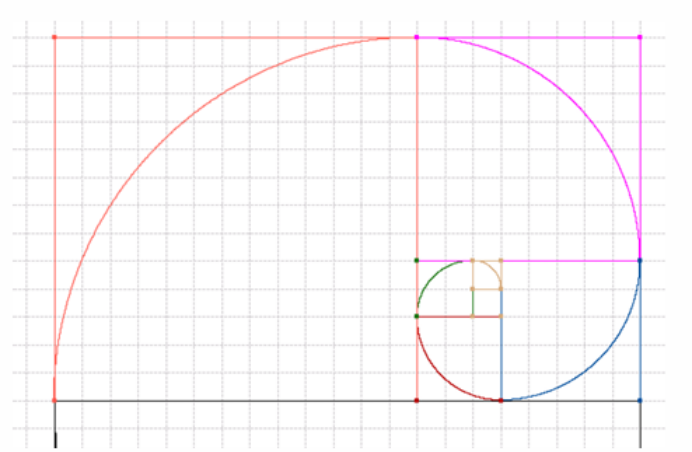
24H du Code 2026

STMicroelectronics – Mission Nexus Prime

Epreuve 0 – Entrainement



Dump Ram - backend	BalkisJerad <capitainji@gmail.com>
Merge branch 'Epreuve-5'	POUDEROUX Tom <tom.pouderoux@
remotes/origin/Epreuve-5 Epreuve 5 mk2	POUDEROUX Tom <tom.pouderoux@
C++	POUDEROUX Tom <tom.pouderoux@
Avancement Epreuve 5	POUDEROUX Tom <tom.pouderoux@
ajout affichage ram et out	Jack Parrot <aurelien.samson@sii.fr>
Ajout .bin comme param à simulateur_front	BalkisJerad <capitainji@gmail.com>
ajout fonctionnalité affichage stack	Jack Parrot <aurelien.samson@sii.fr>
Merge branch 'master' of https://git.serveurtom.fr/Tom/24H_du_code_202	Jack Parrot <aurelien.samson@sii.fr>
MAJ gitlab	POUDEROUX Tom <tom.pouderoux@
Epreuve2 Version 2 (Gestion des DB)	BalkisJerad <capitainji@gmail.com>
Gestion DT	BalkisJerad <capitainji@gmail.com>
Ajout fichiers	POUDEROUX Tom <tom.pouderoux@
Assembleur final	POUDEROUX Tom <tom.pouderoux@
return des valeurs à récupérer pour affichage dynamique	Jack Parrot <aurelien.samson@sii.fr>
Merge branch 'master' of https://git.serveurtom.fr/Tom/24H_du_code_202	Jack Parrot <aurelien.samson@sii.fr>
Epreuve 0 - rev 1	POUDEROUX Tom <tom.pouderoux@
Epreuve 3	BalkisJerad <capitainji@gmail.com>
Création interface et éléments	Jack Parrot <aurelien.samson@sii.fr>
Dictionnaire	POUDEROUX Tom <tom.pouderoux@

```
_main:
    MOV R0 0    ; a
    MOV R1 1    ; b
    OUT R0      ; print 0

_loop:
    OUT R1      ; print b
    MOV R2 0    ; 0
    MOV R3 R0   ; c = a
    SUB R2 R1   ; 0 - b
    SUB R3 R2   ; a - (0 - b) = a - -b = a + b

    MOV R0 R1   ; a = b
    MOV R1 R3   ; b = c

    CMP R1 R0
    JLT _end    ; end si b < a
    JMP _loop

_end:
    RET
```

Epreuve 1 – Le chronocode

Epreuve 2 – Le programme d'assemblage

Gestion propre des DB

Gestion des tabulations des des multi-espaces

Controle de la validité et la cohérence des paramètres

```
_main:
    MOV R0 0    ; a
    MOV R1 1    ; b
    OUT R0      ; print 0

_data1:
    DB 0         ;00000000
    DB 'C'       ;01000011

_loop:
    OUT R1      ; print b
    MOV R2 0    ; 0
    MOV R3 R0   ; c = a
    SUB R2 R1   ; 0 - b
    SUB R3 R2   ; a - (0 - b) = a - -b = a + b

    MOV R0 R1   ; a = b
    MOV R1 R3   ; b = c

    CMP R1 R0
    JLT _end    ; end si b < a
    JMP _loop

_end:
    LDR R2 R3 _data1
    RET
```

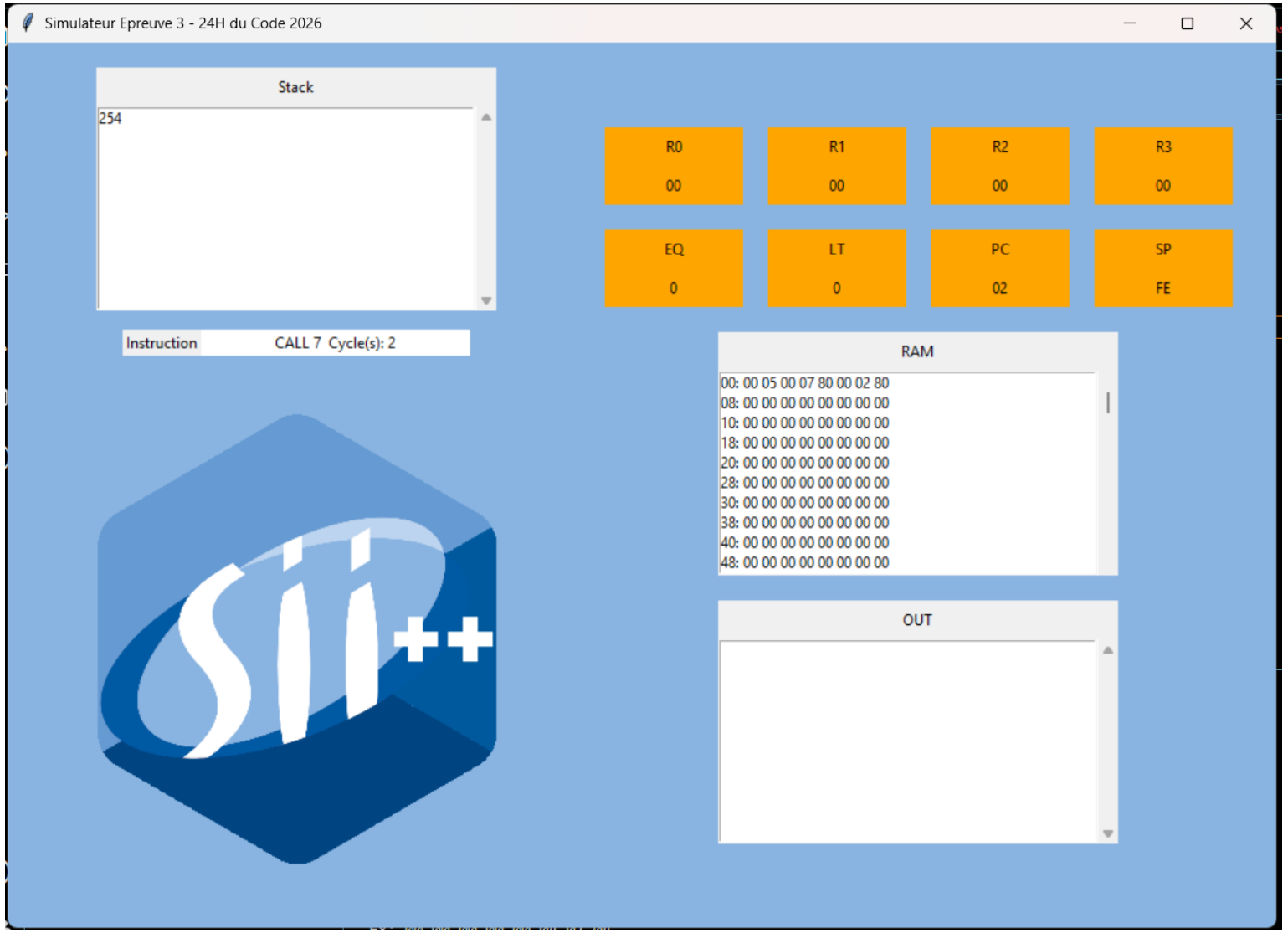
```
================================================
 ASSEMBLY PREVIEW: .\Epreuve2.asm
================================================
ADDR    | HEX   | BINARY
-----------------------------
0x00    | e0    | 11100000  (_main)
0x01    | 00    | 00000000
0x02    | e1    | 11100001
0x03    | 01    | 00000001
0x04    | f0    | 11110000
0x05    | f1    | 11110001  (_loop)
0x06    | e2    | 11100010
0x07    | 00    | 00000000
0x08    | 5c    | 01011100
0x09    | d9    | 11011001
0x0A    | de    | 11011110
0x0B    | 51    | 01010001
0x0C    | 57    | 01010111
0x0D    | 34    | 00110100
0x0E    | c0    | 11000000
0x0F    | 12    | 00010010
0x10    | 40    | 01000000
0x11    | 05    | 00000101
0x12    | bb    | 10111011  (_end)
0x13    | 15    | 00010101
0x14    | 80    | 10000000
0x15    | 00    | 00000000  (_data1)
0x16    | 43    | 01000011
================================================
```

```
          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded Text
00000000  E0 00 E1 01 F0 F1 E2 00 5C D9 DE 51 57 34 C0 12  . . . . . . . . \ . . Q W 4 . .
00000010  40 05 BB 15 80 00 43                             @ . . . . . C
```

Epreuve 3 – Programmation d'un simulateur

Interface Graphique
du Simulateur Visualisant
pour chaque instruction :

INS	RGs	Stack	RAM
Cycle	PC	SP	EQ
	LT	OUT	



Epreuve 5 – Envoyer le robot à sa zone d'activation

- Mission Nexus Prime accomplie avec Succes

- Allumer les LEDs du Robot en fonction des Registres contenant la suite de Fibonacci

- Afficher le Logo de l'équipe sur l'écran OLED

Perspectives

- Ecrire un ASM émulant un programme 16 Bits s'executant sur une architecture 8 Bits

Fibbo16b.asm

```
_main:
    MOV R0 1   ; b
    SUB R1 R1  ; b
    SUB R2 R2  ; a
    SUB R3 R3  ; a

_loop:
    OUT R2
    OUT R3

    PUSH R0  ; Sauvegarde de b
    PUSH R1

    PUSH R1    ; R0 R1 => b   R2 R3 => a. Retourne b = b + a = c
    SUB R1 R1
    SUB R1 R2
    SUB R0 R1
    POP R1      ; R0 = R0 + R2

    CMP R0 R2   ; Si overflow, il faut +1 R3 (a oct fort)
    JLT _add16cr1debut
    JMP _add16cr1fin
_add16cr1debut:
    SUB R3 255
_add16cr1fin:

    PUSH R0
    SUB R0 R0
    SUB R0 R3
    SUB R1 R0
    POP R0      ; R1 = R1 + R3
    CMP R1 R3
    JLT _add16cr2debut
    JMP _add16cr2fin
_add16cr2debut:
    JMP _end
_add16cr2fin:

    POP R3  ; Reprise de b
    POP R2
    JMP _loop

_end:
    POP R3
    POP R2
    RET
```