Stack & Subroutines
Ch. 3, part 2

ECE331

Rev.S11

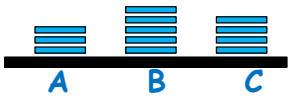
Outline

- Stack
  - concept
  - hardware
  - ASM instructions
  - examples
- Subroutines
  - concept
  - ASM instructions
  - examples
- Parameter passing techniques

Stack

- Stack = section of memory used for temporary storage
  - often used to store CPU register values before jumping to *subroutines*
  - has first-in/last-out (**FILO**) structure

    Example:



- 68HC12 definitions
  - "bottom_of_stack" = highest stack memory address
    - just "below" (higher address) starting point of the stack
  - "top_of_stack" = memory address of last stacked value
    - decreases as values are added to stack
    - addresses above top_of_stack are considered empty

- Stack can generally be any memory location
  - defined starting at $3F80 for lab evaluation board

Hardware and ASM Instructions

- Stack Hardware
  - Stack Pointer = 16b CPU register holding value of top_of_stack
    - initially set to bottom_of_stack
    - automatically decreases/increases value as items added/taken to/from stack
      - points to top (lowest address) filled stack location

- Stack ASM Instructions
  - LDS, load stack, set initial value of stack
    - EXAMPLE:

for lab evaluation board

  - PUSH (PSHA, PSHB, ..D, ..CCR, ..X, ..Y)
    - SP ← SP – 1, copy register data onto stack @ <SP>
  - PULL (PULA, PULB, ..D, ..CCR, ..X, ..Y)
    - copy stack data to register, SP ← SP + 1
  - Also: STS, INS, DES, TSX, TXS, TSY, TYS

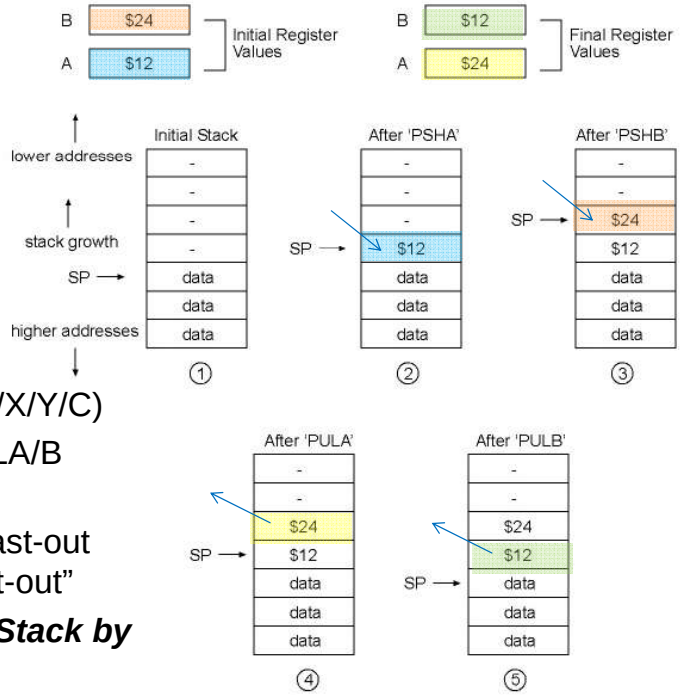
16b Push
- SP ← SP – 2
- <SP, SP+1>

16b Pull
- <SP, SP+1>
- SP ← SP + 2

order of instr. operations important to understanding how stack works

Stack Example

- ASM code executed



- Observations
  - SP points to "top of stack"
  - SP is decreased by 1 for PSHA/B (by 2 for PSHD/X/Y/C)
  - SP is increased by 1 for PULA/B (by 2 for PULD/X/Y/C)
  - Stack functions as first-in / last-out (FILO), same as "last-in, first-out"
  - ***Data is not removed from Stack by a PUL instruction***
    - data above SP is ignored, not deleted

Explaining Stack ASM Code

- What does the following code do?

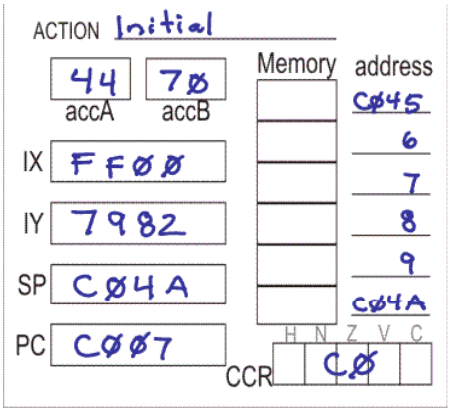
```
LDS   #$8000
PSHA
PSHB
PSHX
PSHY
TSX
LDAA  3,X
LDAB  5,X
```

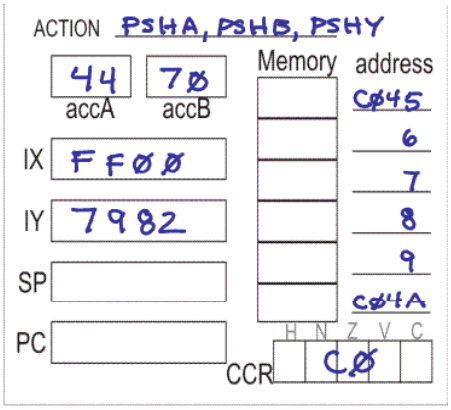
Initial Values			*Stack*		*Final Values*		
A	AA			7FF9	A		
B	BB			7FFA	B		
X	11	22		7FFB	X		
Y	F0	0F		7FFC	Y		
SP	80	00		7FFD	SP		
				7FFE			
				7FFF			
				8000			

Another Stack Example

op-code prog. address

- Show results after execution of PSHA
  PSHB
  PSHY





- What ASM code would restore accB (and affect nothing else)?

Option1

```
INS
INS
PULB
DES
DES
DES
```

Option2

Example Stack Code

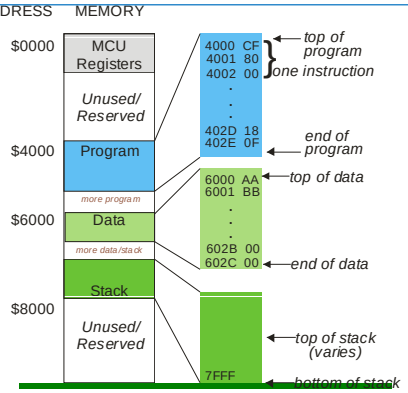
- .ASM code

label[1]	mnemonic[2] directive[1]	operand[3]	prog./data mem. addr.[4]	machine opcode[5]
	ORG	$6000	=00006000	
D_REC	FCB	$AA,$BB,$CC,$FF,$EE,$01	6000	AA BB CC FF EE 01
	FCB	$A1,$B2,$C3,$F4,$01	6006	A1 B2 C3 F4 01
	FCB	$81,$72,$63,$54,$45,$E5,$01	600B	81 72 63 54 45 E5 01
RESULT	RMB	$0A	6012 +000A	
D_CPY	RMB	$10	601C +0010	
	ORG	$4000	=00004000	
	LDS	#$8000	4000	CF 8000
	LDX	#D_REC	4003	CE 6000
	LDY	#D_CPY	4006	CD 601C
	LDAA	#$0	4009	86 00
	PSHA		400B	36
LOOP	LDAB	0,X	400C	E6 00
	STAB	0,Y	400E	6B 40
	CMPB	#$01	4010	C1 01
	BEQ	ENDLP	4012	27 05
	INX		4014	08
	INY		4015	02
	INCA		4016	42
	BRA	LOOP	4017	20 F3
ENDLP	PULB		4019	33
	PSHY		401A	35
	LDY	#RESULT	401B	CD 6012
	STAA	B,Y	401E	6A ED
	PULY		4020	31
	INCB		4021	52
	PSHB		4022	37
	CPX	#RESULT	4023	8E 6012
	BEQ	DONE	4026	27 05
	INX		4028	08
	INY		4029	02
	CLRA		402A	87
	BRA	LOOP	402B	20 DF
DONE	TBA		402D	18 0F
	END			

.LST file

```
Machine Code Upload Record (.S19 file)
S0030000FC
S1136000AABBCCFFEE01A1B2C3F40181726354457 3
S1136010E501000000000000000000000000000096
S10F6020000000000000000000000070
S1134000CF8000CE6000CD601C860036E6006B4099
S113401C10127050802422 0F33335CD60126AED51
S1124020315237 8E6012270508028720DF180FF0
S9030000FC

Notes:
YELLOW HIGHLIGHTS: show memory addresses
GREEN HIGHLIGHTS: show data for data memory
PINK HIGHLIGHTS: show reserved memory bytes in data memory, initiated with value 00
BLUE HIGHLIGHTS: show ASM program opcodes to be stored in program memory
```



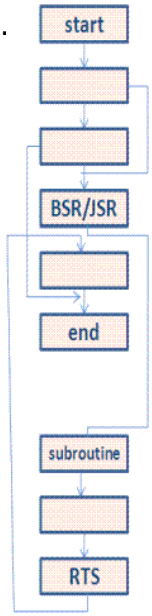
Subroutines

- Subroutine = independent program module performing a specific task
  - can be called repeatedly by main program or another subroutine
  - similar to a library function in higher level languages

- Advantages of subroutines (relative to branch loops)
  - less program memory than repeating multiple branch loops in a linear sequence
  - write once, use in multiple programs

Subroutine ASM Instructions

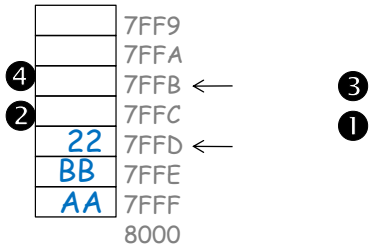
- BSR, branch to s/r (subroutine)
  - adjust PC by -127 to +128 → s/r must be close in program mem.
- JSR, jump to s/r
  - s/r can be anywhere in program memory



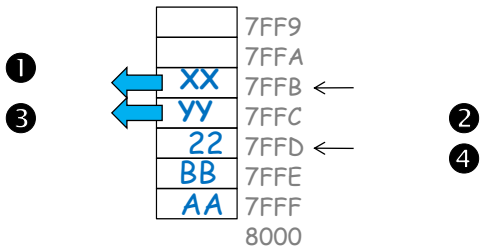
- Operation of BSR,JSR
  - current PC value (points to next instruction) automatically stored to STACK
  - PC value set to location of s/r
- RTS , return from s/r
  - restores PC value from STACK
  - s/r must end with SP pointing to exact position when s/r began
    - otherwise, it can't reload correct PC value from STACK

Automatic Subroutine Actions

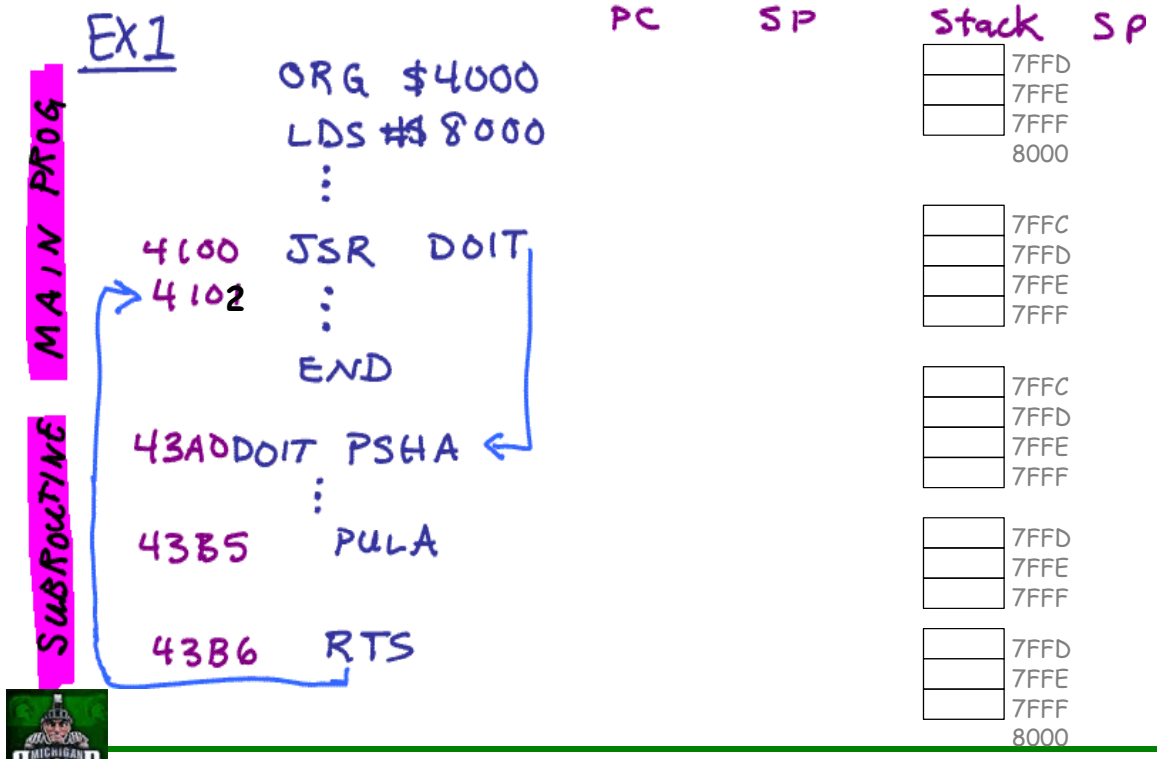
- BSR, JSR
  1. SP ← SP – 1
  2. PC_L → $STACK_{SP}$
  3. SP ← SP – 1
  4. PC_H → $STACK_{SP}$



- RTS
  1. $STACK_{SP}$ → PC_H
  2. SP ← SP + 1
  3. $STACK_{SP}$ → PC_L
  4. SP ← SP + 1



Subroutine/Stack Example

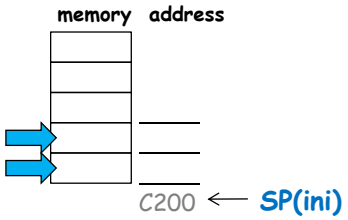


Examples

- Example 2: For .lst output below, s/r "bonk" begins at what program memory address?

```
                    prog. address .lst machine code
LDS     #$C200  $C0FD       CFC200
BRS     BONK    $C100       0720
...     ...     $C102       ...
```

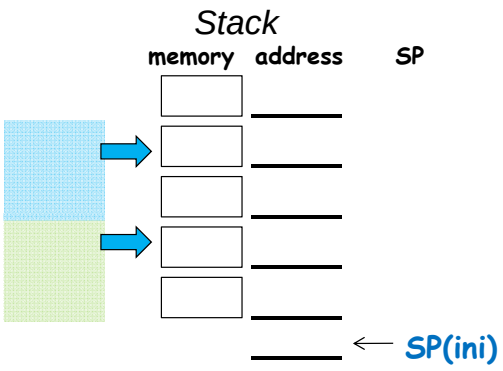
- Example 3: Fill in STACK when s/r "bonk" begins
  - Q1: stack addresses?
  - Q2: what "information" is put on stack?
  - Q3: what data values go in stack?
  - Q4: where is final SP?



Examples

- Example 4: Nested Subroutines
  - Illustrate STACK after NOP (no-operation) instruction
    - Q1: what is bottom_of_stack (initial SP value)?
    - Q2: what are remaining STACK addresses?
    - Q3: what information/values go to stack @ BSR?
    - Q4: what information/values go to stack @ JSR
    - Q5: where is SP @ NOP?

```
BB0      LDS  #$BAF
         LDY  #$111
         TYS
BB8      BSR  ONE
BBA       :
BD0  ONE JSR  TWO
BD3       :
C13  TWO NOP
          :
```



Subroutine Techniques

- Good Subroutine Is
  - Independent: does not rely on other programs or s/r that could change
  - Transparent: restores CPU registers to values before s/r
    - typically store CPU registers temporarily to STACK
  - Relocatable: data and code is location independent
    - do not rely on data in specific memory locations
    - best to use only variables defined w/in s/r; avoid DIR and EXT addr. modes
- Parameter Passing Techniques
  - How should you pass input/output data to/from a subroutine?
    - store data in CPU registers
    - store address of data in CPU registers
    - push data to STACK (tricky!)
    - **data memory is not a good option**; other program might change it
  - Describe parameter requirements in s/r comments
  - Always restore non-parameter CPU registers
    - return from s/r with values before s/r back in CPU registers