EECE 218
Microcontrollers

Introduction to addressing modes
What is ‘addressing mode’?

**Def:** How the CPU ‘finds’ the data.

Example:

```
LDAA $1100 ← The 16-bit address of the data
```

**EXTENDED addressing mode:**

the operand has the 16-bit address of data.

+: can point to anywhere in memory

-: fixed (can’t change it)

If an address needs to change from execution to execution → store the address in a place where it can be changed….like a register! (Remember PC?)
Addressing mode(s)

- INDEXED: Address in index register (X or Y)
- OK, but how to load the address into the index register?
- Method 1:
  » Load from memory, using EXTENDED addressing

LDX $1200
$1200: 10
$FE 12 00
$1201: 00

will load $1000 into X (from locations $1200,1201)

Formally: (1st byte into ‘H’ byte

X ← [$1200]:[$1201] 2nd byte into ‘L’ byte)
Addressing mode(s)

- Method 2:
  - Load the value (the constant) $1000 itself into X
    
    ```
    LDX #$1000
    $$CE 10 00
    ```
    
    # Just notation, changes the meaning of the mnemonic

    **IMMEDIATE** addressing mode: the data is in the operand. (it is an ‘immediate’ value)

    Trick: different opcode for EXT and IMM addressing modes!

    OK, back to our problem:
    
    ```
    LDAA $1000
    LDX #$1001
    ```
    
    Now add to A the value pointed to by X ← How?
Addressing mode(s)

- INDEXED addressing mode: address of data is in an index register (X/Y)
  Many variants for IDX mode, for now only the simplest cases: constant offset
  How we write it: \texttt{nn,X} where \texttt{nn} can be:
    - 5-bit signed number (-16..+15) \texttt{\leftrightarrow WILL USE THIS}
    - 9-bit signed number (-256..+255)
    - 16-bit signed number (-32,768..+65,536)
  For our example: \texttt{0,X}
  And the instruction: \texttt{ADDA 0,X}
    \$AB 00
Addressing mode(s)

- INDEXED addressing mode – with offset:
  ALWAYS supply an offset
  (if offset = 0 you can write: ,X)

Meaning:
  address of data = X + offset

I.e. ADDA 0,X means:
  A ← A + [X+0] : add to A the content of the
  memory cell whose address is X’s content plus the
  offset (=0 here).

For different addressing modes, different opcodes
are used. (HC12 also uses secondary opcodes,
called ‘post-bytes’).
Addressing mode(s)

Our code thus far:

- **LDAA** $1000
- **LDX** #$1001
- **ADDA** 0,X

Now what? ---- Go to the next location (with X)!

- **INX** : increment (add 1 to) X
- **$08**

Operand? → In the ‘instruction’ itself!

→ INHERENT addressing mode
Addressing mode(s)

Next: Repeat the operation? (do again add, inx, until we reach the end).

An instruction: \( \text{JMP addr : PC} \leftarrow \text{addr} \) 
\( \$06 \) \( hh \) \( ll \)

Not good --- always jumps!

We need:

“’goto’ if we are not at the end of the list”, i.e.

\[ \begin{align*}
A & \leftarrow A + [X+0] \quad \text{(adda)} \\
X & \leftarrow X + 1 \quad \text{(inx)} \\
X & =?= \$1005 \quad \text{check and jump}
\end{align*} \]
Code (cont.)

- In machine code programming
  - 1.) Pose a question (X =?= $1005)
  - 2.) Select between two choices and transfer control.

Pose a question: is X equal to $1005?

Instruction:

CPX #$1005  Compare X and the $8E 10 05 constant $1005

Where is the answer?

In the CCR (cond. code. reg.)!
Code (cont.)

CCR bits: S X H I N Z V C

The lower bits indicate various things about the result: (if bit is 1)
N(egative) - Result is negative
Z(ero) - Result is zero
(o)V(erflow) - There was an arithmetic overflow
C(array) - There was a carry

Here: the result of the COMPARE is like the result of a subtraction, i.e. if X == $1005, then the result is equal to Z(ero)
Z = 1 if X == $1005, Z = 0 otherwise.
Code (cont.)

Depending on answer, transfer control.
In machine code: **conditional branch** instruction

```
BNE rel : Branch if not equal
$26 rr (Z is not set)
```

\( rr = \text{the distance between} \)
\( (1) \text{the address of the next instruction, and} \)
\( (2) \text{the address of the target instruction} \)

**RELATIVE** addressing mode

\( rr = 0: \text{next instruction, } rr = $FE: \text{the branch!} \)
Code (almost there)

B6 10 00   LDAA   $1000
CE 10 01   LDX    #$1001
AB 00     ADDA   0,X
08       INX
8E 10 05   CPX    #$1005
26 F8     BNE    RR

RELATIVE addressing modes: SIGNED!!
  rel8     -256..+255 locations
  rel9     -512..+511
  rel16    Any address in 64K memory space
## Code (there!)

<table>
<thead>
<tr>
<th>Code</th>
<th>Instruction</th>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>B6 10 00</td>
<td>LDAA</td>
<td>$1000</td>
<td></td>
</tr>
<tr>
<td>CE 10 01</td>
<td>LDX</td>
<td>#$1001</td>
<td></td>
</tr>
<tr>
<td>AB 00</td>
<td>ADDA</td>
<td>0,X</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>INX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8E 10 05</td>
<td>CPX</td>
<td>#$1005</td>
<td></td>
</tr>
<tr>
<td>26 F8</td>
<td>BNE</td>
<td>RR</td>
<td></td>
</tr>
<tr>
<td>6A 00</td>
<td>STAA</td>
<td>0,X</td>
<td></td>
</tr>
</tbody>
</table>

**Last instruction:** store A into memory

STAA 0,X  Indexed!
Some more code, instructions

Other way to store:

STAA $1005 : [$1005] ← A

Store A using EXT addressing

STAA $1005 vs STAA 0,X:

(1) 3 bytes vs 2 bytes, (2) IDX works because X happens to have the right content. (!!!)
Some more code, instructions

For the lab – new instructions:

- **LDAB** #$04 : Load into B the value $4
- **DECB** : Decrement (sub 1 from) B
- **SWI** : Software interrupt (special instruction to ‘stop’ computer)
- **LDD** #$1234 : Load into D the value $1234 (2byte value!!!)
Some more code - Examples

1. Fill page with address of next page:
   Use X and Y (pointer to target, address/value)
   
   \[\text{init } X,Y\]
   \[X+0 \leftarrow Y \quad \text{: STY using 0,X}\]
   \[X \leftarrow X+2; Y \leftarrow Y+2\]
   if \(X \neq \text{end of page}\) do it again

2. Copy page:
   Use X and Y (pointer to source/target)
   
   \[\text{init } X,Y\]
   \[Y+0 \leftarrow [X+0]\]
   \[X \leftarrow X+1; Y \leftarrow Y+1\]
   If \(X \neq \text{end of page}\) do it again
Subroutines

- Concept from high-level languages: code that is written once, but used many times.
- In lab ‘printf’ subroutine: can print a 0-terminated sequence of ASCII codes

Use:
  - LDD  #address_of_string
  - LDX  $EE88  ; Addr of subroutine
  - JSR  0,X   ; Subroutine call

Code for last instruction: $15 00

Note: LDD and JSR are 1 byte opcode + 2 bytes operand (16 bit address)