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:
  
  ![Example Diagram]

- 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
    
    16b Push
    - SP ← SP – 2
    - <SP, SP+1>
    16b Pull
    - <SP, SP+1>
    - SP ← SP + 2

    16b Push
    - SP ← SP – 2
    - <SP, SP+1>
    16b Pull
    - <SP, SP+1>
    - SP ← SP + 2

    Also: STS, INS, DES, TSX, TXS, TSY, TYS

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?

<table>
<thead>
<tr>
<th>LDS</th>
<th>PSHA</th>
<th>PSHB</th>
<th>PSHX</th>
<th>PSHY</th>
<th>TSX</th>
<th>LDAA 3,X</th>
<th>LDAB 5,X</th>
</tr>
</thead>
<tbody>
<tr>
<td>#$8000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Initial Values  | Stack  | Final Values |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A AA</td>
<td></td>
<td>A</td>
</tr>
<tr>
<td>B BB</td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>X 11</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Y 22</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>SP 80</td>
<td></td>
<td>SP</td>
</tr>
</tbody>
</table>

Initial Stack | After PSHA | After PSHB |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Data is not removed from Stack by a PUL instruction
Another Stack Example

- Show results after execution of
  - PSHA
  - PSHB
  - PSHY

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

**Option 1**
- INS
- INS
- PULB
- DES
- DES

**Option 2**
- INS
- INS
- PULB
- DES
- DES

---

Example Stack Code

- **ASM code**
- **.LST file**

---

Stack-Subroutines p.7
ECE 331, Prof. A. Mason
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

**EX1**

```assembly
ORG $4000
LDS #8000

4100 JSR DOIT
4102 :
END

43A0 DOIT PSH A
43B5 PULA
43B6 RTS
```

Stack-Subroutines p.11

ECE 331, Prof. A. Mason
Examples

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

<table>
<thead>
<tr>
<th>proq. address</th>
<th>.lst machine code</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDS</td>
<td>#$C200</td>
</tr>
<tr>
<td>BRS</td>
<td>BONK</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

```
LDS #$C200  $C0FD  CFC200
BRS BONK    $C100  0720
...
CFC200      0720...
$C0FD       $C100  $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?

**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?
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