12)}{0(1)}$ Go

Steps Examples

$\frac{(5(2)+12)}{0(1)}$

Solution

Undefined

Show Steps

Save Noteb Sig Send us Fee

emulator: Calculating.com

file math debug view external virtual devices virtual drive help

Load reload step back waiting for input stop step delay ms: 0

registers

	H	L
AX	00	24
BX	00	00
CX	00	01
DX	01	ED

CS	F400
IP	01C0
SS	0700
SP	FFF8
BP	0000
SI	0000
DI	0000
DS	0700
ES	0700

F400:01C0

```
F41C0: FF 255 RES
F41C1: FF 255 RES
F41C2: CD 205 =
F41C3: 16 022 ▬
F41C4: CF 207 ╧
F41C5: 00 000 NULL
F41C6: 00 000 NULL
F41C7: 00 000 NULL
F41C8: 00 000 NULL
F41C9: 00 000 NULL
F41CA: 00 000 NULL
F41CB: 00 000 NULL
F41CC: 00 000 NULL
F41CD: 00 000 NULL
F41CE: 00 000 NULL
F41CF: 00 000 NULL
F41D0: FF 255 RES
F41D1: FF 255 RES
F41D2: CD 205 =
F41D3: 11 017 ◄
F41D4: CF 207 ╧
F41D5: 00 000 NULL
F41D6: 00 000 NULL
F41D7: 00 000 NULL
F41D8: 00 000 NULL
F41D9: 00 000 NULL
F41DA: 00 000 NULL
F41DB: 00 000 NULL
F41DC: 00 000 NULL
F41DD: 00 000 NULL
```

F400:01C0

```
BIOS DI
INT 016h
IRET
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
ADC DI, CX
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
ADC AX, 000CFh
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
...
```

screen source reset aux vars debug stack flags

edit: E:\Program Files\emu8086\MySource\soal2integerbanyak.asm

file edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator convertor options help About

```
034 D dw ?
035 E dw ?
036
037 string_input_A    DB 'Masukkan A : $'
038 string_input_B    DB 13,10,'Masukkan B : $'
039 string_input_C    DB 13,10,'Masukkan C : $'
040 string_input_D    DB 13,10,'Masukkan D : $'
041 string_input_E    DB 13,10,'Masukkan E : $'
042 string_res        DB 13,10,'Y=$'
043 sring_undefined   DB 'Undefined$'
044 ; Temporary Variables;
045 TEMP dw ?
046 TEMP2 dw ?
047 ten              dw      10      ; used as multiplier.
048 four             dw      4       ; used as divider.
049 make_minus       db      ?       ; used as a flag in procedures.
050
051
052 start:
053
054         mov dx, offset msg0
055         mov ah, 9
056         int 21h
057
058         ; store A number:
059         lea dx, string_input_A
060         mov ah, 09h     ; output string at ds:dx
061         int 21h
062
063         ; get the multi-digit signed number
064         ; from the keyboard, and store
065         ; the result in cx register:
066         call scan_num
067         mov A,cx
```

line: 90 col: 86 drag a file here to open

emulator screen (118x39 chars)

```
Fitur:
+ Bisa menggunakan bilangan bulat positif maupun negatif:
+ Hasil bisa berupa floating number:
Pembatasan:
- Nilai pada var A,B,C,D,E tidak bisa mengandung floating number

Y = (A x B + C) / ( D x E)
Masukkan A: 35
Masukkan B: -201
Masukkan C: -20
Masukkan D: -8
Masukkan E: 4
Y=(35x-201+-20)/(-8x4)

Maka nilai Y =  220.46875
Arigatou Nee OwO ~
```

clear screen change font 0/16

((42(-52)+33))/(5(12))

symbolab.com/solver/step-by-step/%5Cfrac%7B%5Cleft(42%5...

Update

Apps AI Software School Book Event Anime Coding Draw Security All Bookmarks

Sy EN Upgrade

$$\frac{(42(-52)+33)}{5(12)}$$

Solution

$$-35\frac{17}{20}$$

+1

decimal

-35.85

Show Steps

Send us Fee

emulator: Calculating.com

file math debug view external virtual devices virtual drive help

Load reload step back waiting for input stop step delay ms: 0

registers

	H	L
AX	00	24
BX	FF	E0
CX	00	04
DX	01	ED

CS	F400
IP	01C0
SS	0700
SP	FFF8
BP	0000
SI	0000
DI	0000
DS	0700
ES	0700

F400:01C0

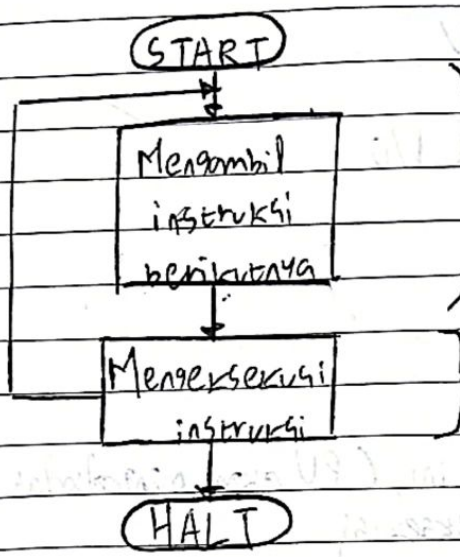
```
F41C0: FF 255 RES
F41C1: FF 255 RES
F41C2: CD 205 =
F41C3: 16 022 ▬
F41C4: CF 207 ±
F41C5: 00 000 NULL
F41C6: 00 000 NULL
F41C7: 00 000 NULL
F41C8: 00 000 NULL
F41C9: 00 000 NULL
F41CA: 00 000 NULL
F41CB: 00 000 NULL
F41CC: 00 000 NULL
F41CD: 00 000 NULL
F41CE: 00 000 NULL
F41CF: 00 000 NULL
F41D0: FF 255 RES
F41D1: FF 255 RES
F41D2: CD 205 =
F41D3: 11 017 ◄
F41D4: CF 207 ±
F41D5: 00 000 NULL
F41D6: 00 000 NULL
F41D7: 00 000 NULL
F41D8: 00 000 NULL
F41D9: 00 000 NULL
F41DA: 00 000 NULL
F41DB: 00 000 NULL
F41DC: 00 000 NULL
F41DD: 00 000 NULL
```

F400:01C0

```
BIOS DI
INT 016h
IRET
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
ADC DI, CX
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
ADC AX, 000CFh
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
...
```

screen source reset aux vars debug stack flags

3. Berikut adalah penjelasan mengenai siklus instruksi dalam CPU



Siklus instruksi adalah proses yang terjadi setiap kali instruksi dieksekusi dalam CPU. Proses ini terdiri dari dua tahap utama, yaitu siklus fetch dan siklus eksekusi. Siklus Fetch adalah tahap pertama dalam siklus instruksi. Pada tahap ini, CPU akan membaca instruksi dari memori. Instruksi ini biasanya disimpan dalam bentuk kode biner yang dapat diinterpretasikan oleh CPU. Ada register khusus dalam CPU yang berfungsi untuk mengawasi dan menghitung instruksi selanjutnya, yang disebut Program Counter (PC). PC akan menambah satu hitungannya setiap kali CPU membaca instruksi. Instruksi yang dibaca akan disimpan dalam register instruksi (IR)

Setelah instruksi di fetch, CPU kemudian akan melakukan aksi yang diperlukan berdasarkan instruksi tersebut. Ada beberapa jenis aksi yang dapat dilakukan oleh CPU, antara lain:

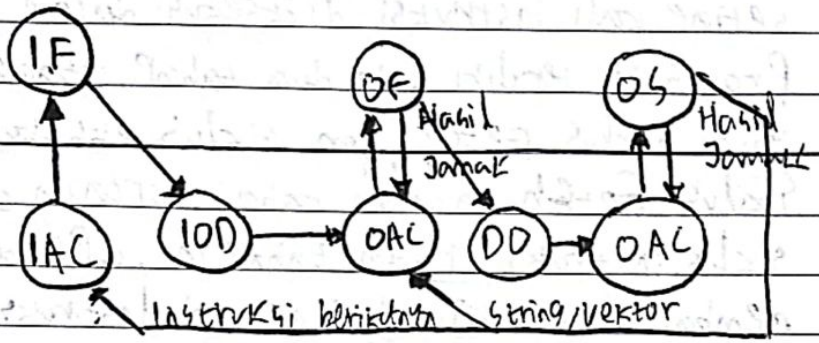
1). CPU Memori : ini adalah perpindahan data dari CPU ke memori dan sebaliknya. Proses ini penting untuk memastikan bahwa data yang diperlukan oleh CPU tersedia saat dibutuhkan

2). CPU I/O : ini adalah perpindahan data dari CPU ke modul I/O dan sebaliknya. Proses ini memungkinkan CPU untuk berinteraksi dengan perangkat lain dalam sistem.

3) Pengolahan Data : CPU melakukan sejumlah operasi aritmatika dan logika terhadap data. Operasi ini dapat mencakup penambahan, pengurangan, perkalian, pembagian, dan operasi logika lainnya.

4) Kontrol : Ini adalah instruksi untuk pengontrolan fungsi atau kerja. Misalnya, instruksi dapat mengubah urutan eksekusi instruksi lainnya

Setelah siklus fetch, berikutnya adalah siklus eksekusi. Siklus eksekusi adalah tahap kedua dalam siklus instruksi dan melibatkan beberapa langkah yaitu:



Akses CPU ke memori atau modul I/O

Operasi Internal

1) Instruction Address Calculation (IAC): Pada tahap ini, CPU akan menentukan atau menentukan alamat instruksi berikutnya yang akan dieksekusi.
2) Instruction Fetch (IF): CPU akan membaca atau mengambil instruksi dari lokasi memori ke CPU.
3) Instruction Operation Decoding (IOD): CPU akan menganalisa instruksi untuk menentukan jenis operasi yang akan dibentuk dan operand yang akan digunakan.
4) Operand Address Calculation (OAC): CPU akan menentukan alamat operand. Hal ini dilakukan apabila melibatkan referensi operand pada memori.
5) Operand Fetch (OF): CPU akan mengambil operand dari memori atau dari modul I/O.
6) Data Operation (DO): CPU akan membentuk operasi yang diperintahkan dalam instruksi.
7) Operand Store (OS): CPU akan menyimpan hasil eksekusi ke dalam memori.

Proses ini berulang-ulang setiap kali instruksi dieksekusi dalam CPU. Dengan demikian, siklus instruksi adalah proses fundamental yang memungkinkan CPU untuk menjalankan program dan melakukan operasi yang diperlukan