

ການພັນວັນ F

ຢູ່ມືອງກີບໄກ / ໄກສອນ

ການທດລອງທີ 6 ການພັນນາໂປຣແກຣມພາສາແອສເຊັ່ນ

ການທດລອງນີ້ຄາດວ່າຜູ້ອ່ານຜ່ານການເຂື້ອນຫຼືອພັນນາໂປຣແກຣມດ້ວຍພາສາ C ໃນການທດລອງທີ 5 ການພັນວັນ E ແລ້ວ ແລ້ວມີຄວາມຄຸ້ນເຄຍກັບ IDE ຈາກການພັນນາໂປຣແກຣມແລກວິທີການດ້ວຍພາສາ C/C++ ດ້ວຍ Code-Blocks ດັ່ງນັ້ນ ການທດລອງມີວັດຖຸປະສົງສົດເລື່ອນີ້

- ເພື່ອໃຫ້ເຂົ້າໃຈການພັນນາແລກວິທີການ ໂປຣແກຣມພາສາແອສເຊັ່ນ ຕໍ່ IDE ຈຶ່ງ CodeBlocks ບໍ່
ຮະບບປະເປົດຕິການ Raspbian/Linux/Unix
- ເພື່ອໃຫ້ເຂົ້າໃຈຄວາມແຕກຕ່າງຮ່ວງການພັນນາໂປຣແກຣມພາສາແອສເຊັ່ນ ຕໍ່ IDE ແລ້ວ Makefile

F.1 ການພັນນາໂປຣແກຣມພາສາແອສເຊັ່ນ

1. ພິມພົບຄໍາສັ່ນນີ້ໃນໂປຣແກຣມ Terminal ເພື່ອເຮັດວຽກ CodeBlocks

\$ codeblocks

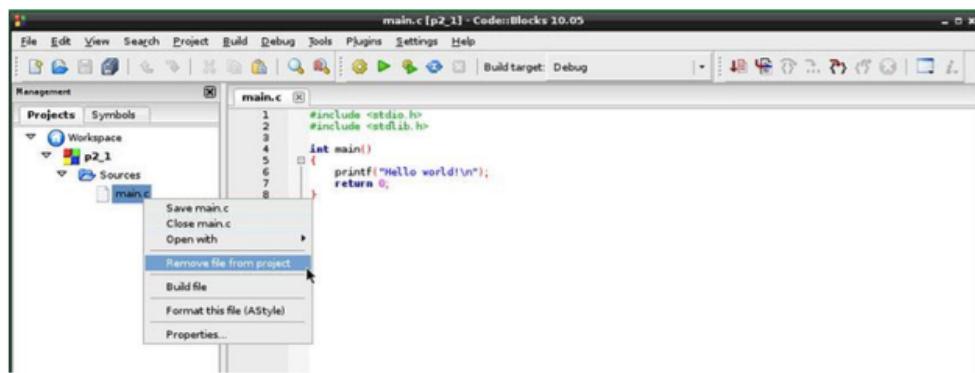
2. ໜ້າຕ່າງໆ ລັກຈະປະກຸບບັນຫຼາຍ ຮັບໃຈການພັນນາ ຜູ້ອ່ານສາມາດຮັບໄດ້ການສ້າງໂປຣແກຣມໃໝ່ ເພື່ອເລືອກ "Create a new project" ໃນຊ່ອງດ້ານໜ້າຍ ແລ້ວເລືອກ "Console application" ໃນຊ່ອງດ້ານຂວາເພື່ອສ້າງໂປຣແກຣມ

3. ກຣອກຊື່ໂປຣແກຣມໃໝ່ Lab6 ໃນຊ່ອງ Project title: ແລກກຣອກຊື່ໄດ້ເຮັດວຽກ /home/pi/asm/ ໃນ
ຊ່ອງ Folder to create project in: ໂປຣສັງເກດຂໍ້ອຄວາມໃນຊ່ອງ Project filename: ວ່າຕຽນກັບ
Lab6.cbp ໃຊ່່ໂໝ່ ແລ້ວຈຶ່ງກັດ Next>

4. ໂປຣແກຣມ CodeBlocks ຈະສ້າງໄດ້ເຮັດວຽກຕ່າງໆ ກາຍໃຫ້ໄດ້ເຮັດວຽກໃໝ່ /home/pi/asm/Lab6/

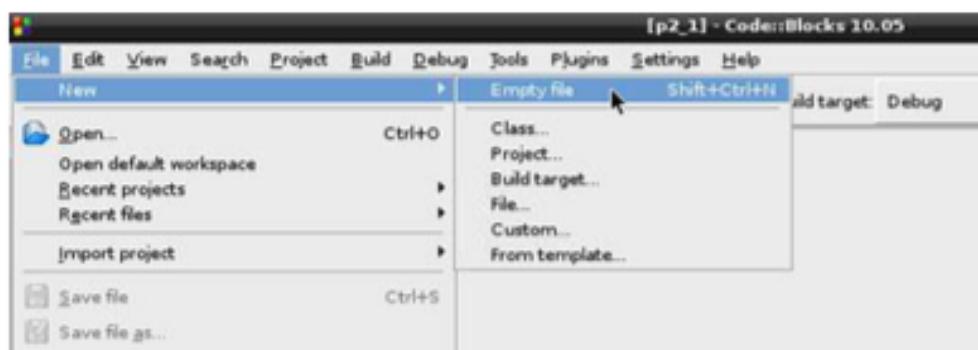
5. ກົດປຸ່ມ "Next>" ເພື່ອດຳເນີນການຕ່ອແສ ສຸດທ້າຍຈະເປັນບັນດາການເລືອກຄອນພິກຸງເຮັດວຽກ (Configuration) ສໍາຫັບຄອມໄຟເລວ່ອ ເລືອກອອພ້ານ Debug ແນະສໍາຫັບການເຮັດວຽກ ແລ້ວຈຶ່ງ
ກົດປຸ່ມ "Finish" ເມື່ອເສົ້າງສິນ

6. ຄລິກຊື່Workspace ໃນໜ້າຕ່າງໆ ເພື່ອຂໍ້ອຍໄຍໂຄຮສ້າງໂປຣແກຣມໃໝ່ ແລ້ວກັນໄຟໄລ໌ຊື່ "main.c"
ຄລິກຂວາບນ້ຳໄຟລ໌ ແລ້ວເລືອກເມີນ "Remove file from project" ຕາມຮູບທີ F.1



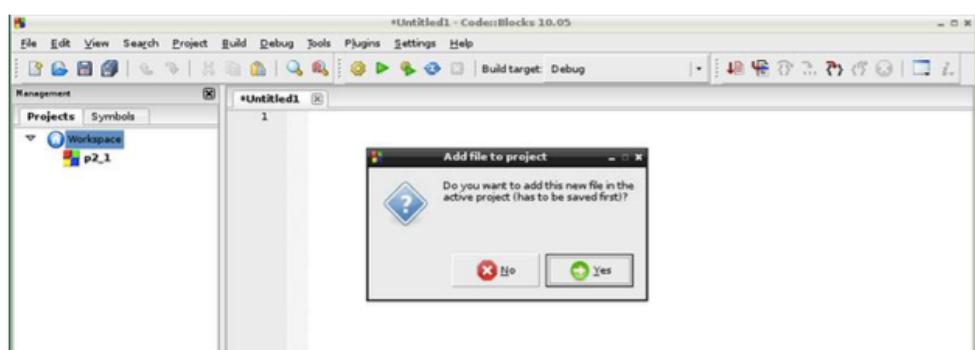
รูปที่ F.1: การย้ายไฟล์ main.c ออกจากโปรเจ็ค

7. เพิ่มไฟล์ใหม่ลงในโปรเจ็คโดยกดเมนู File->New->Empty file ตามรูปที่ F.2



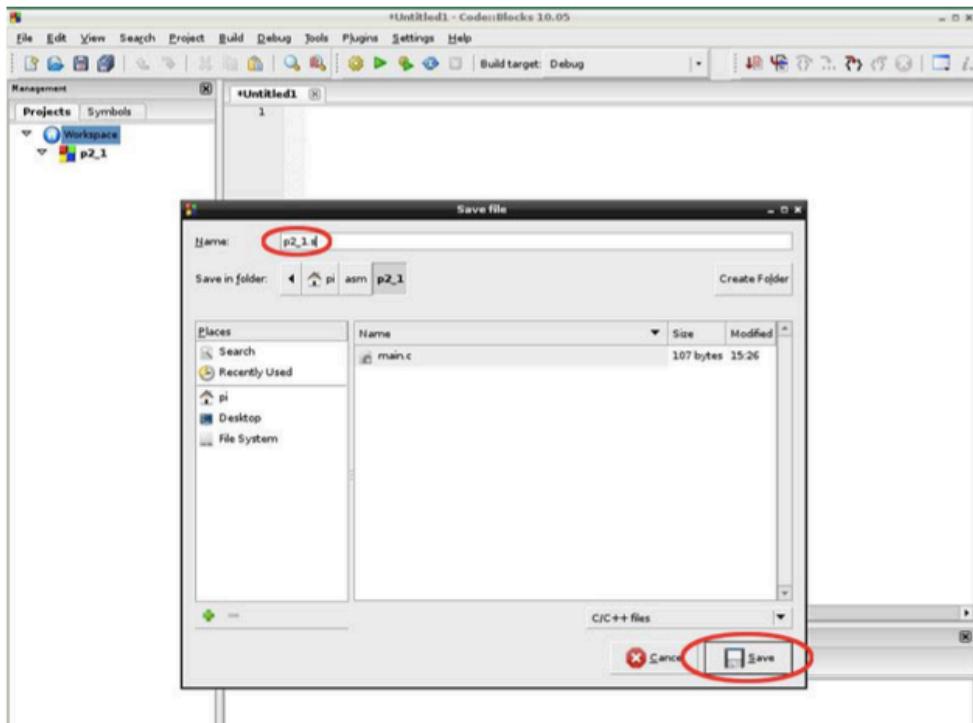
รูปที่ F.2: การเพิ่มไฟล์ใหม่ลงในโปรเจ็ค

8. คลิกปุ่ม ”Yes” เพื่อยืนยันในรูปที่ F.3



รูปที่ F.3: หน้าต่างกดปุ่ม ”Yes” เพื่อยืนยัน

9. หน้าต่าง ”Save file” จะปรากฏขึ้น กรอกชื่อไฟล์ว่า main.s และวิจิ้งกดปุ่ม ”Save” ดังรูปที่ F.4



รูปที่ F.4: หน้าต่าง Save File ชื่อไฟล์ว่า main.s

10. คลิกชื่อ Workspace ในหน้าต่างด้านซ้าย และคลิกขวาเพิ่ม (Add) ไฟล์ main.s เข้าไปในโปรเจ็คท์

11. ป้อนคำสั่งเหล่านี้ในไฟล์ main.s

```
.global main
main:
    MOV R0, #0
    MOV R1, #2
    MOV R2, #4
    ORR R0, R1, R2
    BX LR
```

$$\begin{array}{r} R_1 \quad 0\ 0\ 1\ 0 \\ R_2 \quad 0\ 1\ 0\ 0 \\ \hline R_0 \quad 0\ 1\ 1\ 0 \end{array}$$

$$R_0 = R_1 \mid R_2 \quad (\text{Bitwise OR})$$

$$\begin{array}{r} R_1 \quad 0\ 0\ 1\ 0 \quad (ORR) \\ R_2 \quad 0\ 1\ 0\ 0 \\ \hline R_0 \quad 0\ 1\ 1\ 0 \end{array}$$

12. เลือกเมนู Build->Build เพื่อแปล (Assemble) โปรแกรมที่เขียนให้เป็นโปรแกรมภาษาเครื่อง

13. เลือกเมนู Build->Run เพื่อรันโปรแกรม

14. อ่านและบันทึกประโยชน์ที่เกิดขึ้นในหน้าต่าง Terminal ที่ปรากฏขึ้นมา

$$R_0 = b$$

ทบทวนหน้าต่าง .

Mov wasn't Move

Can we move our Register

The screenshot shows the VisUAL debugger interface. The assembly code window displays:

```
1 main      MOV      R0, #0
2          MOV      R1, #2
3          MOV      R2, #4
4          ORR      R0, R1, R2
```

The status bar indicates "Emulation Complete" with 4 issues and 0 errors. The register window on the right shows the following values:

Register	Value	Dec	Bin	Hex
R0	0x6	Dec	Bin	Hex
R1	0x2	Dec	Bin	Hex
R2	0x4	Dec	Bin	Hex
R3	0x0	Dec	Bin	Hex
R4	0x0	Dec	Bin	Hex
R5	0x0	Dec	Bin	Hex
R6	0x0	Dec	Bin	Hex
R7	0x0	Dec	Bin	Hex
R8	0x0	Dec	Bin	Hex
R9	0x0	Dec	Bin	Hex
R10	0x0	Dec	Bin	Hex
R11	0x0	Dec	Bin	Hex
R12	0x0	Dec	Bin	Hex
R13	0xFF000000	Dec	Bin	Hex
LR	0x0	Dec	Bin	Hex
PC	0x14	Dec	Bin	Hex

The status bar also shows "Clock Cycles" and "Current Instruction: 1 Total: 4".

F.2 การดีบักโปรแกรมโดยใช้ IDE

1. ในไฟล์ main.s เลื่อนเมาส์ไปบรรทัดที่มีคำสั่ง ORR R0, R1, R2 คลิกเมนู Debug->breakpoint หรือกดปุ่ม F5 ผู้อ่านจะสังเกตวงกลมสีแดงปรากฏขึ้นด้านซ้ายของคำสั่ง ORR
 2. กดเมนู Debug->Debugging Windows->CPU Registers เพื่อแสดงค่าของ CPU register ในหน้าต่างที่ปรากฏขึ้นมาเพิ่มเติม
 3. เมื่อพร้อมแล้ว ผู้อ่านสามารถเริ่มต้นการดีบักโดยกดเมนู Debug->Start/Continue หรือกดปุ่ม F8 โปรแกรมจะเริ่มต้นทำงานตั้งแต่ประযุกแครกจนหยุดที่คำสั่ง ORR R0, R1, R2
 4. อ่านและบันทึกค่าของ R0 และ PC ในหน้าต่าง CPU Registers
 5. ประมาณผลคำสั่งถัดไปโดยกดเมนู Debug->Next Instruction หรือกดปุ่ม Alt+F7 พร้อมกัน
 6. อ่านและบันทึกค่าของ R0 และ PC ในหน้าต่าง CPU Registers และสังเกตการเปลี่ยนแปลงที่เกิดขึ้นโดยเปรียบเทียบกับค่า R0 และ PC ในข้อ 4 กับข้อนี้
 7. อธิบายว่าเกิดอะไรขึ้นกับค่าของรีจิสเตอร์ R0 และ PC
- R0 ลง 1 หนึ่ง步 0 → 6 #

F.3 การพัฒนาโดยใช้ประยุกคำสั่งที่ลับขั้นตอน

ผู้อ่านควรเข้าใจคำสั่งพื้นฐานในการแปลงโปรแกรมภาษาแอสเซมบลีที่สร้างขึ้นใน CodeBlocks ก่อนหน้านี้ตามขั้นตอนต่อไปนี้

1. ใช้โปรแกรมไฟล์เมเนจरเพื่อเบรนไฟล์ในไดเรกทอรี /home/pi/asm/Lab6
 2. ดับเบลคลิกบนชื่อไฟล์ main.s เพื่อเปิดอ่านไฟล์และเปรียบเทียบกับไฟล์ที่เขียนในโปรแกรม CodeBlocks
 3. เปิดโปรแกรม Terminal หน้าต่างใหม่แล้วป้อนไดเรกทอรีปัจจุบัน (cd: change directory) ไปยัง /home/pi/asm/Lab6 โดยใช้คำสั่ง
- ```
$ cd /home/pi/asm/Lab6
```

4. แปลไฟล์ขอร์สโค๊ดให้เป็นไฟล์อ๊อบเจ็คท์ โดยเรียกใช้คำสั่ง as (assembler) ดังนี้

```
$ as -o main.o main.s
```

5. ใช้คำสั่ง ls -la ใน Terminal เพื่อค้นหาไฟล์อ๊อบเจ็คท์ชื่อ main.o ว่ามีจริงหรือไม่
6. ทำการลิงค์และแปลงไฟล์อ๊อบเจ็คท์เป็นไฟล์โปรแกรมโดย

\$ gcc -o Lab6 main.o

*compile -*

*link.*

as -o main.o main.s

gcc -o Lab6 main.o

7. ใช้คำสั่ง ls -la ใน Terminal เพื่อค้นหาไฟล์โปรแกรมชื่อ Lab6 ว่ามีจริงหรือไม่

8. เรียกโปรแกรม Lab6 โดยพิมพ์

\$ ./Lab6

%\$ echo \$?

น่าจะดีกว่านี้

9. เปรียบเทียบหมายเลขที่ปรากฏขึ้นว่าตรงกับผลการรันใน IDE หรือไม่ ออย่างไร

## F.4 การพัฒนาโดยใช้ Makefile

การใช้ **makefile** สำหรับพัฒนาโปรแกรมภาษาแอสเซมบลีคล้ายกับการทดลองที่ 5 ในภาคผนวก E ก่อนหน้านี้

1. เปิดไดเรกทอรี /home/pi/asm/Lab6 ด้วยโปรแกรมไฟล์เมเนจero

2. กดปุ่มขวาบนมาสีในพื้นที่ไดเรกทอรีเพื่อสร้างไฟล์เปล่าใหม่ (New Empty File) โดยกำหนดชื่อ **makefile**

3. ป้อนข้อความเหล่านี้ลงในไฟล์ makefile:

```
Lab6: main.o
 gcc -o Lab6 main.o
main.o: main.s
 as -o main.o main.s
clean:
 rm *.o Lab6
```

*Link.*

*compiler (Assembler.)*

4. บันทึกไฟล์แล้วปิดหน้าต่างบันทึก ผู้อ่านควรตรวจสอบรายชื่อไฟล์ที่อยู่ภายใต้ไฟล์นี้ว่ามีไฟล์อะไรบ้าง

5. ลบไฟล์อ้อมเจ็คท์ที่มีอยู่โดยใช้คำสั่ง

\$ make clean

ในโปรแกรม Terminal เพื่อเปรียบเทียบหลังจากที่รันคำสั่ง make clean

6. ใช้คำสั่ง ls -la ใน Terminal ค้นหาไฟล์อ้อมเจ็คท์ main.o และ Lab6 ว่าถูกลบหรือไม่

7. ทำการแosoสเซมเบล main.s โดยใช้คำสั่ง make ในโปรแกรม Terminal และขอให้สังเกตวันเวลาของไฟล์ ต่างๆ

\$ make

8. ใช้คำสั่ง ls -la ใน Terminal เพื่อค้นหาไฟล์ชื่อ main.o และ Lab6 ว่ามีจริงหรือไม่

9. เรียกโปรแกรม Lab6 โดยพิมพ์

\$ ./Lab6

%% echo \$?

$$\begin{array}{l}
 R_1 (2) \rightarrow \begin{array}{c} 0 \\ 0 \\ 1 \\ 0 \end{array} \text{ (AND)} \\
 R_2 (4) \rightarrow \begin{array}{c} 0 \\ 1 \\ 0 \\ 0 \end{array} \\
 R_0 (0) \rightarrow \begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \end{array} \\
 \hline
 \text{ผล} R_0 = 0
 \end{array}$$

## F.5 กิจกรรมท้ายการทดลอง

### bitwise And

1. จงปรับแก้คำสั่ง ORR เป็นคำสั่ง AND ในโปรแกรม main.s และตรวจสอบผลการเปลี่ยนแปลงแล้วจึง อธิบาย + Bitwise OR ใน operation. Bitwise And ( bitwise AND กับมัน )
2. จงปรับแก้โปรแกรมใน main.s เป็นดังนี้ จดบันทึกผลการทดสอบและอธิบาย

```

1 .global main
2
3 main:
4 move R5, #1 R5=1
5
6 loop:
7 compare R4, #0 R4
8 ble end
9
10 branch less than or equal to
11 else:
12 mov R5, #2
13
14 end:
15
16 mov R0, R5
17
18 return
19

```

คำสั่งนี้จะทำงานอย่างไร

3. จงปรับแก้โปรแกรมใน main.s เป็นดังนี้ จดบันทึกผลการทดสอบและอธิบาย

```

.data
.balign 4
var1: .word 1
.text
.global main
main:

```

คำสั่งนี้ทำงานอย่างไร

## F.5. กิจกรรมท้ายการทดลอง

note:

```

MOV R1, #2
LDR R2, var1addr
STR R1, [R2]
LDR R0, [R2]
BX LR
var1addr: .word var1

```

DST  $\leftarrow$  [SRC]

LDR Load Register (โหลดค่า)

STR Store Register

[SRC]  $\rightarrow$  [DST]

bcd ; because what in memory

269

ก้าวเดิน步 2

#

(CMP)

compare  
two operands.

(BLE)

Branch If  
Less than  
or equal to

2. จะปรับแก้โปรแกรมใน main.s เป็นดังนี้ จดบันทึกผลการทดสอบและอธิบาย

```

.global main
main: r(move)
 MOV R5, #1
loop:
 CMP R4, #0 } ถ้า R4 ขนาดน้อยกว่าหรือเท่ากับ 0 ให้เข้า
 BLE end } ถ้า jump ไปที่ end ถ้ามากกว่า 0 ให้หัวใจต่อไป
else:
 MOV R5, #2 จะรักษา R5 = 2
end:
 MOV R0, R5 ผ่านมา R0 = R5
 BX LR

```

การ return 0;  $\rightarrow$  BX LR คุณสามารถ

จะรักษา R5 ขนาด = 1

ถ้า R4 ขนาดน้อยกว่าหรือเท่ากับ 0 ให้เข้า  
ถ้า jump ไปที่ end ถ้ามากกว่า 0 ให้หัวใจต่อไป

จะรักษา R5 = 2

ผ่านมา R0 = R5

untitled.S - [Unsaved] - VisUAL

File Help

New Open Save Settings Tools Emulation Complete Line Issues 7 0 Execute Reset Step Backwards Step Forwards

```

Reset to continue editing code
1 main MOV R5, #1
2
3 loop CMP R4, #0
4 BLE endl ← R4 <= 0 ถ้า jump/end
5 else MOV R5, #2
6
7 endl MOV R0, R5
8

```

| R0  | 0x1        | Dec | Bin | Hex |
|-----|------------|-----|-----|-----|
| R1  | 0x0        | Dec | Bin | Hex |
| R2  | 0x0        | Dec | Bin | Hex |
| R3  | 0x0        | Dec | Bin | Hex |
| R4  | 0x1        | Dec | Bin | Hex |
| R5  | 0x0        | Dec | Bin | Hex |
| R6  | 0x0        | Dec | Bin | Hex |
| R7  | 0x0        | Dec | Bin | Hex |
| R8  | 0x0        | Dec | Bin | Hex |
| R9  | 0x0        | Dec | Bin | Hex |
| R10 | 0x0        | Dec | Bin | Hex |
| R11 | 0x0        | Dec | Bin | Hex |
| R12 | 0x0        | Dec | Bin | Hex |
| R13 | 0xFFFFFFFF | Dec | Bin | Hex |
| LR  | 0x0        | Dec | Bin | Hex |

R4 → 0x0 ถ้า jump

Clock Cycles Current Instruction: 1 Total: 6

CSPR Status Bits (NZCV) 0 1 1 0

*กิจกรรมที่ 3*

9/ Emulator នឹង រាយរាយដែលបានបង្កើតឡើង។

The screenshot shows the VisUAL emulator interface. The assembly code window displays the following instructions:

```

1 var1 DCD 123456
2 var1addr DCD var1
3
4 main MOV R1, #5
5 MOV R2, #0x100
6 MOV R10, #0b0111 ; 7
7 MOV R11, #0b0110 ; 6
8 MOV R12, #0b0000 ; 0
9
10 LDR R2, =var1 ; load var1 Address to R2
11 LDR R3, =var1addr ; load var1addr Address to R3
12 LDR R4, [R3] ; load value of R3 -> R4
13 LDR R5, [R4] ; load value of R4 -> R5
14
15 STR R1, [R2] ; R1->[R2]
16 LDR R0, [R2] ; R0<-[R2]
17
18 ADD R12,R10,R11 ; Addition 12 = 7+6 ; R12=R10+R11
19 ORR R12,R10,R11 ; Bit wise OR 0b0111 = 0b0111 | 0b0110
20

```

The Registers window shows the following values:

| Register | Value      | Dec | Bin | Hex |
|----------|------------|-----|-----|-----|
| R0       | 0x5        | Dec | Bin | Hex |
| R1       | 0x5        | Dec | Bin | Hex |
| R2       | 0x100      | Dec | Bin | Hex |
| R3       | 0x104      | Dec | Bin | Hex |
| R4       | 0x100      | Dec | Bin | Hex |
| R5       | 0xE240     | Dec | Bin | Hex |
| R6       | 0x0        | Dec | Bin | Hex |
| R7       | 0x0        | Dec | Bin | Hex |
| R8       | 0x0        | Dec | Bin | Hex |
| R9       | 0x0        | Dec | Bin | Hex |
| R10      | 0x7        | Dec | Bin | Hex |
| R11      | 0x6        | Dec | Bin | Hex |
| R12      | 0x7        | Dec | Bin | Hex |
| R13      | 0xFF000000 | Dec | Bin | Hex |
| LR       | 0x0        | Dec | Bin | Hex |
| PC       | 0x38       | Dec | Bin | Hex |

The Memory dump window shows the following values:

| Type | Name     | Address (value of symbol) | Contents of memory location |
|------|----------|---------------------------|-----------------------------|
| Code | main     | 0x0                       | MOV R1, #5                  |
| Data | var1     | 0x100                     | 0x5                         |
| Data | var1addr | 0x104                     | 0x104                       |

The Status Bits window shows the following values:

| CSPR Status Bits (NZCV) | 0 | 0 | 0 | 0 | 0 |
|-------------------------|---|---|---|---|---|
|-------------------------|---|---|---|---|---|

ការអនុវត្ត

,data នូវការបង្កើតឡើងនៅលទ្ធផលនៃសម្រាប់ static.

.text នូវការបង្កើតឡើងនៅលទ្ធផលនៃសម្រាប់ dynamic នៅក្នុងអាជីវកម្ម.

RESET TO CONTINUE

```

1 var1 DCD 123456 } ← នូវការបង្កើតឡើងនៅលទ្ធផលនៃ static
2 var1addr DCD var1 } ← នូវការបង្កើតឡើងនៅលទ្ធផលនៃ dynamic
3
4 main MOV R1, #5
5 MOV R2, #0x100
6 MOV R10, #0b0111
7 MOV R11, #0b0110
8 MOV R12, #0b0000
9
10 LDR R2, =var1
11 LDR R3, =var1addr
12 LDR R4, [R3]
13 LDR R5, [R4]
14
15 STR R1, [R2]
16 LDR R0, [R2]
17
18 ADD R12,R10,R11
19 ORR R12,R10,R11
20

```

ការអនុវត្តន៍ជាព័ត៌មាននៃការអនុវត្តន៍នៃការបង្កើតឡើង

នូវការបង្កើតឡើងនៅលទ្ធផលនៃ static នៃ memory ដូចជា Declare word in memory.

} ជាដំឡើង value (Move) នៃ Register ទៅ CPU ដូចជា move register និង move memory.

} LDR (Load Register) (from memory or register)
 ពួកវានិយាយថា Address នៃ memory និង Register នឹងត្រូវបានផ្តល់ជាងគេ
 តាម var1 = 123456 // Address = 0x100
 LDR R2, =var1
 R2 នឹងត្រូវបានផ្តល់ជាងគេ 0x100 (បីនាទី Address var1)

} STR (Store Register) (to memory)
 ពួកវានិយាយថា Register នឹងត្រូវបានផ្តល់ជាង memory.

} Operation using CPU.

ADD (Addition)

ORR (Bitwise OR)

AND (Bitwise And)