Tài liệu ROP-ASM

(Dành cho nX-U8/16 , nếu là VN thì là 580 VNX)

Tập 1

>Những thứ cơ bản:

I.Lời nói đầu :

-ROP, hay còn gọi là lập trình hướng trả về (Return Oriented Program) là hướng lập trình sử dụng chính những hàm có sẵn trên phương tiện lập trình để tạo lập ra một chương trình khác.

-Lấy ví dụ máy tính Casio, nếu ta sử dụng một hàm cập nhật màn hình (print và render) thì ta có thể dùng chính hàm đấy để tạo ra một màn hình in chữ theo ý của mình, không bắt buộc phải chạy theo chương trình mặc định mà máy tính đã lập ra.

-Cách thức thực hiện: Sử dụng các lỗ hổng trên phương tiện lập trình.

II. ROP đối với máy tính Casio

-Là phương tiện giúp người dùng lập trình theo ý muốn mà không chịu tác động từ hướng lập trình mặc định của máy.

-Hay nói cách khác, lập trình ROP trên Casio là việc can thiệp vào chương trình chạy của máy để tạo ra chương trình riêng của bản thân.

III.Ngôn ngữ lập trình trên casio

-Ngôn ngữ lập trình được sử dụng trên nX U8/16 là Assembly (gọi tắt là ASM)

IV.Các lý thuyết cần biết :

1.Thanh ghi (register)là gì ? :

-Thanh ghi là nơi lưu giữ dữ liệu với nhiều mục đích khác nhau.

-Mỗi thanh ghi đều có khả năng lưu trữ khác nhau.

1.1.Các thanh ghi cơ bản đặc biệt và tác dụng :

-Thanh ghi cơ bản :

Loại thanh ghi	Lưu trữ được (bytes)
Rn (R)	1 bytes
Ern (ER)	2 bytes
XRn (XR)	4 bytes
QRn (QR)	8bytes

Với điều kiện : 0<n<16

-Thanh ghi đặc biệt :

Loại thanh ghi	Tác dụng
EA	Tương tự thanh ghi thông thường nhưng chịu ảnh hưởng khi gọi lệnh có chứa "[EA]+" hoặc "[EA]−" và một số lệnh khác
LR	Thanh ghi có tác dụng giữ lệnh quay về hàm chính khi hoàn thành xong hàm phụ (Link register)
PC	Giúp chạy trực tiếp câu lệnh tại địa chỉ ROM được lưu ở chính thanh ghi này (Program Counter)
SP	Thanh ghi gián tiếp chạy câu lệnh trên máy tính

-1 số lệnh phổ biến :

ST	Đưa giá trị từ thanh ghi vào địa chỉ trong RAM.
L	Đưa giá trị từ địa chỉ trong RAM về thanh ghi.
MUL	Nhân giá trị của hai thanh ghi với nhau hoặc thanh ghi với một giá trị cố định.
DIV	Chia giá trị của 2 thanh ghi với nhau hoặc 1 thanh ghi với 1 giá trị cố định.
PUSH	Đẩy giá trị từ thanh ghi thông thường ra địa chỉ trên RAM được lưu ở thanh ghi SP.

	Khi đó SP sẽ được trừ lên tương ứng với số byte của thanh ghi đứng sau POP.
ADD	Cộng hai giá trị của thanh ghi với 1 giá trị cố định.
SUB	Trừ 2 giá trị của thanh ghi với 1 giá trị cố định.
CMP	So sánh giá trị 2 thanh ghi với 1 giá trị cố định.
POP	Đưa giá trị từ địa chỉ trên RAM được lưu ở thanh ghi SP vào thanh ghi thông thường. Khi đó SP sẽ được cộng lên tương ứng với số byte của thanh ghi đứng sau POP.
MOV	Đưa giá trị từ thanh ghi này sang thanh ghi khác.

>Hướng dẫn ROP :

I.In chữ 1 đến 4 line cơ bản (printline) :

-Trong phần này bạn sẽ làm quen với lệnh mới và nhiều thứ nữa!

org 0xe9e0 (#địa chỉ của programs vd e9e0,d730)

setlr (#set link register)

setsfr (#reset sfr)

buffer_clear (#xóa tất cả vùng hiển thị , hay là màn hình casio)

print: (#trong comp cái này giúp dễ nhận bt đc đag lm j ko phải lệnh :))

xr0 = hex 0a 01 , adr_of text (# ở đây bỏ 0a(font chữ) 01 (vị trí/line pos) vào gadget xr0 , sau đó là adr_of (nói đơn giản là gọi tên cho lệnh đó để giúp programs gắn gọn-tôi ko bt giải thích ntn nữa , text ở đây là label/tên đặt j cũng đc nhưng ở cuối phải gọi đúng y tên mà bạn đặt)

smallprint (# ở đây là lệnh in chữ)

render.ddd4 (#nói đơn giản là hiện lên màn hình)
loop: (# vòng lặp , lặp lại mọi thứ từ đầu programs)
xr0=0xd184d630 (#cái ni tôi ko bt nói)
BL strcpy (#cái ni tôi ko bt nói)
er6=0xd62e (#cái ni tôi ko bt nói)
sp=er6,pop er8 (#cái ni tôi ko bt nói)
text: (# gọi lại label để bỏ text)
str "~Text~here" (# str là string(tự tìm) sau đó thì " và bỏ text vào/ko bỏ đc tiếng việt đâu, à mà nhớ kết thúc bằng đóng ngoặc nhé"
0x00 (# nói đơn giản là ngăn cách nó vs text hoặc gì khác)

•Note : Trong ROP đấu ~ là dấu cách , 0x như hex ,nhưng hex cần cách và viết thẳng còn 0x phải đảo lại và viết liền vd hex 04 02 = 0x0204

Line pos/vị trí chữ :
01 = line 1
11 = line 2 21
= line 3
31 = line 4
--vẫn có thể dùng đc line pos khác nhưng phải từ 01->31, tuy nhiêu sẽ bị cắt chữ !
1.2 in chữ 4 line :

```
org 0xe9e0
setlr
setsfr
buffer_clea
r print:
xr0 = hex 01 30, adr_of text
printline
xr0 = hex 11 30, adr_of text2 ( #in line 2 )
printline (# in line 2 )
```

```
xr0 = hex 21 30, adr_of text3 ( #in line 3 )
printline ( #in line 3 )
xr0 = hex 31 30, adr_of text4 ( #in line 4
)
printline ( #in line 4 )
render.ddd4
text:
str"1"
0x00
text2:
str"2"
0x00
text3:
str"3"
0x00
text4:
str"4"
0x00
```

-2,3 line tự suy ra ! (bằng cách như vậy ta suy ra 8 line !)

II.In chữ 8 line (smallprint):

-smallprint cũng in chữ nhi printline chỉ khác là nóc chỉnh đc font còn printlinr thì ko :

1. 1.In chữ 1 line font

```
0a org 0xe9e0 setlr
setsfr buffer_clear
print:
xr0 = 0x010a, adr_of font_0a
smallprint
render.ddd4
font_0a:
hex 30 61
00
```

-ở đây tôi dùng 0x,hex để thấy đc nó khác nhau !

-tương tự 4 line ta cần x8 lên là đc :

Font cho 08 : (với 0a thì cái này dùng line như 0e cx đc)

Line :	Hex
1	01
2	09
3	11
4	19
5	21
6	29
7	31
8	39

1.2. in chữ 8 line font 08

```
org 0xe9e0
screen:
setlr
setsfr
buffer_clear
xr0 = 0x0001d113
[er0] = r2
xr0 = 0x0015f031
[er0] = r2 spell:
  xr0 = 0x0108, adr_of text1
smallprint
  xr0 = 0x0908, adr_of text2
smallprint
  xr0 = 0x1108, adr_of text3
smallprint
  xr0 = 0x1908, adr_of text4
smallprint
  xr0 = 0x2108, adr_of text5
  smallprint
  xr0 = 0x2908, adr_of text6
smallprint
```

```
  xr0 = 0x3108, adr_of text7
smallprint
  xr0 = 0x3908, adr_of
text8
smallprint
render.ddd4
text1:
  str"1111111111111111111111111111111"
text2:
  str"2222222222222222222222222222222"
text3:
  str"3333333333333333333333333333333"
text4:
  str"4444444444444444444444444444444"
text5:
  str"5555555555555555555555555555555"
text6:
  str"6666666666666666666666666666666"
text7:
  str"7777777777777777777777777777777"
text8:
  str"8888888888888888888888888888888"
```

-Trong phần ASM này có 1 đoạn :

```
xr0 = 0x0001d113
[er0] = r2 xr0 =
0x0015f031
  [er0] = r2
```

-Thế nó là cái gì ?

-đó là cursor no flash (ẩn con trỏ)!

III.Hình thành kiến thức mới !:

1.1.waitshift/waishift_blank:

-nói đơn giản là chờ người dùng bấm phấm shift mới tiếp tục thực hiện lệnh !

Và nó đc dùng như sau :

-bỏ nó trước các lệnh mà bạn muốn phải ấn shift!

Vd:ấn shift hiện chữ :

```
org 0xe9e0
home:
setlr
setsfr
buffer_clear
xr0 =
0x0001d113
[er0] = r2
xr0 =
0x0015f031
[er0] = r2
print:
  xr0 = 0x110e, adr_of
text_1
waitshift
render.ddd4
text_1:
  hex 31 31 31 31 31
```

1.2.Invert (bôi đen)

-đây là lệnh giúp bôi đen 1 thứ gì đó mà bạn muốn ! Vd cách dùng : xr0 = 0x0004d138 call 203d2

-trong phần code này có call 203d2 ?

Call cũng giống như lệnh vậy

Cách dùng call + hex của lệnh đó !

1.3.tắt invert

-sau khi bạn invert mà muốn tắt thì đây là hướng dẫn!

Vd cách tắt:

```
setlr
```

buffer_clear

xr0 = 0xd138, 0x01, 0x30

[er0] = r2

1.4.delay

-như tên tức làm chậm lệnh kế tiếp

Vd cách dùng :

er0 = 0x0000

delay

-0000 là hex mà bạn muốn delay bao lâu

1.5.scroll

-nó làm cho cuộn xuống/lên cả màn hình !

Vd cách dùng :

```
er2 = 0x0101 (#0101 là lên , ffff là xuống  )
er8 = 0xf039
call 09ca0
0x30303030
```

1.6.line_draw

-như tên nó đc dung để vẽ 1 đường thẳng ngan/dọc/nghiêng

Vd các dùng:

```
xr0 = hex x1 y1 x2 y2
BL line_draw
```

-ta có :

+ x1 = vị trí ngang bắt đầu
+ y1 = vị trí dọc bắt đầu
+ x2 = vị trí ngang kết thúc
+ y2 = vị trí dọc kết thúc

-Tại sao tôi lại dung BL line_draw thay vì là line_draw ?

-ở đây BL được dùng khi có nhiều lệnh (chỉ áp dụng vs 1 số như line_draw,printline,smallprint)

-Áp dụng từ 1 lệnh trở lên

1.7.render_bitmap

-render_bitmap là lệnh cho phép sử dụng đc hình ảnh hay bitmap (nhưng ko đc quá giới hạn màn hình casio nhé)

Vd cách dung

```
setlr
setsfr
buffer_clear
bitmap:
xr0 = hex x1 y1 size-x size-y
render_bitmap
er0 = adr_of bitmap
render.ddd4
```

```
bitmap:
  hex 3c 7e ff ff ff ff 7e 3c
# x1,y1 như đã nói vị trí theo x và y
-size x : là kích cỡ của bitmap ngang
-size y : là kích cỡ của bitmap dọc
➔cách bt đc size là dec>hex trong menu 3 casio
# ở phần hex bitmap hãy dịch tranh sang hex (bằng
cách dung hmtl,và nếu ko đủ bitmap theo hex thì
bitmap sẽ lỗi hình !
```

1.8.line_print__call__

```
-chỉ đơn giản là in chữ nhưng có thể căn lề
Vd cách dùng :
  xr0 = adr_of txt, 0xYY, 0x00
  call 2edca
  render.ddd4
  brk
txt:
  str "Helllo"
  0x00
-YY là vị trí
-00 = r3 :
Phần R3:
+ R3 = 0 => in chữ căn lề phải
+ R3 ≠ 0 => in chữ căn lề giữa
```

1.9.Cách dùng calc_func_verify

```
-như tên cái này giúp xác minh
Vd cách dùng :
xr0 = adr_of adrcalc , adr_of output
calc_func_verify ( thật ra calc_func cx đc )
# call 13344 btw
…
adrcalc:
adr_of calc
calc:
[Phép tính 1][dấu <>≠≤≥= lần lượt là
FB01/FB02/FB03/FB04/FB05/A5][Phép tính 2]
[0x00]
output:
<trả về num 1 nếu Phép tính 1 hợp dấu với phép
tính 2, không hợp trả về num 0. Chi tiết vào MENU
B, thử 3=3 và xem RAM chỗ D16C
Calc_func_verify example
Ví dụ: so sánh x+1 > y
org 0xe9e0
xr0 = adr_of adrcalc , 0xd16c
calc_verify
call 33030
adrcalc:
adr_of calc
calc:
hex 48 a6 31 fb 02 49 00
```

→ Sau khi chạy calc_verify, thì địa chỉ D16C sẽ chứa num 1 nếu thỏa mãn x+1 > y, num 0 nếu không thỏa mãn.
Chú ý: vì output dạng num nên phải để vùng output khoảng 10 hex (14 hex nếu là CWII).
Fact : Cái này dùng để so phép tính đúng/sai cũng đc !

2.0. input_func

-Dùng để lấy kí tự nhập từ bàn phím, và thực hiện phép tính, kết quả lưu tại vị trí gọi input func - 32 (dec)
Vd cách dùng :

```
xr0 = 0xd114, 0x1a, 0x30
[er0]=r2 # Cái này để cho người dùng khi làm syntax ERROR, vị trí sẽ không thay đổi.
xr0 = 0xd10e, 0x42, 0x30
[er0]=r2 # Nếu ko có cái này thì con trỏ sẽ ko hoạt động
call 2f210 #input_func
```

Nhược điểm : nhập đc tối đa 100 num/số

2.1.Ea_switchcase hay getscankeycode

-nói đơn giản nó giúp có các key để thực hiện lệnh cho từng key riêng biệt

```
check_key:
   er0 = adr_of key
   setlr
   getkey
   ea = adr_of key_table
   pop er0
key:
   hex 00 00
   call 09c20
   call 1c64a
   setlr
   sp = er6,pop er8
 … sau đó là các chức năng của từng key !
key_table:
   hex ( 4 số tượng cho số tương ứng vs key )
   adr_of [-2] label chức năng key mà bạn chọn cho nó làm
```

ASM của mouse by me (byrio)

```
org 0xd730
home:
   setlr
   setsfr
   buffer_clear
xr0 = 0x0001d113
   [er0] = r2
xr0 = 0x0015f031
   [er0] = r2
mouse:
   xr0 = hex 58 17 10 10
```

```
  render_bitmap
  er0 = adr_of @
  render.ddd4
check_key:
  er0 = adr_of key
  getkey
  ea = adr_of key_table
  pop er0
key:
  hex 00 00
  call 09c20
  call 1c64a
  sp = er6,pop er8
1_up:
  er2 = 0xff00
  er8 = adr_of [+4788] mouse
  call 09ca0
  0x30303030
  goto loop
2_down:
  er2 = 0x0100
  er8 = adr_of [+4788] mouse
  call 09ca0
  0x30303030
  goto loop
3_right:
  er2 = 0x0001
  er8 = adr_of [+4788] mouse
  call 09ca0
  0x30303030
  goto loop
4_left:
  er2 = 0xffff
  er8 = adr_of [+4788] mouse
  call 09ca0
  0x30303030
  goto loop
invert:
  xr0 = 0x0004d138
  call 203d2
  goto loop
off_invert:
  setlr
  buffer_clear
  xr0 = 0xd138, 0x01, 0x30
  [er0] = r2
loop:
  xr0 = 0xd184d630
  BL strcpy
  er14 = 0xd62e
  sp = er14,pop er14
key_table:
  0x3030303039303030303030
  0x0201
  adr_of [-2] off_invert
```

```
   0x0101
   adr_of [-2] invert
   0x0480
   adr_of [-2] 1_up
   0x0840
   adr_of [-2] 2_down
   0x0880
   adr_of [-2] 3_right
   0x0440
   adr_of [-2] 4_left
   0x0000
   adr_of [-2] loop
@:
  hex 0C 00 0A 00 09 00 08 80 08 40 08 20 08 10 08
08 08 04 08 3C 0B 20 0D 20 00 90 00 90 00 60 00 00
```

Tập 1 chỉ đến đây thôi ! Hẹn mn vào tập 2

Trách nhiệm về nội dung
@byriooo (Nhật Huy)
Biên soạn/tổng hợp nội dung
@byriooo (Nhật Huy)
Cách nội dung trong đây được tham khảo từ nhiều nguồn ROP

=== Nếu có sai sót hãy liên hệ với tôi ===
Và cảm ơn một số người đã hỗ trợ tôi !
Hi vọng tài liệu này sẽ giống bạn hiểu thêm về ROP-ASM trên casio 580!

Trong thời gian tôi làm tập 2 :v
Nhớ đọc thêm ASM của những người giỏi ROP để học thêm!

Hết!