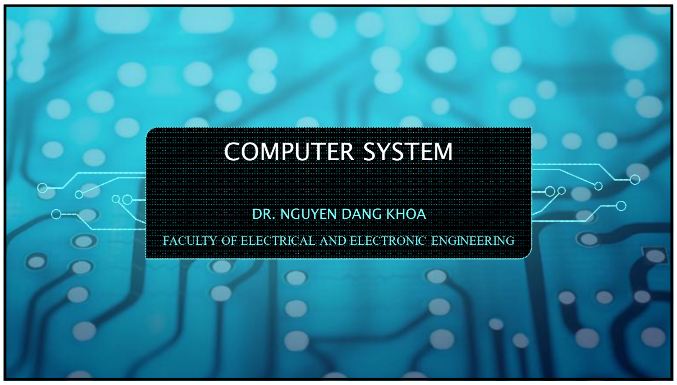
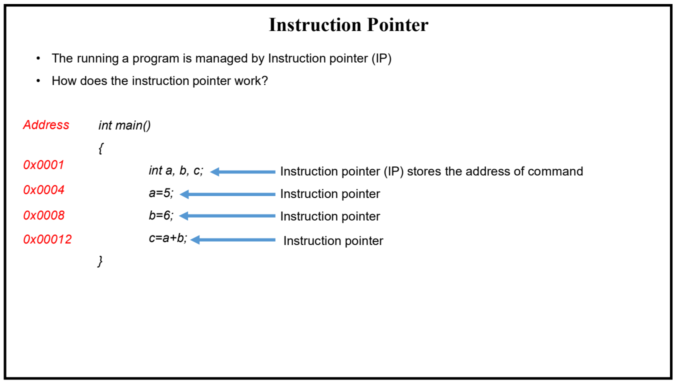


COMPUTER SYSTEM

DR. NGUYEN DANG KHOA

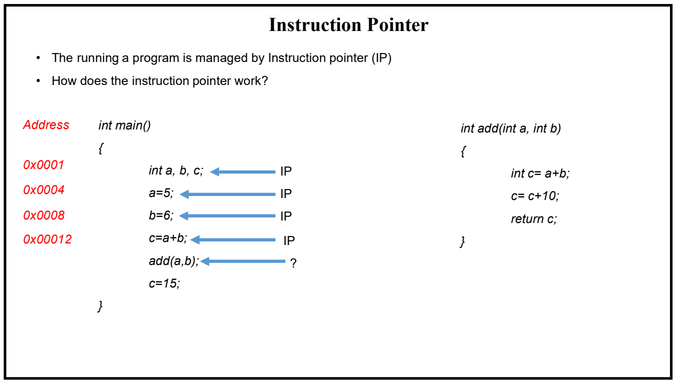
FACULTY OF ELECTRICAL AND ELECTRONIC ENGINEERING



Instruction Pointer

- The running a program is managed by Instruction pointer (IP)
- How does the instruction pointer work?

```
Address   int main()
          {
0x0001         int a, b, c;  <----  Instruction pointer (IP) stores the address of command
0x0004         a=5;  <-------  Instruction pointer
0x0008         b=6;  <-------  Instruction pointer
0x00012        c=a+b;  <-----  Instruction pointer
          }
```



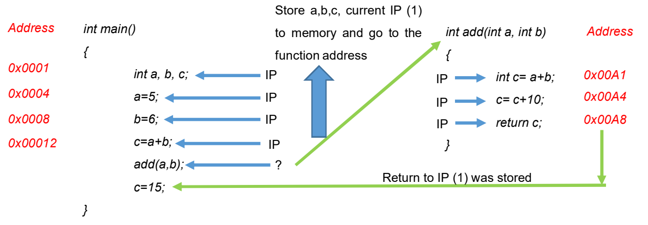
Instruction Pointer

- The running a program is managed by Instruction pointer (IP)
- How does the instruction pointer work?

```
Address   int main()                                 int add(int a, int b)
          {                                          {
0x0001         int a, b, c;  <----  IP                    int c= a+b;
0x0004         a=5;  <-------  IP                         c= c+10;
0x0008         b=6;  <-------  IP                         return c;
0x00012        c=a+b;  <-----  IP                    }
               add(a,b);  <----  ?
               c=15;
          }
```

Instruction Pointer

- The running a program is managed by Instruction pointer (IP)
- How does the instruction pointer work?



Store a,b,c, current IP (1) to memory and go to the function address

Address

```
int main()
{
0x0001    int a, b, c;  ← IP
0x0004    a=5;          ← IP
0x0008    b=6;          ← IP
0x00012   c=a+b;        ← IP
          add(a,b);     ← ?
          c=15;
}
```

Address

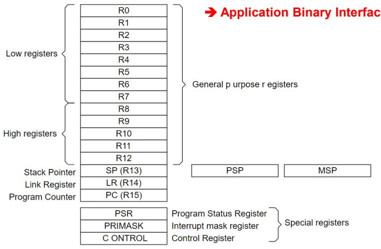
```
int add(int a, int b)
{
IP → int c= a+b;   0x00A1
IP → c= c+10;      0x00A4
IP → return c;     0x00A8
}
```

Return to IP (1) was stored

Instruction Pointer

- What proposes for IP?
  - **lr** and **sp** used for stack management (*link register, stack pointer*)
  - **pc** is the next instruction – we can use it to exit a function call (*program counter*)

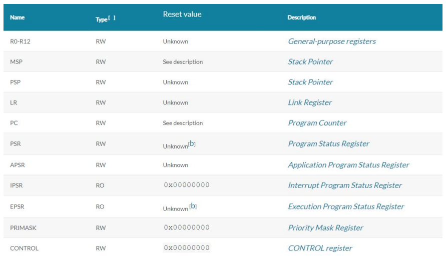
➔ **Application Binary Interface** (ABI) sets standard way of using ARM registers.



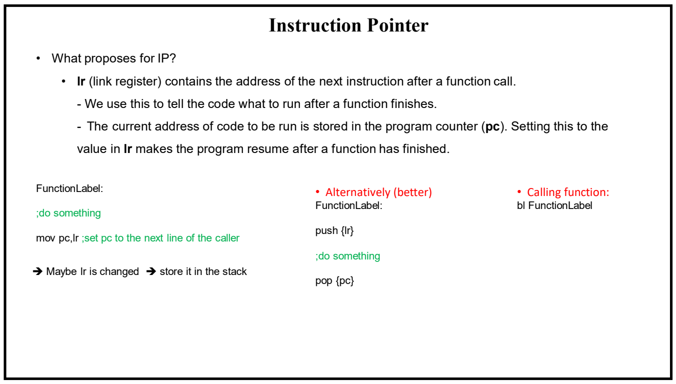
https://developer.arm.com/documentation/dui0497/a/CHDBIBGJ

Instruction Pointer

- What proposes for IP?
  - **lr** and **sp** used for stack management (*link register, stack pointer*)
  - **pc** is the next instruction – we can use it to exit a function call (*program counter*)



Name	Type[ ]	Reset value	Description
R0-R12	RW	Unknown	*General-purpose registers*
MSP	RW	See description	*Stack Pointer*
PSP	RW	Unknown	*Stack Pointer*
LR	RW	Unknown	*Link Register*
PC	RW	See description	*Program Counter*
PSR	RW	Unknown[b]	*Program Status Register*
APSR	RW	Unknown	*Application Program Status Register*
IPSR	RO	0x00000000	*Interrupt Program Status Register*
EPSR	RO	Unknown[b]	*Execution Program Status Register*
PRIMASK	RW	0x00000000	*Priority Mask Register*
CONTROL	RW	0x00000000	*CONTROL register*



Instruction Pointer

- What proposes for IP?
  - **lr** (link register) contains the address of the next instruction after a function call.
    - We use this to tell the code what to run after a function finishes.
    - The current address of code to be run is stored in the program counter (**pc**). Setting this to the value in **lr** makes the program resume after a function has finished.

```
FunctionLabel:
;do something
mov pc,lr ;set pc to the next line of the caller
```

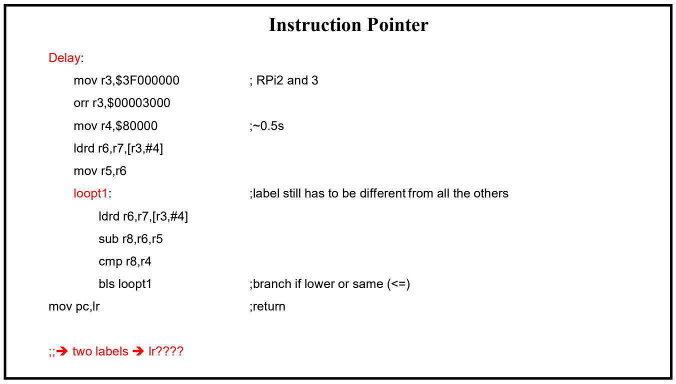
➔ Maybe lr is changed ➔ store it in the stack

- Alternatively (better)

```
FunctionLabel:
push {lr}
;do something
pop {pc}
```

- Calling function:

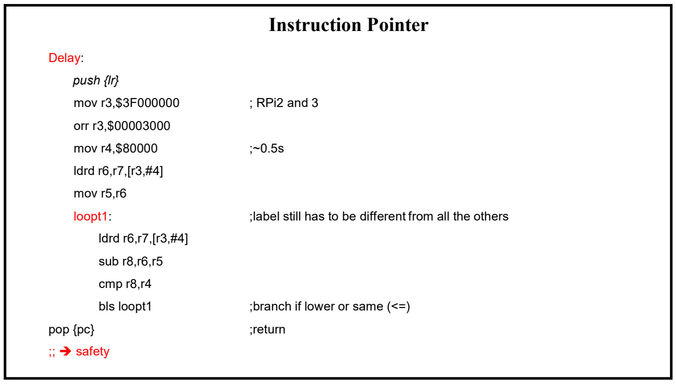
```
bl FunctionLabel
```



Instruction Pointer

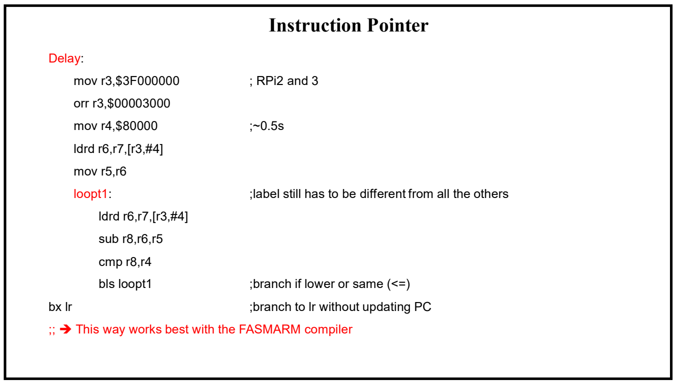
```
Delay:
    mov r3,$3F000000          ; RPi2 and 3
    orr r3,$00003000
    mov r4,$80000             ;~0.5s
    ldrd r6,r7,[r3,#4]
    mov r5,r6
    loopt1:                   ;label still has to be different from all the others
        ldrd r6,r7,[r3,#4]
        sub r8,r6,r5
        cmp r8,r4
        bls loopt1            ;branch if lower or same (<=)
mov pc,lr                     ;return

;;➔ two labels ➔ lr????
```



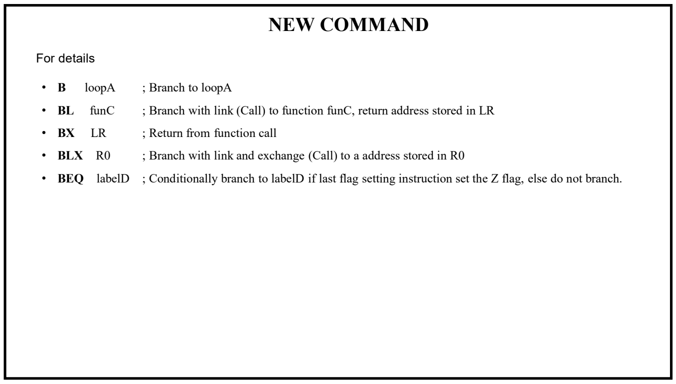
Instruction Pointer

```
Delay:
    push {lr}
    mov r3,$3F000000          ; RPi2 and 3
    orr r3,$00003000
    mov r4,$80000             ;~0.5s
    ldrd r6,r7,[r3,#4]
    mov r5,r6
    loopt1:                   ;label still has to be different from all the others
        ldrd r6,r7,[r3,#4]
        sub r8,r6,r5
        cmp r8,r4
        bls loopt1            ;branch if lower or same (<=)
pop {pc}                      ;return
;; ➔ safety
```



Instruction Pointer

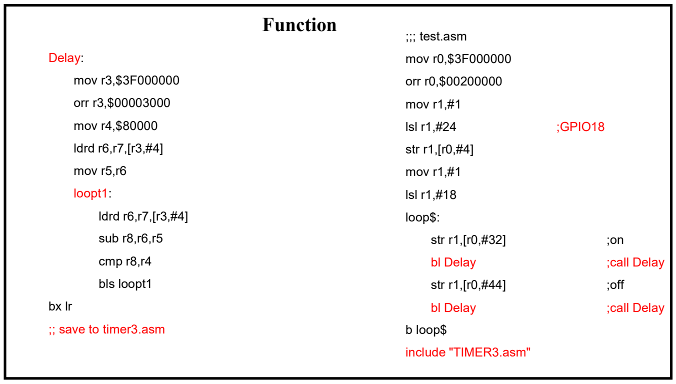
```
Delay:
     mov r3,$3F000000               ; RPi2 and 3
     orr r3,$00003000
     mov r4,$80000                  ;~0.5s
     ldrd r6,r7,[r3,#4]
     mov r5,r6
     loopt1:                        ;label still has to be different from all the others
          ldrd r6,r7,[r3,#4]
          sub r8,r6,r5
          cmp r8,r4
          bls loopt1                ;branch if lower or same (<=)
bx lr                               ;branch to lr without updating PC
;; ➔ This way works best with the FASMARM compiler
```



NEW COMMAND

For details

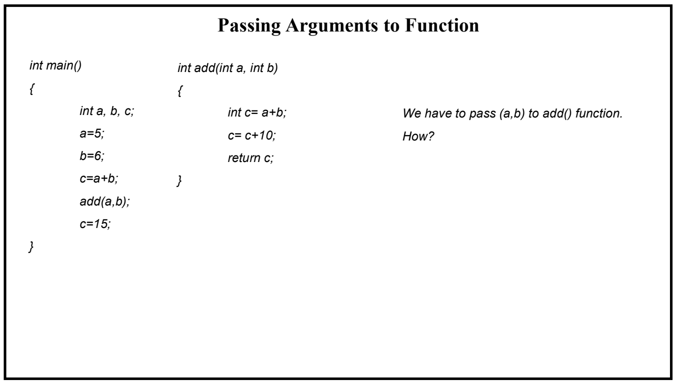
- **B** loopA ; Branch to loopA
- **BL** funC ; Branch with link (Call) to function funC, return address stored in LR
- **BX** LR ; Return from function call
- **BLX** R0 ; Branch with link and exchange (Call) to a address stored in R0
- **BEQ** labelD ; Conditionally branch to labelD if last flag setting instruction set the Z flag, else do not branch.



Function

```
Delay:
     mov r3,$3F000000
     orr r3,$00003000
     mov r4,$80000
     ldrd r6,r7,[r3,#4]
     mov r5,r6
     loopt1:
          ldrd r6,r7,[r3,#4]
          sub r8,r6,r5
          cmp r8,r4
          bls loopt1
bx lr
;; save to timer3.asm
```

```
;;; test.asm
mov r0,$3F000000
orr r0,$00200000
mov r1,#1
lsl r1,#24                    ;GPIO18
str r1,[r0,#4]
mov r1,#1
lsl r1,#18
loop$:
     str r1,[r0,#32]               ;on
     bl Delay                      ;call Delay
     str r1,[r0,#44]               ;off
     bl Delay                      ;call Delay
b loop$
include "TIMER3.asm"
```

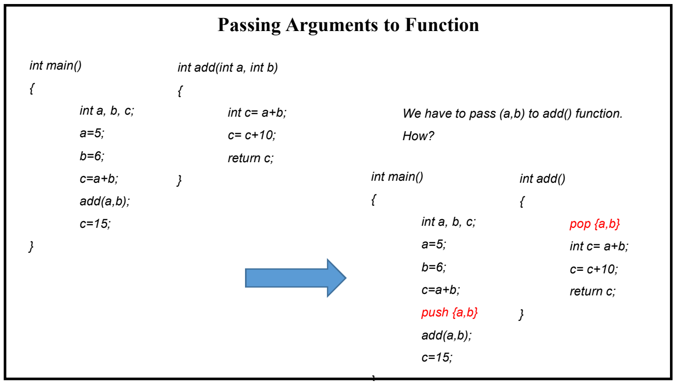


Passing Arguments to Function

```
int main()
{
        int a, b, c;
        a=5;
        b=6;
        c=a+b;
        add(a,b);
        c=15;
}
```

```
int add(int a, int b)
{
        int c= a+b;
        c= c+10;
        return c;
}
```

We have to pass (a,b) to add() function.
How?



Passing Arguments to Function

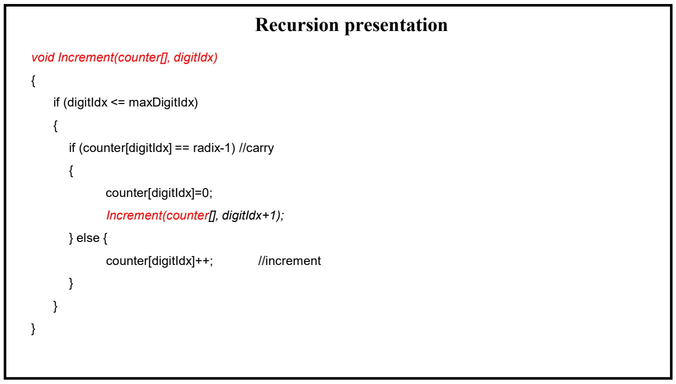
```
int main()
{
        int a, b, c;
        a=5;
        b=6;
        c=a+b;
        add(a,b);
        c=15;
}
```

```
int add(int a, int b)
{
        int c= a+b;
        c= c+10;
        return c;
}
```

We have to pass (a,b) to add() function.
How?

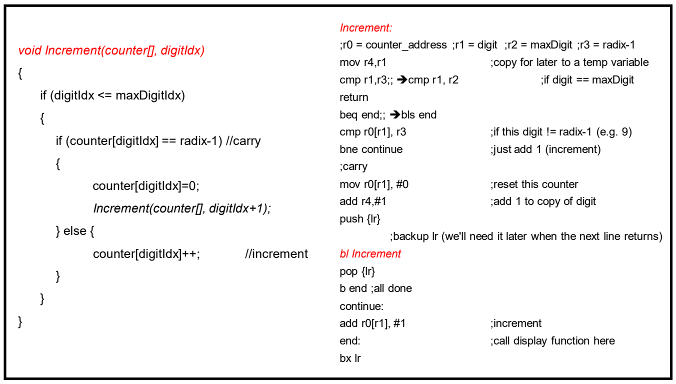
```
int main()
{
        int a, b, c;
        a=5;
        b=6;
        c=a+b;
        push {a,b}
        add(a,b);
        c=15;
```

```
int add()
{
        pop {a,b}
        int c= a+b;
        c= c+10;
        return c;
}
```



Recursion presentation

```
void Increment(counter[], digitIdx)
{
    if (digitIdx <= maxDigitIdx)
    {
       if (counter[digitIdx] == radix-1) //carry
       {
            counter[digitIdx]=0;
            Increment(counter[], digitIdx+1);
       } else {
            counter[digitIdx]++;        //increment
       }
    }
}
```



```
void Increment(counter[], digitIdx)
{
    if (digitIdx <= maxDigitIdx)
    {
        if (counter[digitIdx] == radix-1) //carry
        {
            counter[digitIdx]=0;
            Increment(counter[], digitIdx+1);
        } else {
            counter[digitIdx]++;        //increment
        }
    }
}
```

```
Increment:
;r0 = counter_address ;r1 = digit  ;r2 = maxDigit ;r3 = radix-1
mov r4,r1                       ;copy for later to a temp variable
cmp r1,r3;; ➔cmp r1, r2                 ;if digit == maxDigit
return
beq end;; ➔bls end
cmp r0[r1], r3                  ;if this digit != radix-1 (e.g. 9)
bne continue                    ;just add 1 (increment)
;carry
mov r0[r1], #0                  ;reset this counter
add r4,#1                       ;add 1 to copy of digit
push {lr}
          ;backup lr (we'll need it later when the next line returns)
bl Increment
pop {lr}
b end ;all done
continue:
add r0[r1], #1                  ;increment
end:                            ;call display function here
bx lr
```