EECE 218
Microcontrollers

Subroutines, addressing modes
Subroutines

Subroutine call instruction – in ‘caller’:

JSR address ; Jump to S/R

Return instruction – in subroutine

RTS ; return from S/R

How do they work?

JSR: saves address of next instruction (the ‘return address’)

RTS: restores the saved address (into PC) to jump back

Where is it saved? → STACK
The Stack

- A special section of the memory, accessed via the S(tack)P(ointer) and specific instructions

Note:
1. Stack ‘grows’ towards lower addresses
2. SP points to last element placed into the stack
The Stack

- **JSR** saves the return address onto the stack:
  - $SP \leftarrow SP - 1$
  - $[SP] \leftarrow RTN_{Lo}$
  - $SP \leftarrow SP - 1$
  - $[SP] \leftarrow RTN_{Hi}$

- **RTS** restores return address from stack:
  - $PC_{Hi} \leftarrow [SP]$
  - $SP \leftarrow SP + 1$
  - $PC_{Lo} \leftarrow [SP]$
  - $SP \leftarrow SP + 1$
Stack and subroutines

- **Note:**
  - Subroutines can be nested (Main → SR1 → SR2)
  - Subroutines can be recursive (SR → SR)
    - e.g. fact(N) = if N == 0 then 1 else N*fact(N-1)
  - Stack rule: **last in / first out**
  - Stack can be used to save and restore temporary data
    - Placing data onto stack: ‘PUSH’ (PSH_)
    - Removing data from stack: ‘PULL’ (PUL_)
    - With _ being: A,B,D,X,Y, and C (CCR)
    - Use LOAD/STORE with different addressing modes (next)
  - Subroutine **must** ensure that the SP points to the same location as when it was entered!

- In HC12: X,Y,PC can also be used in a way similar to SP (i.e. move data, and update pointer) -> variation on indexed addressing mode
Indexed addressing modes

- Auto pre/post-decrement/increment
  - *auto*: automatically updates the index register (X, Y, SP, PC) during operation
  - *pre/post*: before/after data transfer
  - *increment/decrement*: increases/decrease the index register

Example: auto pre-decrement indexed addressing
  - Before transferring data, decreases the content of the index register (‘PUSH’ of the data transfer is ‘write’)

Indexed addressing modes

- Auto pre/post-decrement/increment – notation
  - \text{n,-r} auto pre-decrement
  - \text{n,+r} auto pre-increment
  - \text{n,r-} auto post-decrement
  - \text{n,r+} auto post-increment
  - \text{n = 1..8} (but you can transfer bytes/words/etc).
  - \text{r = X,Y,PC,SP}

\begin{align*}
\text{PSHA} & \rightarrow \text{STAA 1,-SP} \\
\text{PULX} & \rightarrow \text{LDX 2,SP+}
\end{align*}
Examples

Our copy program:

LDX #$1100
LDY #$1200
L: LDAA 1,X+
    STAA 1,Y+
CPX #$1200
BNE L

Clear page:

LDX #$1100
L: CLR 1,X+ ; write 0
    CPX #$1200
BNE L

???:

LDX #$1200
LDY #$1200
L: LDAA 1,Y+
    STAA 1,-X
CPX #$1100
BNE L
Other indexed addressing modes

- Accumulator offset – unsigned 8/16 bit in acc!
  - A, r
  - B, r
  - D, r
  
  r can be X, Y, SP, PC

Data is found at address \([A+r]\)

Example: If X has $1000, B has $F2

LDAA B,X : A ← [$10F2]
Other indexed addressing modes

- 16-bit offset indexed-indirect
  
  \[[n,r]\]  
  \(r\) can be \(X, Y, SP, PC\)

  \(n\ is a 16-bit constant -32,768..+65535\)

  Address of data is found at address \([n+r]\), then data is accessed.
  (INDIRECT!)

Example: If \(X\ has \$1000,\)

\[
\text{\$1010:}\text{\$1011} have \$12;\$00
\]

\text{\textbf{LDAA} \[\$10,X\] will}

1.) Access location \$1010:\$1011

2.) Fetch address of data (i.e. \$1200)

3.) Fetch data (content of \$1200) into A

\(A \leftarrow [n+X]\)
Other indexed addressing modes

- Accumulator D offset indexed-indirect
  \[ [D,r] \quad r \text{ can be } X, Y, SP, PC \]

  Address of data is found at address \([D+r]\), then data is accessed. (INDIRECT!)

Example: If X has $1000, D has $0010
\n\[ [$1010:$1011] \text{ have } $12;$00 \]

\[ \text{LDAA } [D,X] \text{ will} \]
1.) Access location $1010:$1011
2.) Fetch address of data (i.e. $1200)
3.) Fetch data (content of $1200) into A

\[ A \leftarrow [[n+X]] \]
Interesting use indirect/indexed addressing mode

- Suppose D has 0,2,4
- Example code:
  
  ```
  JMP [D,PC]
  <DataWord1>
  <DataWord2>
  <DataWord3>
  ```

  will jump to locations pointed to by the selected DataWord!

Remember C/Java:

```java
switch(exp) {
  case ONE: stmt1;
  case TWO: stmt2;
  case THREE: stmt3;
}
```

Technique is called: jump table