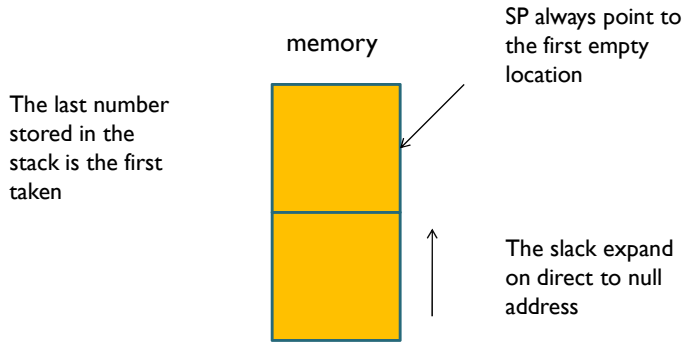
SUBROUTINES

- Stack
- JSR and RTS instruction operation
- using subroutine
- provide parameters to subroutine

Stack

- Stack: a memory part "last in first out" (LIFO) where data is stored temporarily
- Stack point (SP) operating post depreciate or pre amortize
- Applications:
  - Temporary depository to CPU variables and registers (PSH or PUL)
  - Subroutine calls (BSR or JSR and RTS)
  - Interrupt (SWI or RTI)

Stack



Pushing & Pulling

- Programmer can direct achieve slack content with instructions "push" and "pull"
- Push: locate data in stack at address that showed by SP, next depreciate the SP
  - ❑ Push instruction: PSHA, PSHB, PSHX and PSHY
- Pull: amortize SP, next take data from slack using address that showed by SP
  - ❑ Pull instruction: PULA, PULB, PULX and PULY

Stack Overflow & other Stack Errors

- Stack problem: the second most popular after the usage of "#"symbol
- Stack overflow- push too much until data change to other place with wrong sequence of PULL.
- Forget initialize SP. Normally, SP is set to the last byte of RAM
- Forget pull
- Pull too much
- Pull subroutine return address
- Push when in the subroutine but not pull before RTS

Subroutines – "black box"

- Why using subroutines?
- Reduce code for the repeat task
- Improve modular programming. Short code block easier to test than a long block.
- Easy to move program to other systems (portability).
- General format of subroutines:
- Each of the first line of subroutine start with one label. This label is used to call it.
- The last instruction in the subroutine must instruction of RTS.

Subroutine operation

- BSR (branch to subroutine) using relative mode.
- JSR (jump to subroutine) can jump anywhere in the 64K address space.
- With BSR or JSR, the value of PC located on the stack (LSB first).
- RTS (return to subroutine) take PC from on stack and jump to the address.

mnemonic	Function	DIR	EXT	INDX	INDY	INH	REL
BSR	Branch to subroutine						X
JSR	Jump to subroutine	X	X	X	X		
RTS	Return to subroutine					X	

Subroutine design

- Subroutine require to design as "black box'.
  - There are ways to give data to subroutine and what is done in the subroutine.
  - Require documentation so that anyone can use the subroutine by understand the data delivery methods and the function.
  - If some register changed, the content must be stored at the stack and take it back before out from subroutine.

Design Process

- Take time at the first writing program, you would be minimized a lot of entire time.
- Understand the problem
- Break the answer to the function
- Locate the function to design program
- Write program
- Test program

Subroutine operation

- The following is the summary of what happened when JSR or BSR execute to calling subroutine.
- Instruction address that follow JSR or BSR is stored on the stack.
- PC filled with subroutine address.
- RTS cause return address is taken from on the stack and stored to PC.
- Subroutine can be called from in the subroutine , but beware stack overflow.

Send data to subroutine

- Call and return register
  - The easiest way, have 2 registers 8 bit and 2 points 16 bit.
  - The amount of data sent is limited
- Through stack frame
  - Normally only used by HLL
  - Can be hazard. treatment mistake of stack damaging the return address.
- Memory space that set apart (Global variable)
  - Easy for one program, but lost "black box" concept when subroutine need local variable.
  - No reentrant

The usage of CPU Register

*Caller routine

```
LDAB    NUMBER
JSR     MUL8
STAB    HASIL
```

•Subroutine

```
MUL8    ASLB
        ASLB
        ASLB
        RTS
```

Error Handling with C-bit

```
*caller routine
        LDAB    NUMBER
'0' – '9'
        JSR     MUL8
        BCS     ERROR
        STAB    HASIL
*MUL8 subroutine
        ISNOT
MUL8    ASLB
        BCS     M8EXIT
        ASLB
        BCS     M8EXIT
        ASLB
M8EXIT RTS
```

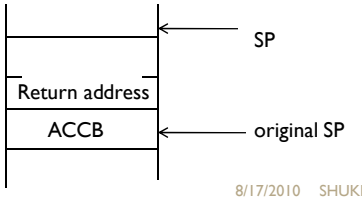
```
*ISDIGIT subroutine
        *check if A between
ISDIGIT  CMPA   #"0
         BL0    ISNOT
         CMPA   #'9
                BHI

         CLC
         BRA    EXIT
ISNOT    SEC
EXIT     RTS
```


The usage of Stack Frame

```
*Caller routine
   LDAB    NUMBER
   PSHB
   JSR     MUL8
   INS
   STAB    HASIL
```

```
*subroutine
MUL8 TSX
     LDAB 0, X
     ASLB
     ASLB
     ASLB
     RTS
```



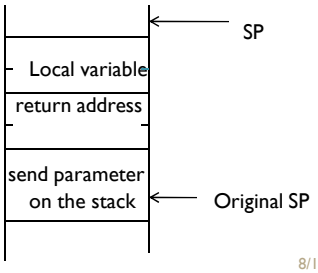
The usage of Stack Frame

*caller routine

```
        LDAB    NUMBER
        PSHB
        JSR     MUL8
        INS
        BCS     ERROR
        STAB    HASIL
```

*subroutine

```
MUL8    PSHX
        TSX
        LDAB    0, X
        ASLB
        BCS     M8EXIT
        ASLB
        BCS     M8EXIT
        ASLB
M8EXIT  PULX
        RTS
```



8/17/2010 SHUKRI - CAIRO UTM 2010 15

The usage of Global Variable

*Caller routine

*global data = input & output

```
        LDK     #VAR
        JSR     MUL8
```

*subroutine

```
MUL8    ASL     0,X
        ASL     0,X
        ASL     0,X
        RTS
```

8/17/2010 SHUKRI - CAIRO UTM 2010 16